MARECHAL Anthony
MARECHAL Benoit

Application et Système d'information

Rapport du projet « Cinéma et sémantique »

Encadrant : Lylia Abrouk et David Gross-Amblard

MASTER 2 MULTIMEDIA - 2007/2008

Table des matières

INTRODUCTION

Dans le cadre du module Application de l'université de Bourgogne nous devons mettre en place un projet utilisant une base de données XML comportant des informations sémantique et une interface web permettant l'interrogation de ces données.

Ce rapport à pour but de modéliser ce projet grâce à UML et aux modèles offert par UWE puis à donner les explications techniques concernant la réalisation de ce projet.

Ce rapport est ainsi composé de deux parties principales. La première concerne la modélisation de notre application issue de notre cours d'Application et la deuxième concerne la partie technique issue du cours de Système d'information.

La partie « Application » comprend ainsi quatre sous parties. La première sous partie concerne les diagrammes des cas d'utilisations servant de fondement aux modèles UWE. La deuxième sous partie détail le modèle conceptuel, en donnant les explications concernant les choix réalisés pour le concevoir. La troisième sous partie détail quant à elle le modèle de navigation et donne des explications du passage du modèle conceptuel à ce modèle de navigation en précisant notamment quelles sont les classes navigable. Et pour la dernière sous partie nous montreront quelles solutions ont été retenues au niveau du modèle de présentation et nous fourniront des copies d'écran des pages a fin de comparaison.

La partie « Système d'information » comprend également quatre sous parties. La première sous parties recadre le sujet par un résumé de la solution que nous proposons. La deuxième sous partie est une analyse de fonctionnalités mise en place. La troisième sous parties rappel l'interface graphique du projet aux travers de copies d'écran. La dernière sous partie est un ensemble de jeux de tests de notre application.

PARTIE APPLICATION

1. Diagrammes des cas d'utilisations (« Use Case »)

Pour pouvoir modéliser notre application Web avec la méthode UWE basé sur UML nous devons concevoir les Use Case de notre site Web regroupant ces fonctionnalités et définir quels acteurs peuvent interagir avec ces fonctionnalités.

Ainsi, nous avons modélisé notre application aux travers de plusieurs diagrammes Use Case, le premier est le Use Case Global qui modélise l'ensemble des fonctionnalités et des acteurs de notre site Web. Nous avons ensuite découpé en package les différentes fonctionnalités afin d'obtenir une meilleur cohérence et indépendance en regroupant les éléments proches d'un point de vue sémantique tout en minimisant les relations nécessaire entre les packages. Nous avons alors pu créer trois diagrammes Use Case précisant quelles sont les fonctionnalités que notre application Web pourrait implémenter. Ces fonctionnalités ont été choisies en fonction du sujet du projet et d'une rapide étude des possibilités généralement offerte par les sites internet traitant du cinéma [1].

1.1. Use Case Global

L'interêt de ce Use Case Global (*Fig. 1*) est d'avoir une vision générale mais commplète des acteurs et des fonctionnalités que nous pourrions implémenter dans notre application.

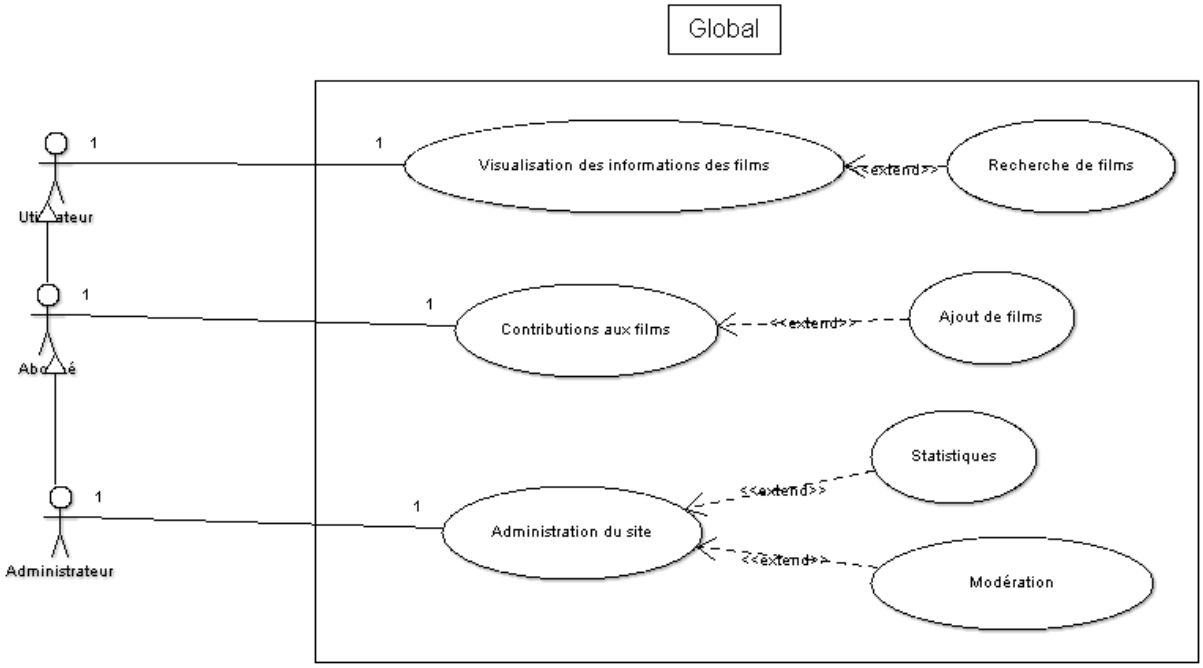


Fig. 1 : Use Case Global

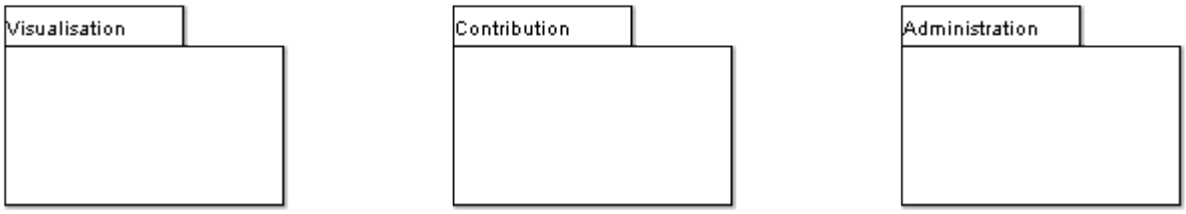
Les trois acteurs *Administrateur* qui hérite d'*Abonné* lui-même héritant des possibilités d'un *Utilisateur* représente toutes les personnes pouvant interagir avec notre site Web.

Les Utilisateurs représentent toute personne accédant à notre site Web mais n'étant pas un membre enregistré. Ces Utilisateurs ont accès à tout ce qui concerne la visualisation des informations de notre site vite ont peut ainsi qu'il on un accès en lecture à notre base de données concernant les films et toute les informations qui gravite autour des films. Les Utilisateurs ont également le droit à la recherche de films afin de visualiser directe le film recherché.

Les Abonné représentent tout internaute s'étant préalablement inscrit sur notre site internet. Ceux-ci ont la possibilité de contribuer en ajoutant des films ou en contribuant aux informations des films (ontologies, notes, commentaires). On peut ainsi considérer que les Abonnés ont, eux, un accès d'écriture sur notre base de données concernant les films et les informations qui y sont liées.

Finalement, l'acteur Administrateur a les droits d'administration du site. Il s'agit plus précisément d'un accès aux statistiques du site et de modération des contributions apportées par les Abonnés.

On peut ainsi apercevoir, à travers de se Use Case, trois fonctionnalités principale dans lesquelles toutes les sous-fonctionnalités peuvent être contenues, nous pouvons alors segmenter les diagrammes en trois packages :



Ces packages nous permettrons d'obtenir une plus grande indépendance et garantir une meilleur cohérence entre les acteurs et les possibilités qui leurs sont offertes. Nous allons maintenant détailler les fonctionnalités regroupées dans chacun de ces packages.

1.2. Use Case Visualisation

Le Use Case « Visualisation » (*Fig. 2*) contient les fonctionnalités consultables par les acteurs de notre application.

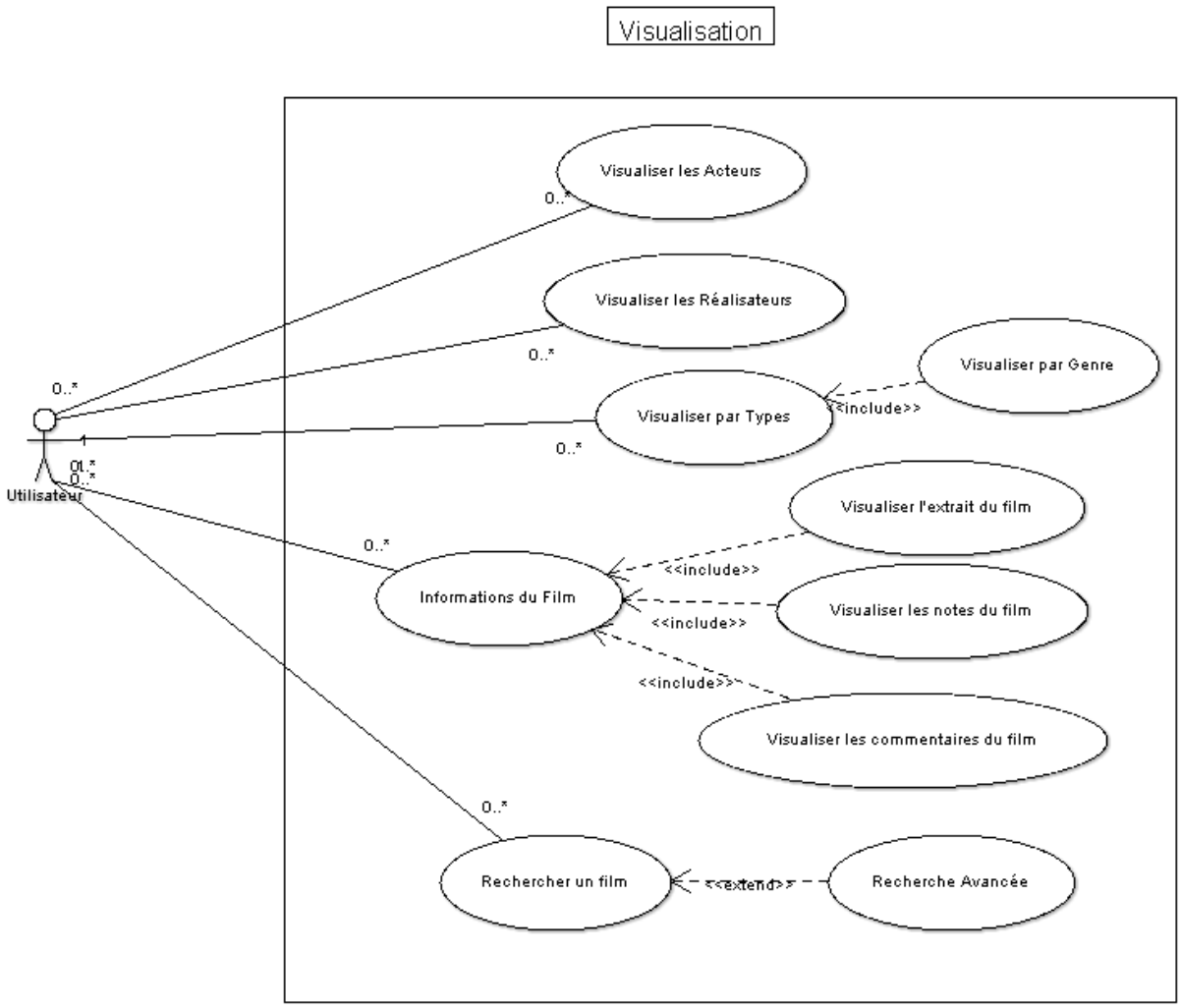


Fig. 2 : Use Case Visualisation

On peut constater ici que les utilisateurs peuvent consulter toutes les informations des films comme les extraits, les notes et les commentaires du film. Ils ont également la possibilité de rechercher les films simplement par une recherche par mots clés ou plus spécifiquement grâce à une recherche avancée.

L'accès aux films leur est également offert à travers la listes des acteurs, des réalisateurs, des types puis des genres de films puisque ceux-ci sont catégorisés en type puis genre (exemple : type = comédie, genre = romantique).

1.3. Use Case Contribution

Ce Use Case « Contribution » (*Fig. 3*) comporte les possibilités qu'un utilisateur enregistré appelé Abonné peux réaliser sur le site web. Ces possibilités, en plus des possibilités de visualisation et recherche de l'information héritées de l'Utilisateur, concerne tout ce qui touche à la contribution à l'information du site.

Fig. 3 : Use Case Contribution

Dans ce diagramme nous retrouvons en premier lieu l'acteur Utilisateur qui doit s'authentifier afin de devenir un Abonné. L'Abonné aura ensuite la possibilité de se déconnecter.

Les fonctionnalités des Abonnés concernent donc la contribution aux informations d'un film tel que noter, commenter, ajouter une ontologie entre des informations du film et modifier cette ontologie. Il peut, plus globalement, contribuer au site en ajoutant un film. Les possibilités de rectification et suppression des informations le concernant son compte sont également présente car obligatoire selon la loi française.

1.4. Use Case Administration

Le Use Case « Administration » (*Fig. 4*) de notre application est réservé aux supers utilisateurs, le webmaster disposera généralement de ces droits.

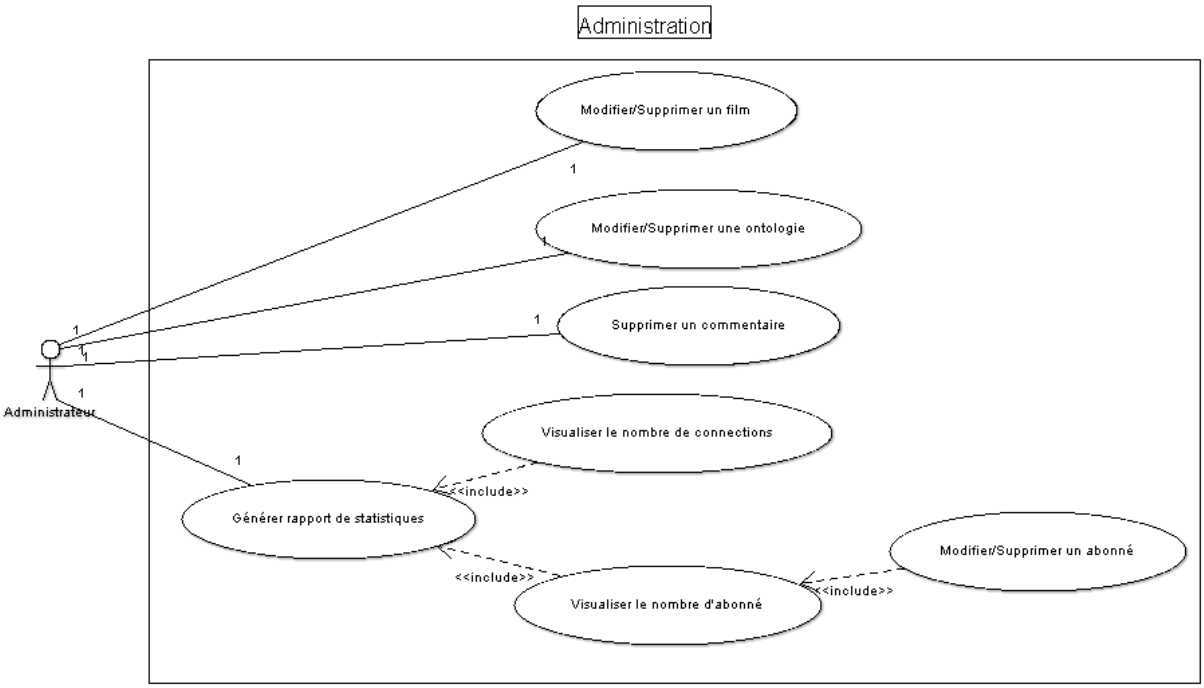


Fig. 4 : Use Case Administration

Les fonctionnalités offerte par notre site internet pour son administration concerne les aspects de modération (rectification ou suppression) des contributions effectuées par les Abonnés c'est-à-dire des ajouts de films, des ontologies et des commentaires. Les notes attribuées aux films ne pouvant être rectifié par le super utilisateur celui-ci ne devant pas dicter son avis à la communauté d'utilisateurs.

L'administrateur pourra également générer des rapports de statistiques d'audience pour connaitre l'évolution du succès ou non de son application et à des fins de veille notamment concernant la charge allouée aux serveurs support du site. Ces rapports de statistiques comprenne le nombre de connections d'Utilisateur sur le site, et aussi le nombre d'Abonné au site avec la possibilité de modérer le compte d'un abonné (afin par exemple de supprimer le compte des abonné ajoutant des films inexistant ou laissant des commentaires ne respectant pas la charte du site).

1.5. Synthèse

Cette première étape de modélisation par des Uses Cases nous permette de connaitre précisément le type de fonctionnalités qui pourraient être disponible pour un site internet concernant les films. Ces fonctionnalités n'ont pas toutes été implémentées dans le projet que nous rendons puisque certaines d'entre-elles ne répondent pas à un besoin de démonstration technologique particulier.

Ainsi, notre projet ne permet pas la distinction entre Utilisateur et Abonné, il n'y a donc pas de fonctionnalité d'authentification/déconnection ni de modification/suppression de compte. Les fonctionnalités d'administration n'ont également pas été implémentées puisque, technologiquement, elles ne présentaient pas d'intérêt particulier où alors les techniques qui auraient pu être

utilisées pour mettre en place ces fonctionnalités ont déjà été implémentées dans le projet (notamment au niveau des possibilités de contribution).

2. Modèle conceptuel

Pour la réalisation du modèle conceptuel nous avons dû trouver les classes et les attributs qui doivent être utilisé puis déterminer les associations entre ces classes. Deux modèles conceptuels ont été mit en place. Le premier est appelé « modèle conceptuel théorique » : il est directement lié aux Uses Cases précédemment réalisés. Le deuxième modèle est celui qui à réellement été implémenté dans notre projet.

2.1. Modèle conceptuel théorique

Le modèle conceptuel élaboré (*Fig. 5*) comprend toutes les informations nécessaires à l'implémentation des fonctionnalités issue des Uses Case précédent. Nous aurions pu le scinder en deux vues différentes : une vue Film qui aurait pu comprendre la classe Film et les classes gravitant autour de celle-ci et une vue Abonné qui aurait comprit les interactions possibles des Abonnés sur les informations des films. Les classes interagissant avec l'Abonné et les informations des films étant « Notes », « Commentaires », « Films » et « Abonné ». Cependant il nous est apparut finalement plus clair de conserver qu'une seule vue à partir de laquelle les interactions entre l'Abonné et les informations auxquelles il peut contribuer soit confondu avec la vue du film et des informations lui étant liées puisque toutes les classes utilisées dans ces deux vues auraient été commune.

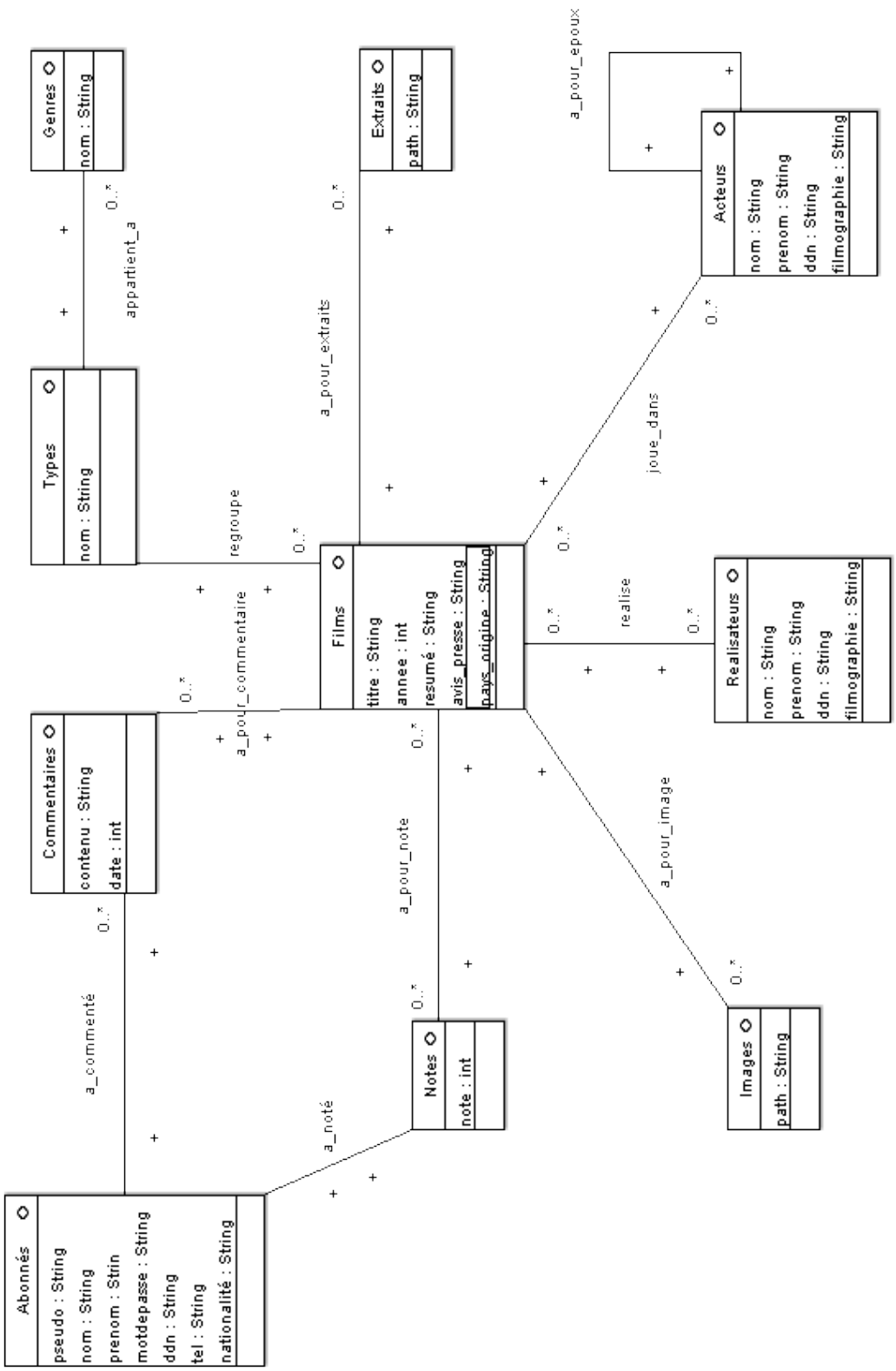


Fig. 5 : Modèle Conceptuel Théorique

Nous pouvons ainsi trouver dans ce diagramme les classes suivantes :

- Films : contient le titre, l'année, le résumé, l'avis de la presse, et le pays d'origine du film. Autour de cette classe des informations lui sont liées comme les commentaires, le type puis sont genre, ses extraits/images ainsi que les réalisateurs/acteurs.

- Abonnés : il s'agit de la classe contenant les informations fournies par l'Utilisateur lors de son enregistrement sur le site et qu'il peut rectifier à tout moment. Ces informations sont : un pseudo, son nom, prénom, date de naissance, son mot de passe, numéro de téléphone, et sa nationalité.

Les relations entre les classes sont simples puisque la table film est au cœur de ce diagramme et toutes les tables qui lui sont liés ont la même forme. Il s'agit d'association de multiplicité 0..* à 1, un Film possédant plusieurs notes, images, extraits, acteurs, réalisateurs et commentaires. Seule la classe « Abonnés » est excentrée et les associations avec cette classe concerne les classes Notes et Commentaires.

Une association récursive concernant la table acteur vers la table acteur à été modélisée pour représenter l'ontologie « a pour époux ». L'hypothèse faite ici est que les acteurs/actrices ont pour époux uniquement des actrices/acteurs (exemple : Brad Pitt et Angelina Joli).

2.2. Modèle conceptuel pratique

Ce modèle (*Fig. 6*) se concentre sur les classes qui ont été implémentées dans notre projet.

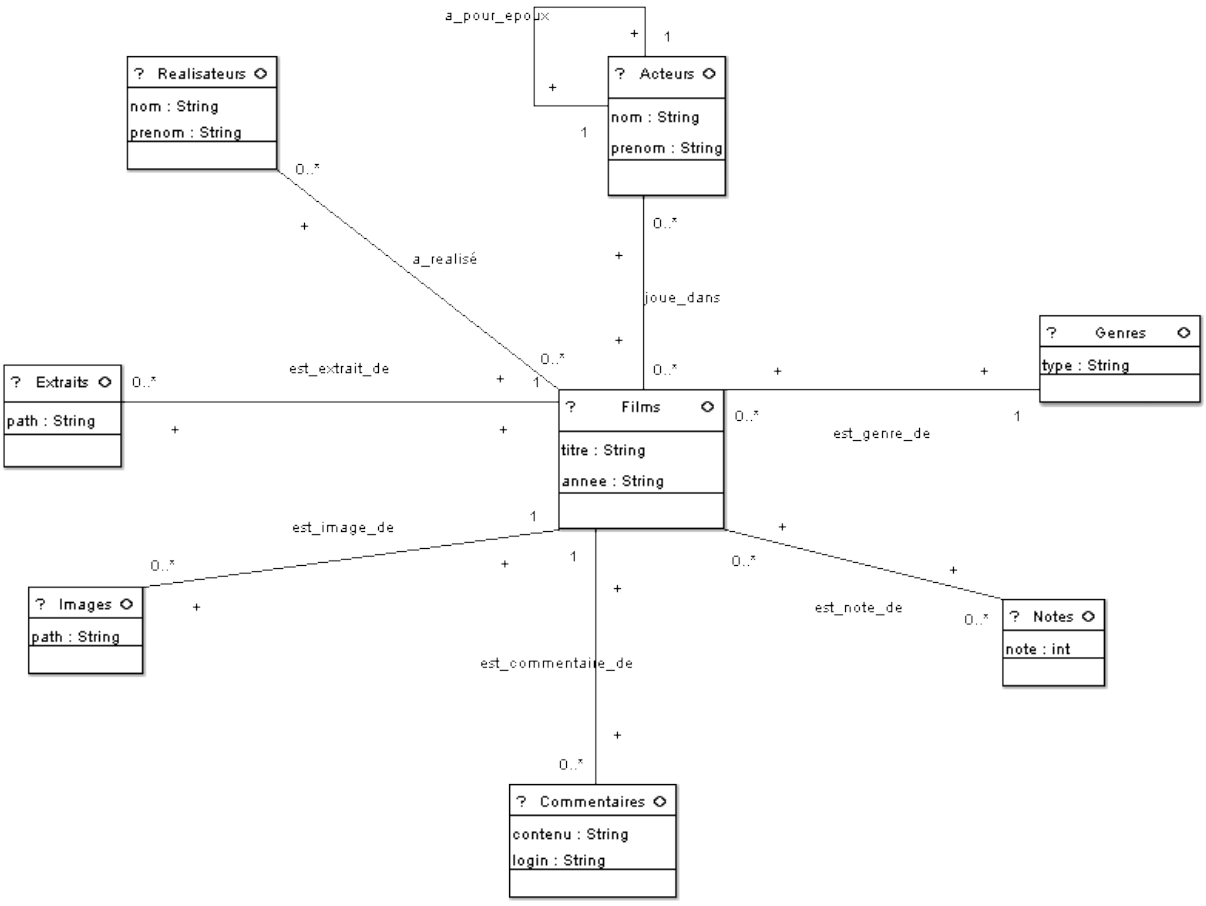


Fig. 6 : Modèle Conceptuel Pratique

On constate ici, comme nous l'avons expliqué lors de la modélisation avec les diagrammes Use Case, que la classe Abonné n'est pas présente. Certains attributs ont également été supprimés dans la table Film notamment lorsqu'ils ne présentaient pas d'intérêt particulier dans leur implémentation.

3. Modèle de navigation

Le diagramme de navigation découle du modèle conceptuel. Là encore deux diagrammes viendront illustrer notre modélisation avec un diagramme théorique et un diagramme pratique qui représentera la navigation retenu pour notre projet.

Pour la conception de ces diagrammes plusieurs conventions du passage Conceptuel à Navigationnel ont été utilisées et seront expliquées.

3.1. Modèle Navigationnel Théorique

Ce modèle navigationnel théorique (*Fig. 7*) représente l'espace de navigation et la structure navigationnel de l'application web telle qu'elle pourrait être implémentée.

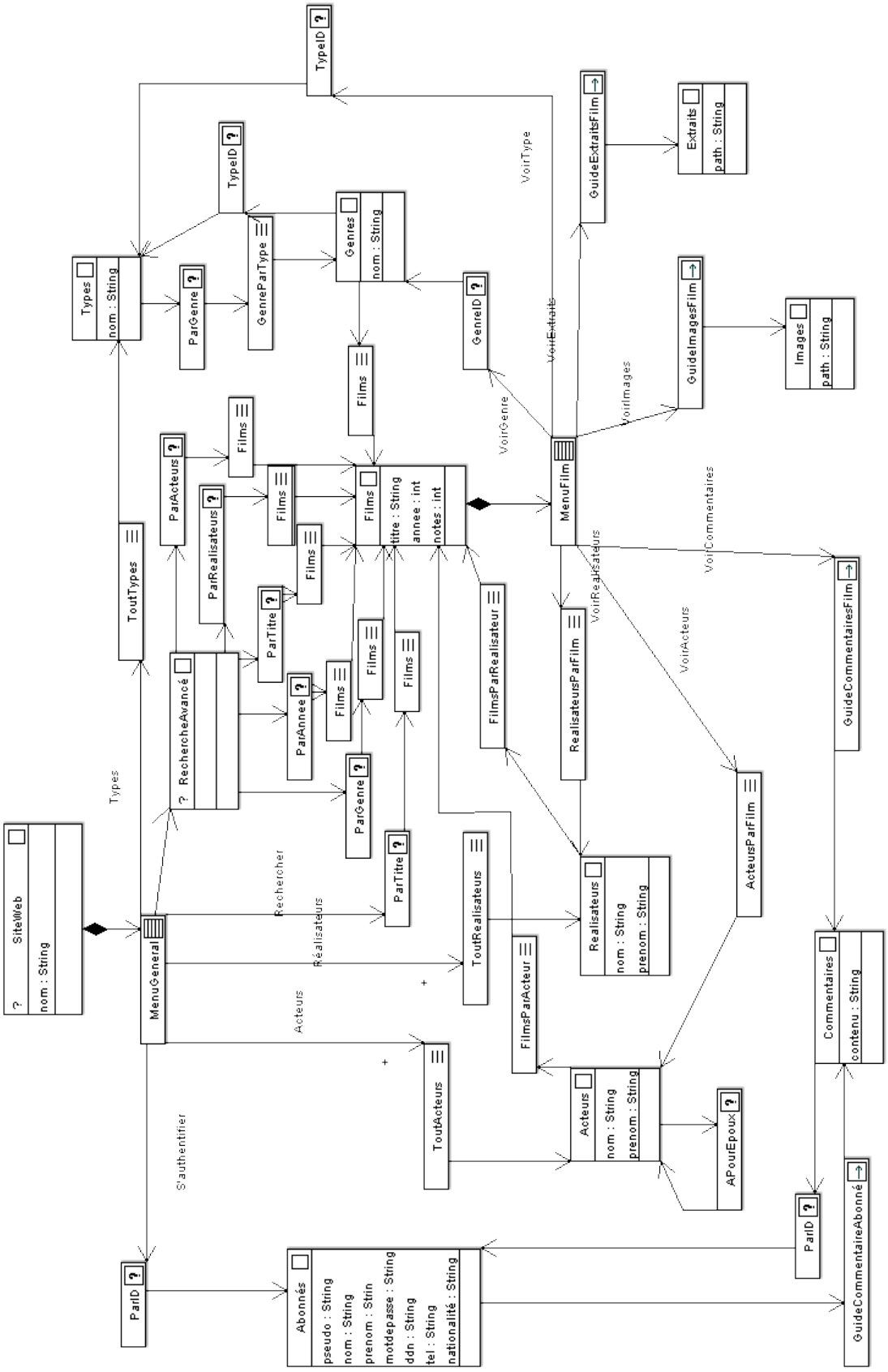


Fig. 7 : Modèle Navigationnel Théorique

Plusieurs règles ont été choisies pour concevoir ce modèle, ainsi :

- Les requêtes (query) ne pointant pas sur un index sont des requêtes qui ne donneront pas une liste de résultat mais un seul ce qui permet l'accès directe à la classe. Ces requêtes ce nomme généralement XxxxID avec ID signifiant que nous possédons l'identifiant de l'instance de la classe Xxxx se qui nous permet d'y accéder sans passer par une liste de résultat.

- Une convention utilisée par les créateurs d'UWE[2] est de créer un XOR entre une requête qui pointe sur une liste de résultat (un index) et cette même requête qui pointe directement sur la classe sans passer pour une liste de résultat dans le cas où un seul résultat est retourné. Cette convention n'a pas pu être implémentée dû à des limitations dans ArgoUWE.

- Les indexs nommés XxxxsParYyyy avec Xxxxs le nom d'une classe au pluriel et Yyyy le nom d'une classe au singulier, ne comprenne pas de requête (query) les précédents comme les créateurs d'UWE[2] (section 4.2.2) le suggère.

Ces conventions étant établies, nous pouvons commenter le passage Conceptuel / Navigationnel par rapport à notre application, ainsi :

- La classe Note est la seule classe non navigable. L'attribut note de cette classe se retrouve dans la classe Film.

- Une nouvelle page « RechercheAvancée » à été créée. Cette page permettra, lors de la création du modèle de présentation grâce à ArgoUWE d'avoir une page contenant les champs de formulaire adéquate.

- Deux menus « MenuGeneral » et « MenuFilm » ont été modélisés. Le MenuGeneral est le menu de notre application depuis l'accueil, il permet d'accéder à une recherche simple, une recherche avancée, la liste de tous les réalisateurs, acteurs, et types de film. Un accès permettant l'authentification des utilisateurs est également possible. Le « MenuFilm », quant à lui, permet l'accès aux réalisateurs et acteurs du film, ainsi qu'aux images, commentaires et extraits et ce grâce à un guide tour. L'accès aux films d'un même genre par une requête GenreID depuis le MenuFilm aboutie à la classe genre directement, cette classe Genre étant en faite une liste de film d'un certain genre.

- L'abonné pourra disposer d'un accès aux commentaires qu'il a donnés sur un film à travers d'un guide tour.

3.2. Modèle Navigationnel Pratique

Le modèle navigationnel (Fig. 8) implémenté dans notre projet est le suivant :

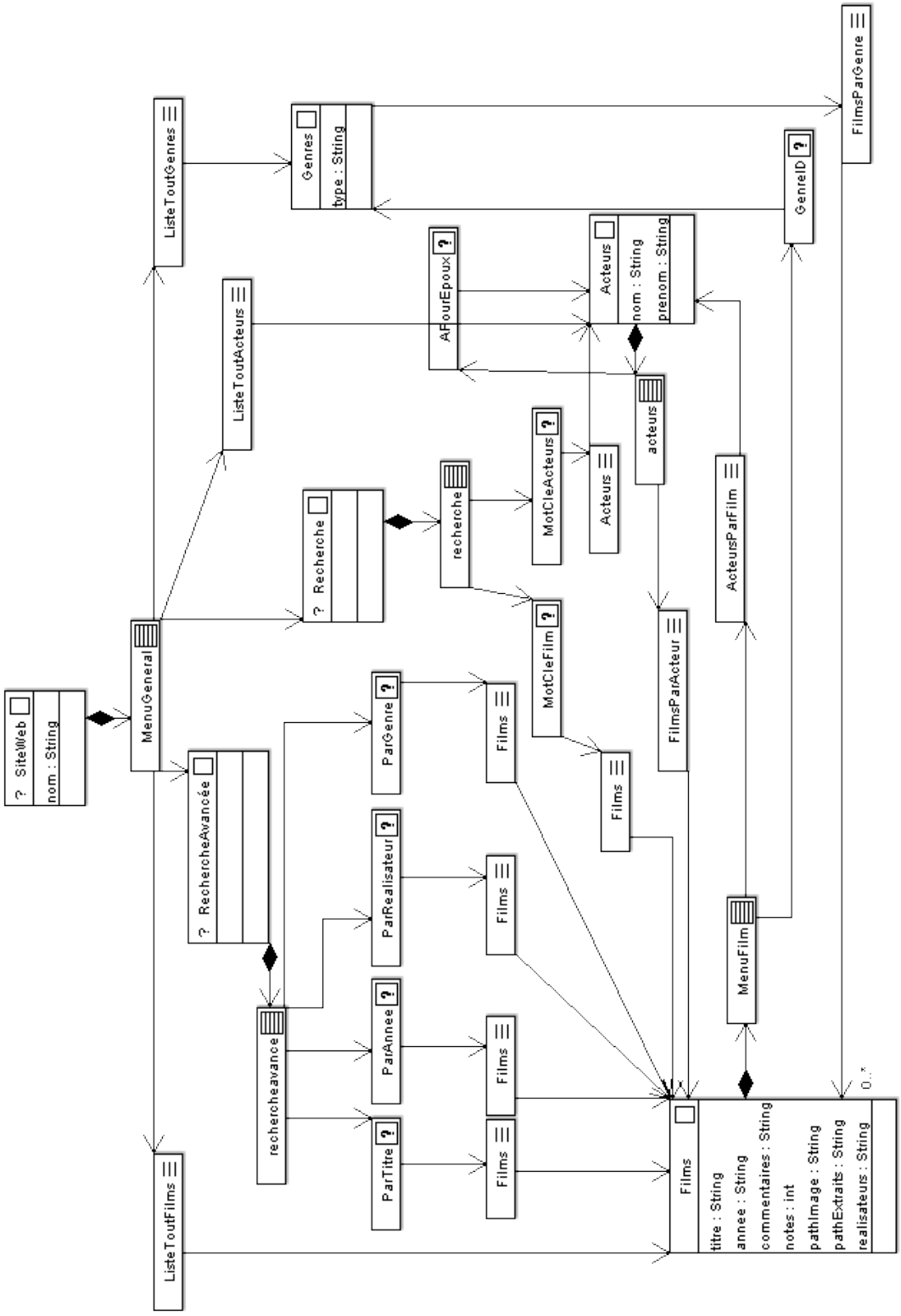


Fig. 8 : Modèle Navigationnel Pratique

Les conventions de passage Conceptuel / Navigationnel sont les mêmes que précédemment, auxquelles on pourra ajouter celle-ci :

- les indexs nommés ListeToutXxx, avec Xxx le nom d'une classe, signifie que cet index liste l'intégralité des instances de la classe. Par exemple, comme on peut

le voir sur ce schéma, depuis le « MenuGeneral », il est possible de lister tout les acteurs.

Les explications entre le passage de notre modèle conceptuel pratique à ce modèle de navigation pratique sont les suivantes :

- Les classes non navigables sont : Réalisateurs, Images, Extraits et Notes.

- Tous les attributs de ces classes se retrouvent dans la classe film.

- Contrairement au modèle théorique, dans notre projet, un film ne peut pas avoir plusieurs images mais seulement une, idem pour les extraits.

- Deux pages ont été créées. La première pour la « RechercheAvancée » et la deuxième pour la « Recherche ». En réalité il n'y a pas réellement de page Recherche nécessaire mais nous avions besoin, depuis une requête par mot clé (un seul champ texte (textfield)) mais accédant à deux classes différentes (Film et Acteur) puisque les résultats sont de ces deux types d'où cette solution.

- Il est également possible de lister l'intégralité des films depuis le MenuGeneral (cf. ListerToutFilm).

4. Modèle de présentation

Les modèles de présentations exposés dans les parties qui suivent sont issus du modèle de navigation pratique, des copies d'écran de notre projet pratique étant disponible afin de comparaison. Ces modèles de présentation ont été créés grâce à ArgoUWE qui offre des possibilités limitées quant aux UIElement disponible. L'avantage est qu'ArgoUWE génère les modèles de présentation de façon cohérente avec le modèle de navigation d'où il provient.

4.1. Modèle de présentation UWE

Le premier modèle de présentation (*Fig. 9*) est la page d'accueil de notre projet. Celle-ci contient le menu à partir duquel les informations peuvent être affichées.

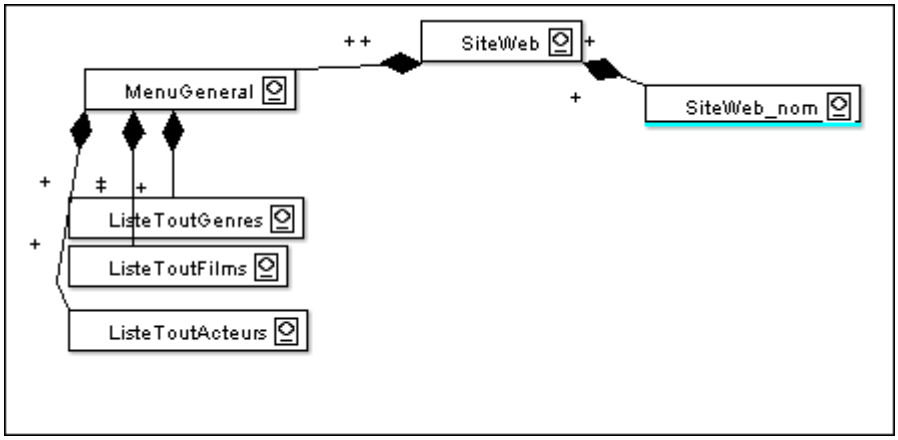


Fig. 9 : Modèle de Présentation - Accueil

Le deuxième modèle de présentation (*Fig. 10*) est la page de recherche.

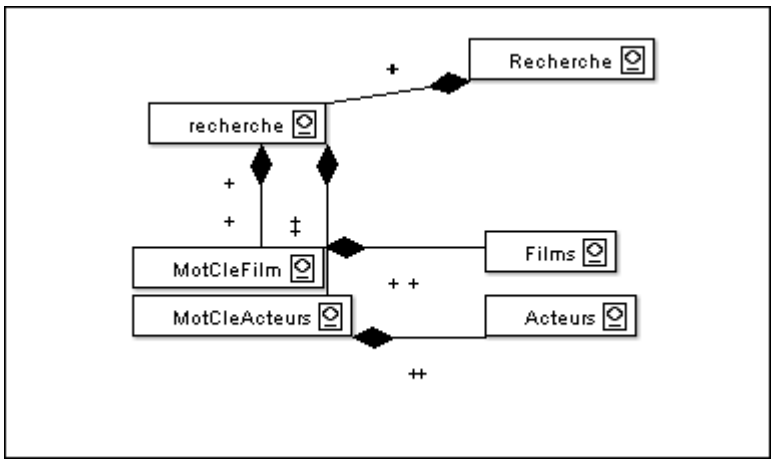


Fig. 10 : Modèle de Présentation - Recherche

Le troisième modèle de présentation (*Fig. 11*) est la page de recherche avancée. Celle-ci contient les champs de formulaire possible pour effectuer cette recherche.

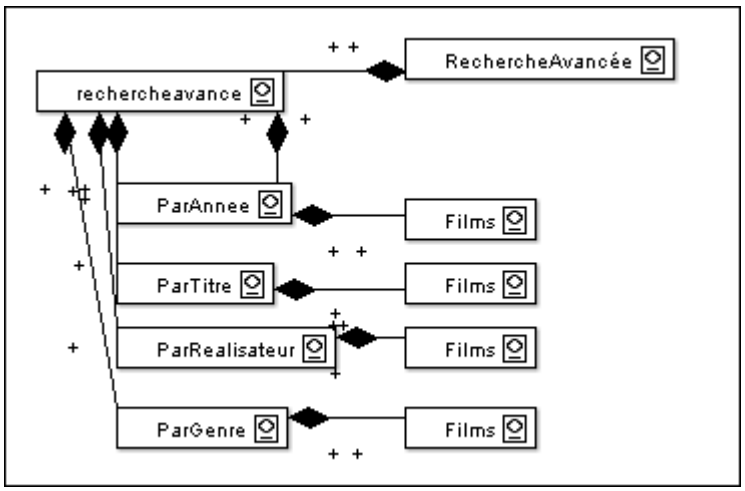


Fig. 11 : Modèle de Présentation – Recherche Avancée

Le quatrième modèle de présentation (*Fig. 12*) est la page film. On y retrouve tout les attributs de film disponible dans le modèle navigationnel et les liens vers la liste des acteurs du film et l'accès aux films du même genre.

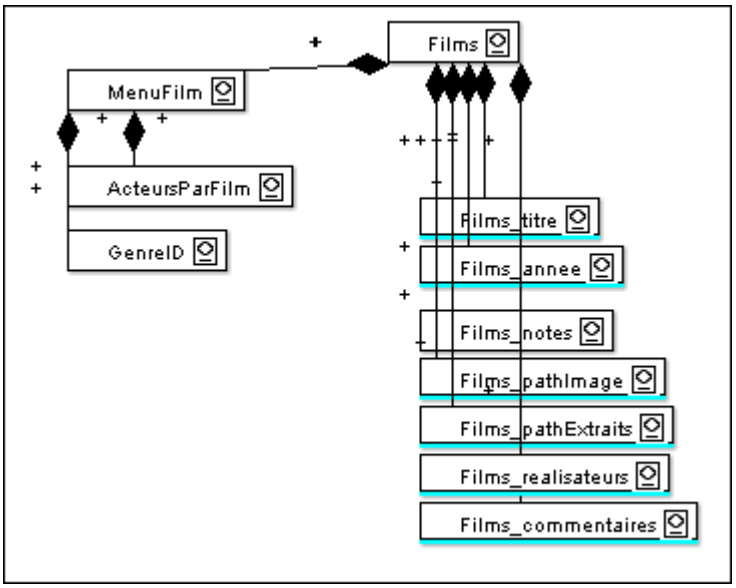


Fig. 12 : Modèle de Présentation – Film

Le quatrième modèle de présentation (*Fig. 13*) est la page Acteurs où l'on peut retrouver les acteurs époux de l'acteur affiché ainsi que la liste de films dans lesquels il a pu jouer. Les attributs de la classe Acteurs sont évidement également disponibles.

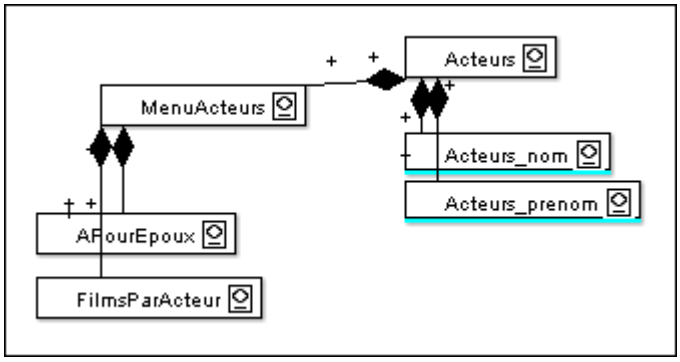


Fig. 13 : Modèle de Présentation - Acteurs

Le cinquième modèle de présentation (*Fig. 14*) est la page genre listant tout les films d'un genre précisé.

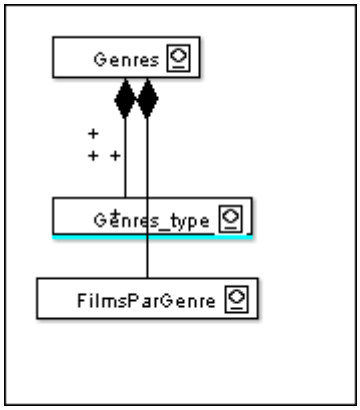
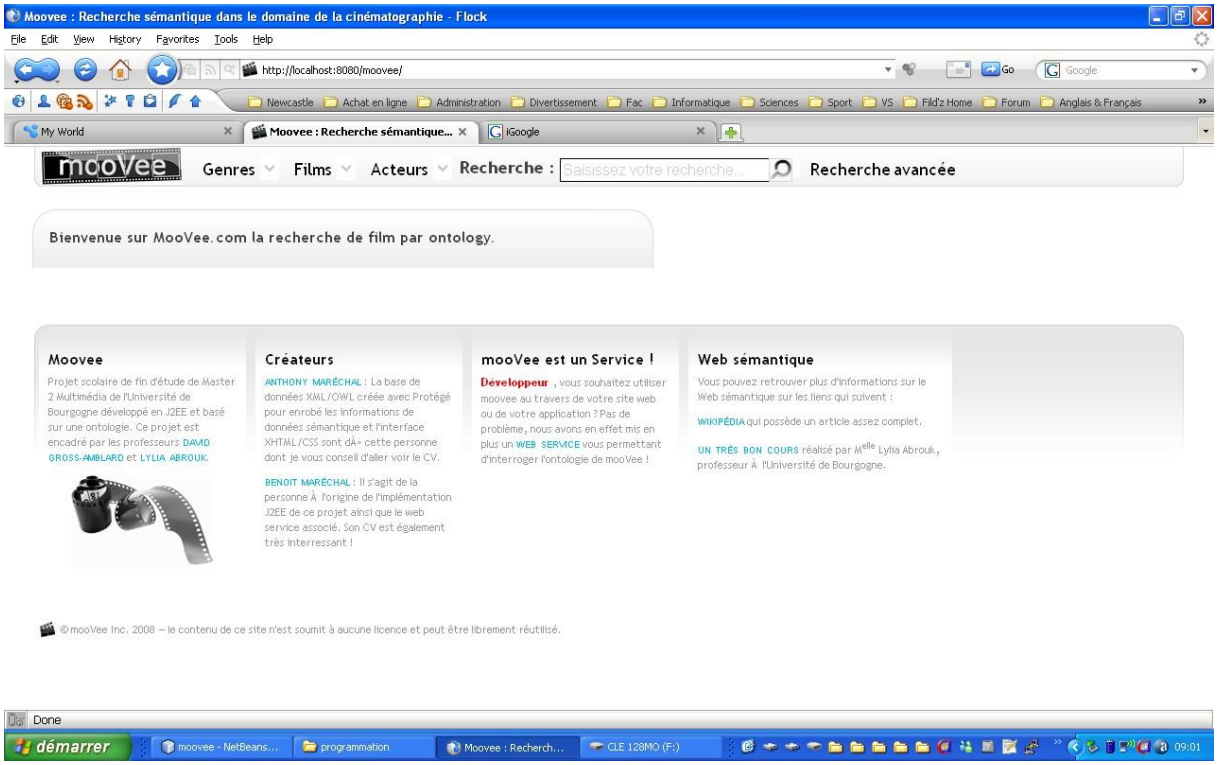


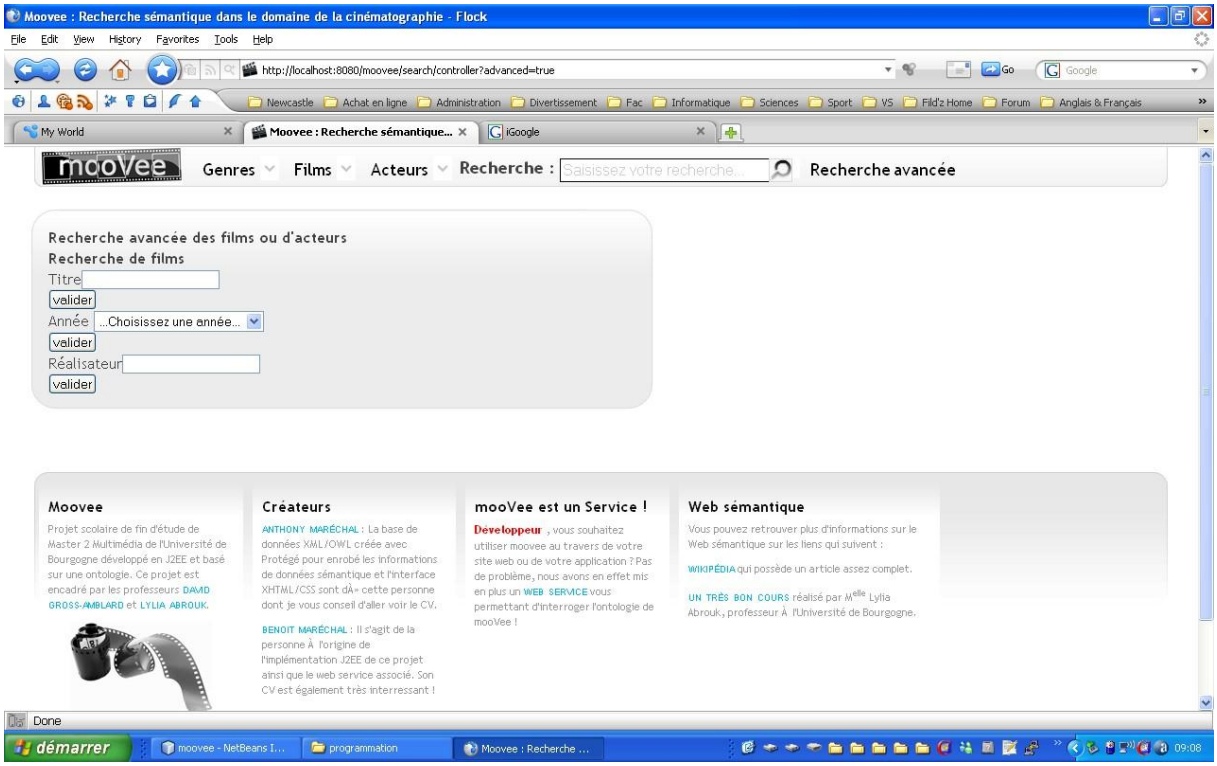
Fig. 14 : Modèle de Présentation - Genres

2. Copie d'écran de l'application

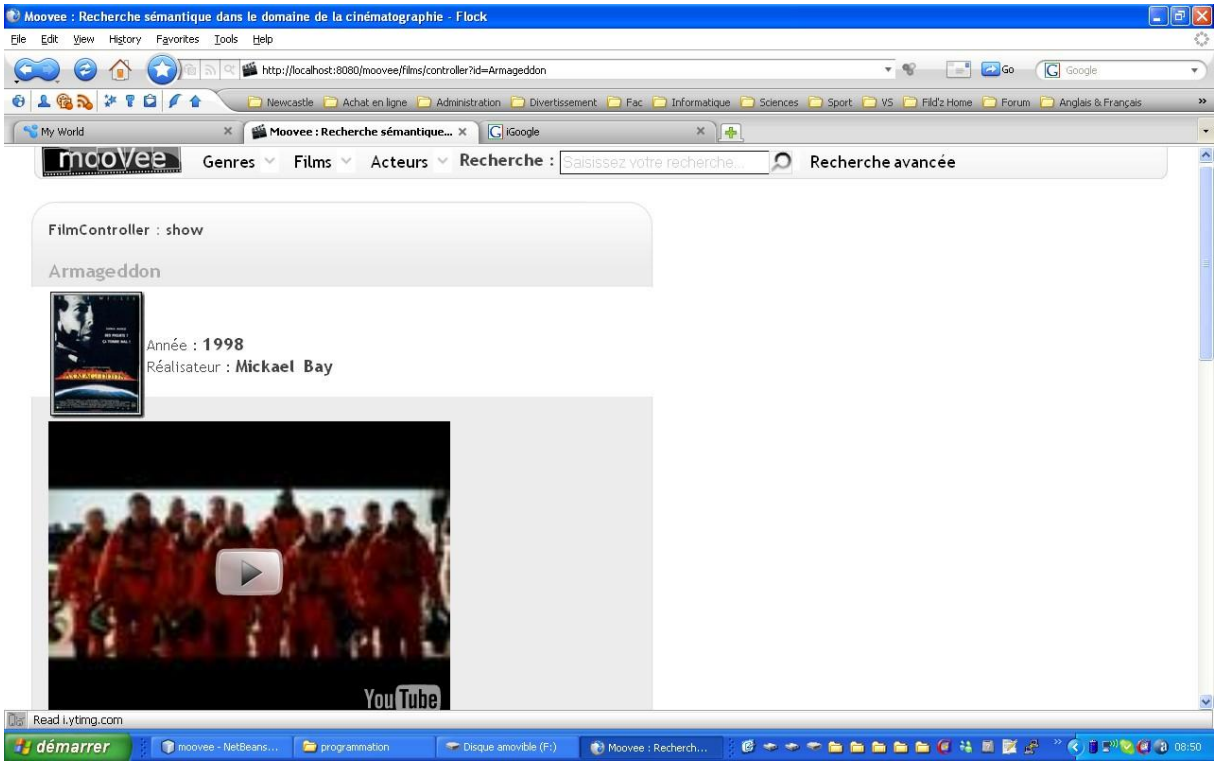
La première copie d'écran représente la page d'accueil :



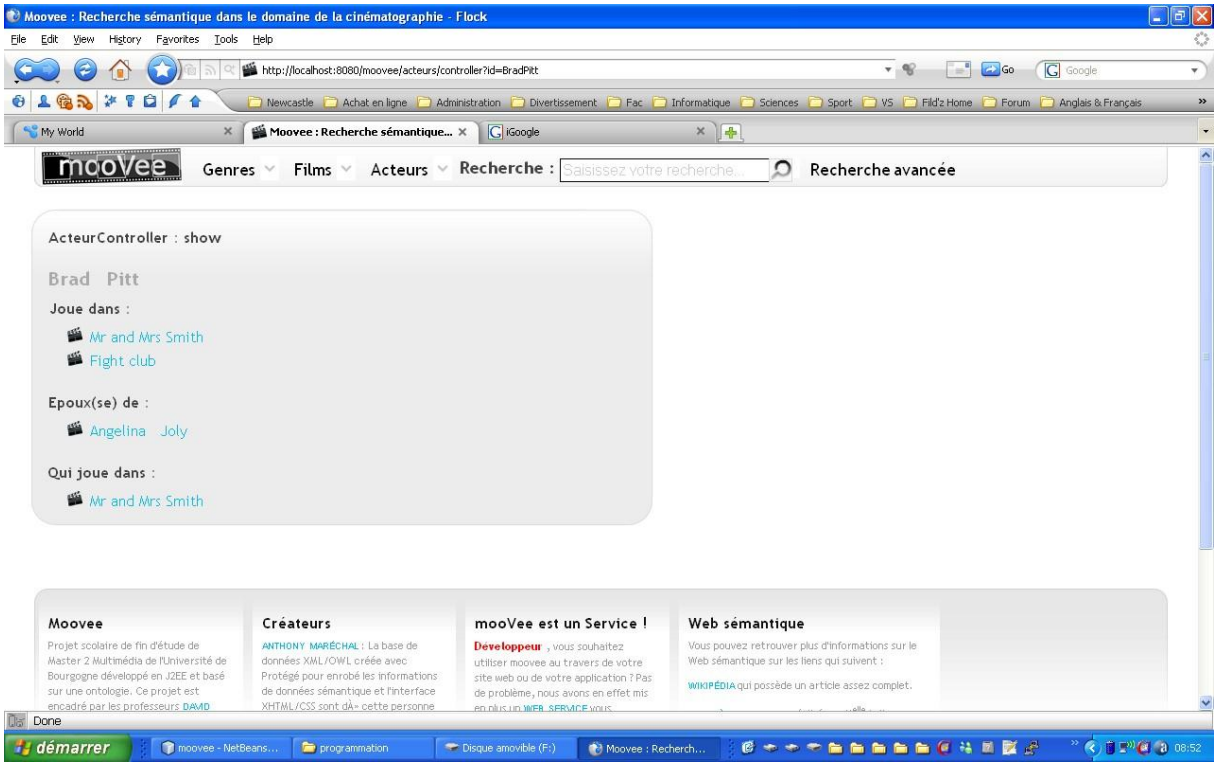
La seconde copie d'écran représente la page Recherche Avancée avec son formulaire de recherche.



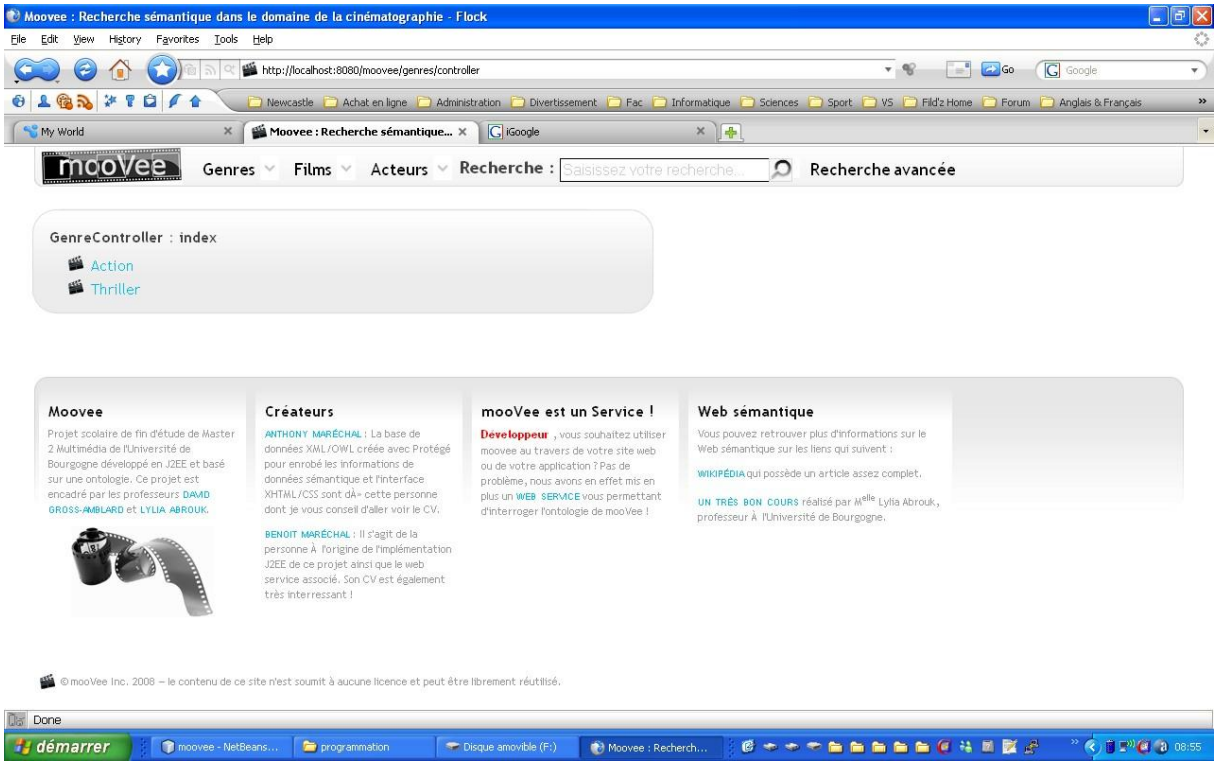
La troisième copie d'écran représente la page Film :



La quatrième copie d'écran représente la page Acteur :



La cinquième copie d'écran représente la page Genre.



Vous pourrez ainsi comparez les informations du modèle de présentation que nous souhaitions avec ces copies d'écran.

PARTIE SYSTEME D'INFORMATION

1. Résumé de notre solution

1.1. Objectif

L'objectif principal du projet est de comprendre, d'apprendre, et de mettre en place les technologies qui sont aujourd'hui à la base de la majorité des systèmes d'informations moderne et notamment du web sémantique. A savoir les langages XML, XSLT, DTD, XPATH mais également RDF, RDFS, OWL, SPARQL et la librairie Jena. Ainsi que la mise en place d'un service web dans un environnement J2EE. Le prétexte utilisé pour développer avec ses technologies a été la réalisation un site web de présentation, de recherche et d'ajout d'informations sur des films et des acteurs.

1.2. Fonctionnalités implémentées

Les fonctionnalités implémentées par notre application sont les suivantes :

- Réalisation d'un site web dans un environnement J2EE avec JSP et Servlet.
- Affichage des films par catégorie.
- Recherche de films dans une ontologie.
- Ajout de notes sur les films.
- Ajout de commentaires sur les films.
- Enrichissements de l'ontologie par les utilisateurs qui peuvent insérer de nouveaux films et de nouveaux acteurs.
- Mise en place d'un service web SOAP.

Notons également les efforts suivants :

- Architecture de programmation respectant le modèle MVC ainsi que le développement RESTFUL. Efforts constant d'encapsulation, de factorisation du code et de respect de conventions pour faciliter les mises à jour et la maintenance de l'application.
- Interface web ergonomique, rapide et simple (cf. : les copies d'écrans) afin de faire une démonstration lors de la soutenance la plus clair possible.

2. Analyse

2.1 Difficultés rencontrées

La première difficulté de ce projet fut le besoin d'identifier quelles technologies et quelle architecture non allions utiliser, car les possibilités étaient nombreuses, et qu'un mauvais choix nous aurait fait perdre beaucoup de temps. La seconde fut de maitriser les concepts du web sémantique ainsi que ses technologies et logiciels (RDF, RDFS, OWL, SPARQL, Protéger et Jena) et de créer une ontologie suffisamment élaborée avant de pouvoir réellement commencer le développement puisque l'ontologie est à la base du projet et doit contenir toutes les données.

Enfin, nous avons été ralentis lors du développement et lors des procédures de tests (*debugging*) par la lourdeur des mécanismes à mettre en place par J2EE : fichiers de configuration à maintenir, grande verbosité du langage notamment comparé aux langages de scripts tel que PHP ou Ruby et lenteur de la compilation et du déploiement pour tester l'application sur une machine modeste. Ainsi que par la nécessité de créer des fichiers XSL pour parser le résultat de chaque type de requêtes SPARQL exécutées pour obtenir une représentation HTML du résultat.

2.2 Architecture du projet

Nous avons retenu une architecture entièrement J2EE pour les raisons suivantes :

J2EE est entièrement basé sur du Java donc Jena (la librairie de manipulation de l'ontologie) est facilement intégrable.

La procédure de création de site web et la mise en place du serveur HTTP est intégré à notre environnement de développement (Netbeans) qui automatise beaucoup de processus de déploiement.

L'envie de se spécialiser en J2EE nous à fait choisir des stages dans ce domaine. Ce projet fut donc l'occasion d'une première approche avec cette technologie.

Vous trouverez ci-dessous (*Fig. 15*) l'architecture générale mise en place pour implémenter les fonctionnalités du sujet, ainsi que les échanges opérés entre les différents éléments impliqués.

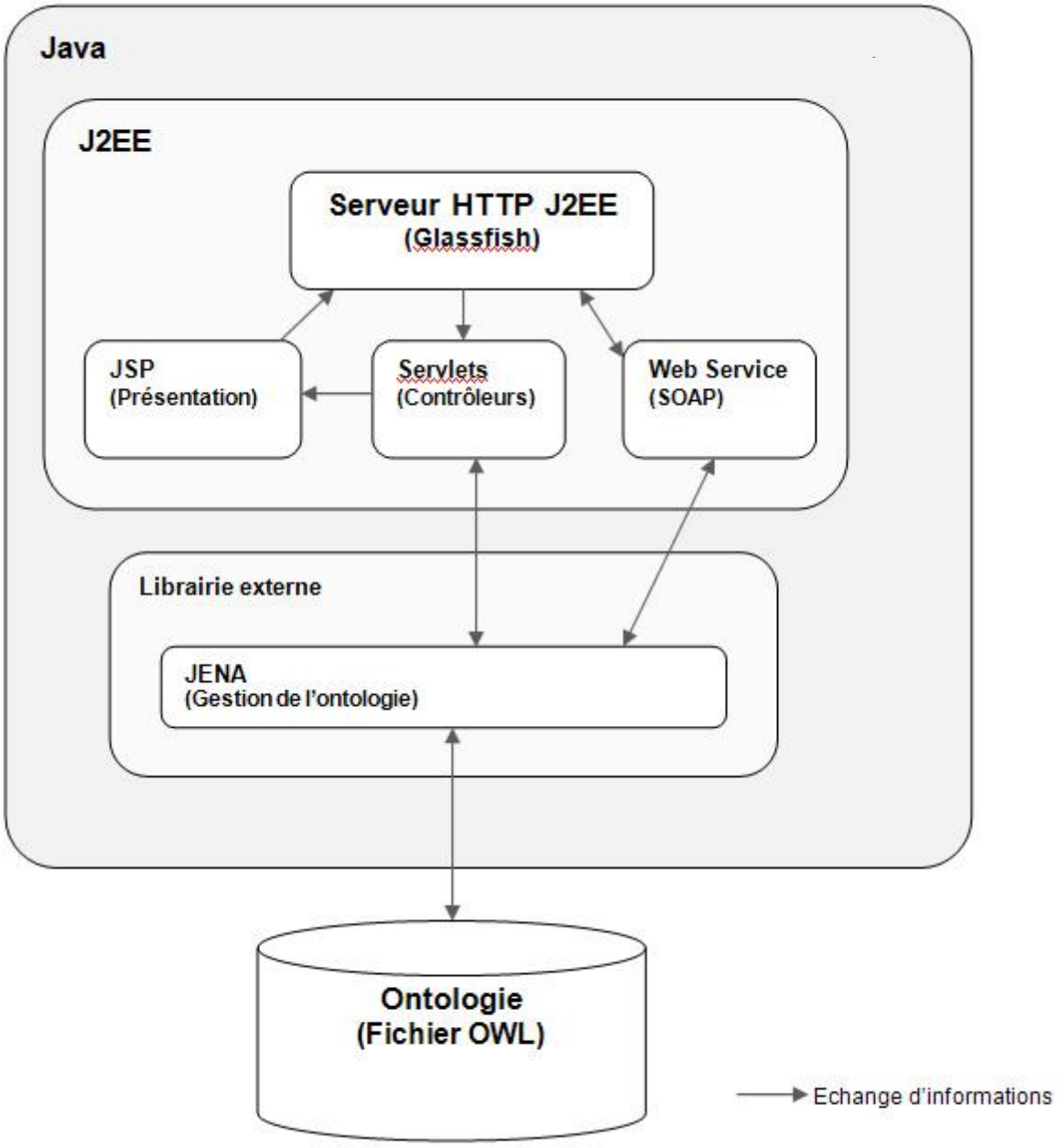


Fig.15 : Architecture générale de l'application

Ce schéma permet de constater que les données sont stockées dans un fichier OWL, que la librairie Jena est utilisée comme intermédiaire entre J2EE et l'ontologie et enfin que le serveur Glassfish est utilisé comme serveur HTTP.

Aussi, nous pouvons effectuer plusieurs remarques sur la figure Fig. 15 :

1. Les pages JSP ont été séparées des Servlets afin de mettre en avant la séparation entre la présentation des données et le traitement des requêtes réalisées par les contrôleurs, mais il ne faut pas oublier que les pages JSP sont en réalité implémentées par des Servlets.

2. La communication entre Jena et l'ontologie est en réalité assez complexe c'est pourquoi et nous la verrons plus en détail dans la suite de ce rapport.

2.3 Architecture de programmation

Le développement en J2EE implique une certaine complexité qu'il est difficile de maitriser sans un minimum d'organisation et d'ordre. C'est pourquoi nous avons choisit dès le départ et pour notre premier projet en J2EE d'appliquer l'architecture Modèle Vue Contrôleur (MVC) offrant une répartition naturel entre les classes d'aiguillage (les Contrôleurs), de communication avec la base de données (les Modèles) et de présentation des données (les Vues). Cependant le modèle MVC reste très permissif en ce qui concerne les choix de nominations des méthodes, des noms des fichiers de présentation et des routes (les urls). Par conséquent, nous avons décidé de respecter pour cela et en parallèle de l'architecture MVC le développement RESTFUL qui impose un ensemble de conventions dont la finalité est de pouvoir offrir un service web sans développer de fichier décrivant les méthodes disponibles tel que le fichier WSDL de SOAP, mais implicitement par l'intermédiaire du protocole HTTP avec une utilisation correcte des verbe http (GET, POST, PUT et DELETE) et de conventions au niveau des URLs[3].

Vous trouverez *Fig. 16* un schéma indiquant l'architecture de programmation notre projet.

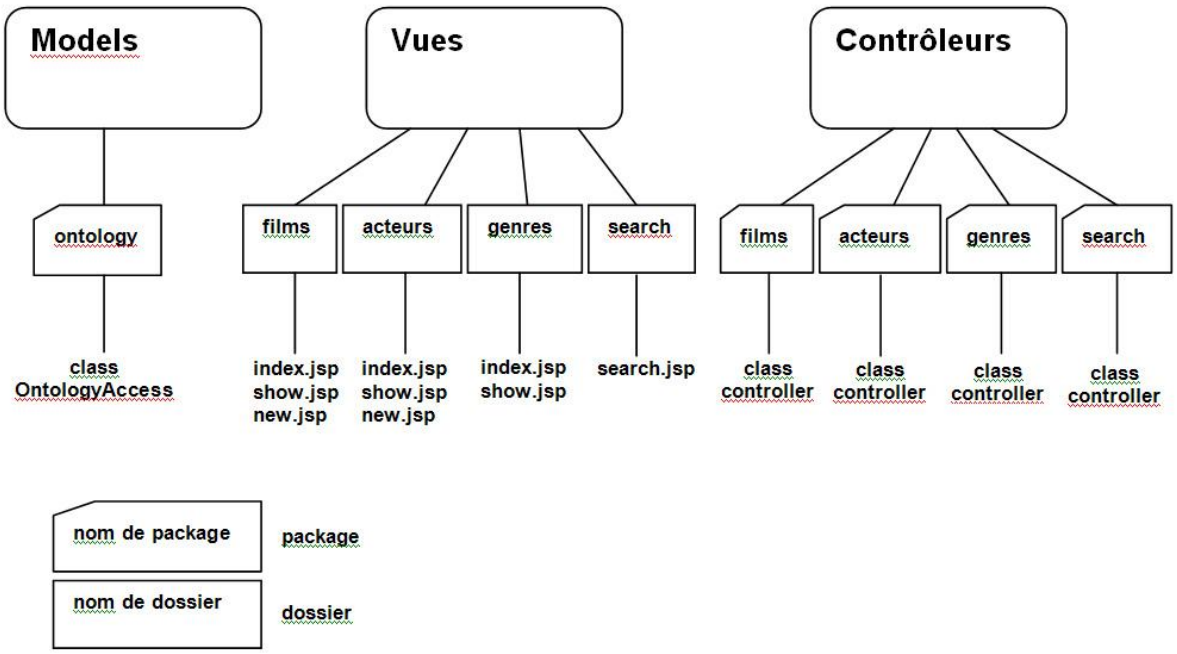


Fig. 16 : Architecture de programmation de l'application

2.3.1 Les Contrôleurs

Le contrôleur est chargé de la synchronisation du modèle et de la vue. Il reçoit tous les évènements de l'utilisateur et enclenche les actions à effectuer. Nous avons mis en place autant de contrôleurs qu'il y a de ressources à manipuler depuis l'interface web à savoir les films, les acteurs et les genres plus un contrôleur dédié aux recherches. Le premier travail de nos contrôleurs est d'aiguiller la requête en fonction des paramètres présents dans l'url (voir la méthode *init()*). Le second est d'appeler la méthode du modèle correspondant à la demande, de mettre le résultat en « attribut » de requête et pour finir d'appeler la page JSP (la vue) adéquate pour l'affichage du résultat.

Toutes les classes contrôleur sont basées sur le même fonctionnement et implémente un certain nombre de méthodes communes.

Variable d'instance commune :

HttpServletRequest request : Contient toutes les informations sur la requête à traiter.

HttpServletResponse response : Contient toutes les informations sur la réponse envoyée au navigateur web.

ontology.OntologyAccess oa : Contient une instance vers le modèle permettant d'effectuer les requêtes sur l'ontologie.

Méthodes communes des contrôleurs gérant une ressource :

void init(HttpServletRequest request, HttpServletResponse response) : Première méthode appelée par le contrôleur, son rôle consiste essentiellement à aiguiller la requête vers la méthode adéquate en fonction d'un paramètre de l'url et selon les conventions d'un développement RESTFUL :

Paramètre « id » présent, méthode show() appelée.

Paramètre « new » présent, méthode new_form() appelée.

Paramètre « create » présent, méthode create() appelée.

Pas de paramètre, méthode index() appelé.

index() : Liste la ressource gérée par le contrôleur. Par exemple liste tous les films pour le contrôleur « Films ».

show() : Affiche les informations sur la ressource. Par exemple affiche tous les détails sur un film pour le contrôleur « Films ».

new_form() : Affiche le formulaire permettant d'ajouter une « instance » (plus ou moins l'équivalent d'un tuple dans le monde des ontologies) de la ressource. Par exemple ajoute une instance de film pour le contrôleur « Films ».

create() : Procède à la création d'une instance dans l'ontologie en fonction des paramètres de la requête HTTP reçus.

Notons que le contrôleur « Films » est pourvu des méthodes add_note() et add_commentaire() qui permettent respectivement d'ajouter une note et d'ajouter un commentaire à un film donné.

Remarque 1 : Développement RESTFUL simplifié, normalement la présence de paramètre dans l'url est liée avec le verbe HTTP utilisé pour déterminer l'action (la méthode) à appeler.

Remarque 2 : Normalement l'action affichant le formulaire d'ajout d'une nouvelle ressource est nommé simplement « new » dans le développement RESTFUL, mais comme il s'agit d'un mot clé réservé en langage Java nous l'avons appelé « new_form() ».

Remarque 3 : Pour implémenter un développement RESTFUL complet pour chaque ressource nous aurions dû également implémenter les actions (méthodes) « edit() » « update() » et « destroy » qui, respectivement, offre un formulaire pré rempli permettant la mise à jour d'une ressource, procède à la mise à jour d'une ressource et détruit une ressource.

2.3.2 Le Modèle : La communication entre Jena et l'ontologie.

Le modèle contient les données manipulées par le programme. Il assure la gestion de ses données et garantit leur intégrité. Notons que pour notre projet nous avons défini un seul modèle gérant toutes les communications avec l'ontologie car

il était plus pratique lors des procédures de tests de n'avoir qu'une seule classe à instancier pour avoir accès à toutes les méthodes concernant l'ontologie. Cependant, maintenant que l'ensemble de ces méthodes sont dépourvus de problèmes, il serait plus propre de créer des classes filles de ce modèle qui encapsulerait les méthodes concernant une même ressource.

La classe implémentant notre modèle s'appelle *OntologyAccess* et voici ces variables d'instance et méthodes les plus pertinentes :

Variable d'instance :

String base_prefix : Contient le préfix de base des ressources de l'ontologie que nous avons créé (fixé à : http://www.moovee.com/ontology#).

OntModel model : Contient la représentation en mémoire de l'ontologie.

public OntologyAccess() : Constructeur c'est lui qui met en mémoire l'ontologie lors de l'instanciation du modèle.

public void creerHTML(String xml, String xsl, String html): Méthode récupérée sur developpez.com permettant de générer un fichier HTML à partir d'un fichier XML et d'un fichier XSL.

public String processSPARQLandXSLT(String queryString,String f_xslt) : Exécute la requête SPARQL, parse le résultat XML avec le fichier XSL donnée en paramètre et retourne sous forme de chaîne de caractères (String) le fragment d'XHTML obtenu. Nous reviendrons sur cette méthode dans la section « Exécution de requête SPARQL sur l'ontologie ».

public void save() : Sauvegarde l'ontologie dans le fichier OWL sur le disque.

public String films_index() : Liste les films grâce à l'appel de la méthode processSPARRQLandXSLT avec la requête et le fichier XSL adéquate.

public String film_show(String id) : Affiche les informations sur le film identifié par « id ».

public String film_create(String titre,String annee,String realisateur_id, String genre,List<String> acteurs) : Ajoute un film à l'ontologie en utilisant les méthodes offertes par la classe OntModel de la librairie Jena.

Remarque : Des méthodes similaires ont été écrites pour les ressources « Acteur » et « Genre ».

public String film_add_note(String id,String note) : Ajoute une note à un film identifié par son identifiant « id ».

public String film_add_commentaire(String id,String login,String contenu) : Ajoute un commentaire à un film.

public String search_acteurs(String q) : Effectue une recherche sur le nom et le prénom des acteurs et renvoie le résultat.

public String search_films(String q) : Effectue une recherche sur le titre d'un film et renvoie le résultat.

2.3.3 Exécution de requêtes SPARQL

L'exécution de requête SPARQL sur l'ontologie est réalisée par la méthode *processSPARQLandXSLT(String queryString, String xslt)* dont l'algorithme est expliqué ci-dessous :

1 – Ajout des préfixes (du préfixe de base ainsi que des préfix rdf et rdfs afin de délimiter les espaces de nom) à la requête SPARQL.

2 – Exécution de la requête SPARQL sur l'ontologie en mémoire et écriture du résultat XML dans un fichier sur le disque.

3 – Transformation du fichier XML résultat en un fichier HTML grâce à la feuille de style XSL par la méthode *creerHTML*.

4 – Lecture du fichier HTML dans une chaîne de caractères qui est donc le résultat produit de cette méthode.

Afin de mieux comprendre les acteurs de cette méthode et leur enchainement nous avons réalisé le schéma ci-dessous :

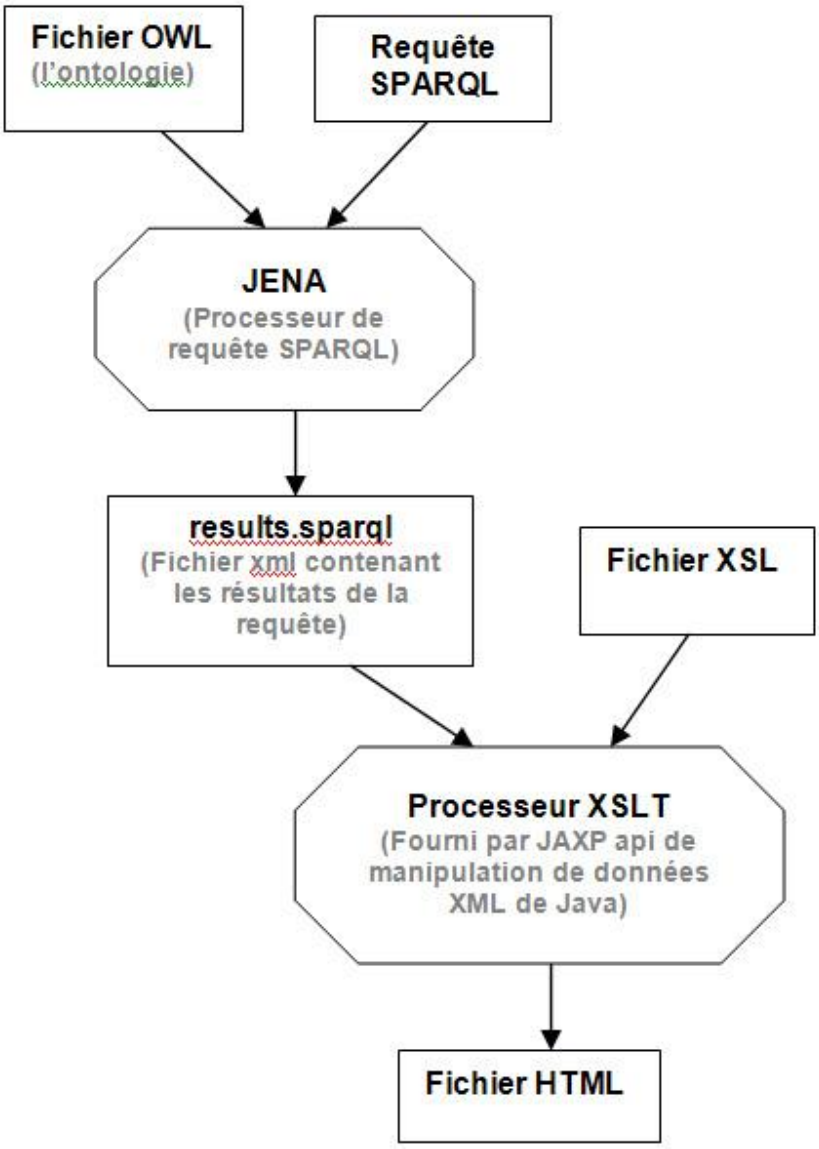


Fig. 17 : Schéma d'exécution de la méthode processSPARQLandXSLT

2.3.4. Insertion de données dans l'ontologie

Conformément au sujet, les utilisateurs peuvent enrichir l'ontologie par l'ajout d'acteurs, de films, de notes et de commentaires. Pour cela nous avons utilisé les méthodes fournis par Jena et notamment la classe OntModel. Retrouvez ci-dessous les étapes typiques d'une insertion.

Remarque : Le vocabulaire utilisé est celui des ontologies.

Récupération de la « Classe » à ajouter. Par exemple « Acteur ».

Création d'une URI unique pour la ressource à ajouter.

Création de la ressource en précisant l'URI et le type de ressource récupéré précédemment.

Ajout des propriétés à cette ressource en récupérant le type de propriété et la valeur qui lui est associé.

Sauvegarde sur le disque dur de l'ontologie.

2.3.5 Les Vues

Des vues sont utilisées pour la présentation des informations récupérées par le modèle. Chaque ressource dispose des pages JSP *index.jsp* et *show.jsp* qui respectivement liste les instances de la ressource et affiche les détails sur une instance donnée. Les ressources « Film » et « Acteur » dispose toutes deux d'une page new.jsp fournissant un formulaire d'ajout d'une nouvel instance. Enfin le formulaire de recherche est contenu par la page search.jsp.

Instructions présente dans une page *JSP* typique de l'application :

- Inclusion de l'entête de page (fichier *header.jsp*)

- Titre

- Insertion du fragment d'XHTML résultat de la requête sur l'ontologie.

- Inclusion du pied de page (fichier *footer.jsp*)

3. Ontologies

L'ensemble des données sont contenu par une ontologie implémentée dans le fichier OWL « *moovee.owl* » situé à la racine de l'application et généré à l'aide du logiciel libre Protégé. Pour définir la sémantique autour d'un film nous avons défini un ensemble de classes aux noms explicites reliées entre elle par des propriétés objet (O*bject properties*) selon le diagramme de la Figure 18.

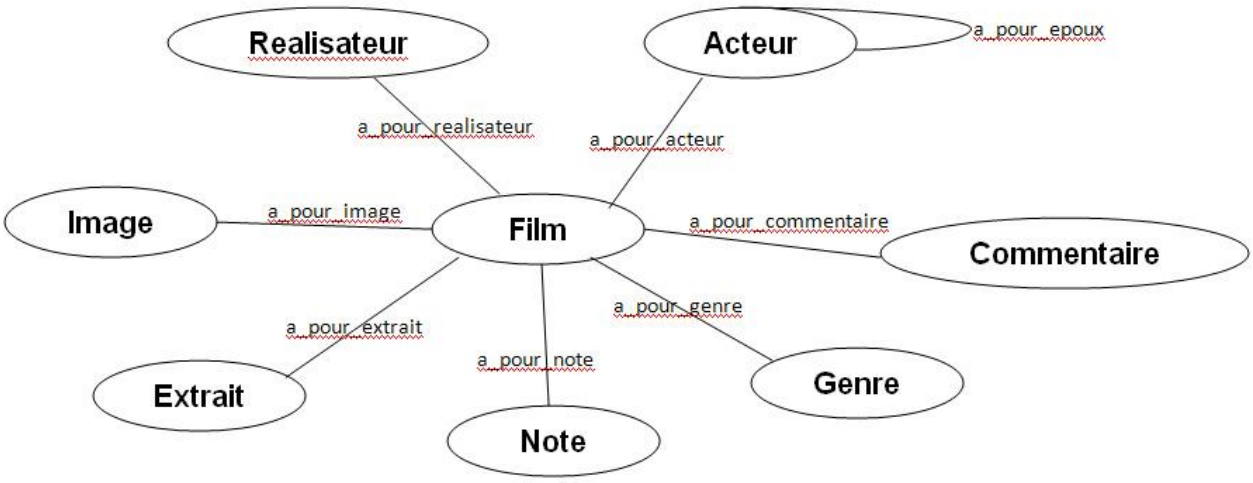


Fig. 18 : Ontologies misent en place autour de la notion de film

Notons que pour chaque propriété nous avons définie sa propriété inverse. Par exemple pour la propriété « a_pour_acteur » nous avons défini « joue_dans » pour propriété inverse permettant une plus grande simplicité d'interrogation de l'ontologie lors de l'écriture des requêtes SPARQL.

Notre ontologie définie également des propriétés de type de données (Datatype properties) pour chaque classe qui figure sous forme d'attribut (voir le diagramme de conception Partie Application).

Le nom des attributs et propriété étant très explicite nous ne reviendrons pas dessus.

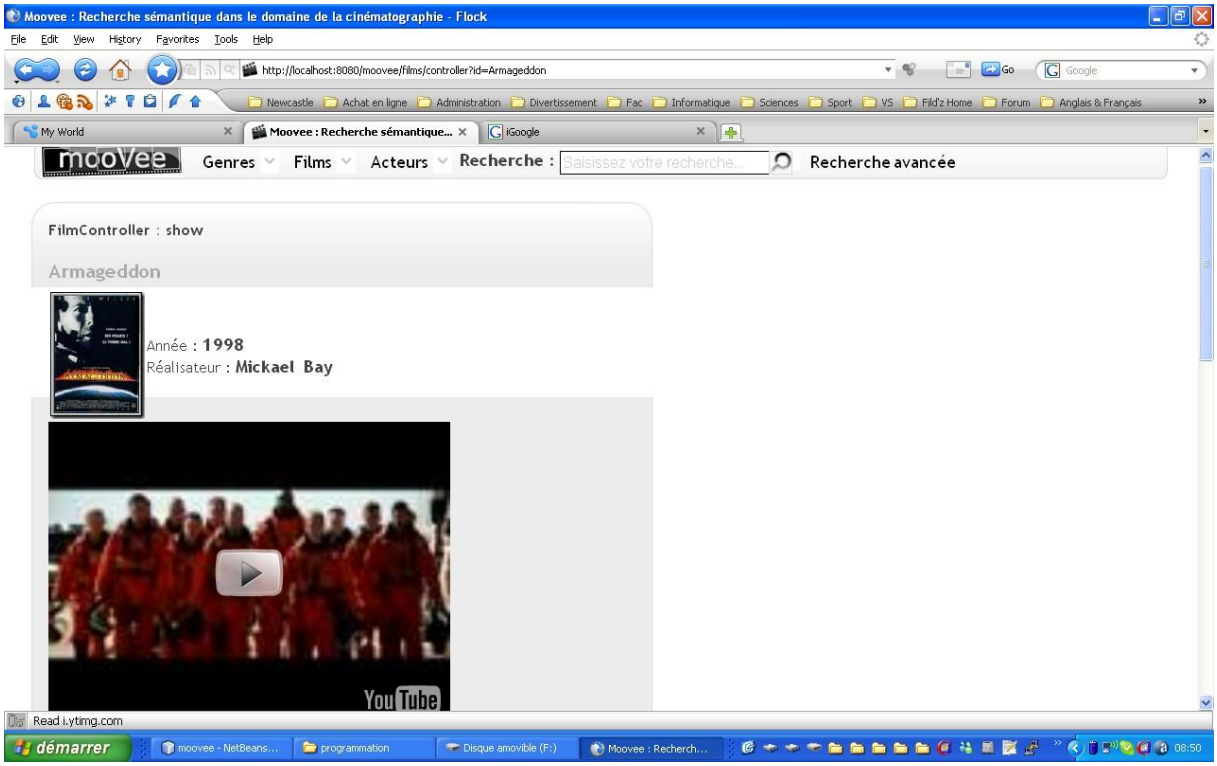
Afin de faire la démonstration de l'application nous avons également ajouté des individus (*Individual*) à notre ontologie afin d'initialiser un certain nombre de films, d'acteurs, de genres et de réalisateurs.

4. Web service

La mise en place du web service à été très rapide puisque nous avons juste eu besoin de créer une classe type Web Service grâce à Netbeans, d'ajouter en variable d'instance notre classe Modèle (*OntologyAccess*) et pour chaque méthode offerte par le modèle écrire la méthode type Web Service correspondante (avec les instructions *@WebMethod* et *@WebParam*). Le serveur et Netbeans s'occupant pour nous de générer une interface de test du Web service, ainsi que de générer le code SOAP correspondant aux requêtes et aux réponses demandé.

3. Interface graphique

Voici une copie d'écran de l'interface graphique qui se voulait la plus ergonomique possible. On rappel par ailleurs que l'interface est réalisée en XHTML transitional et CSS.



4. Jeux de tests

1-Ouvrir et lancer le projet sous Netbeans 6

2-Ouvrir un navigateur web et aller à l'adresse : http://localhost:8080/moovee

3-Afficher la liste des films en cliquant sur l'onglet Films

4-Afficher le détail d'un film en cliquant sur son titre

5-Ajouter une note au film en choisissant une note et en cliquant sur « Valider ».

6-Ajouter un commentaire au film en remplissant le formulaire et en cliquant sur « Valider ».

7-Afficher un acteur en cliquant sur son nom.

8-Afficher l'époux d'un acteur.

9-Afficher la liste des acteurs.

10-Effectuer une recherche en remplissant le formulaire de recherche situé dans le menu principal et appuyer sur la touche « entrer ».

11-Afficher les genres en cliquant sur l'onglet « Genres »

12-Afficher un genre en particulier en cliquant sur l'onglet « Genres » et en sélectionnant un genre.

13-Enrichir l'ontologie en créant un film. Pour cela cliquer sur l'onglet « Films » puis sur « Ajouter »

14-Créer un nouvel acteur en cliquant sur l'onglet « Acteurs » puis sur « Ajouter ».

15-Tester le service web en cliquant sur le lien « WEB SERVICE » dans la partie pied de page du site.

16-Tester les méthodes du service web en remplissant les formulaires associés et en cliquant sur leur bouton de validation respectif.

Bilan

Afin d'utiliser dans notre projet toute les connaissances vues dans le module Système d'information nous avons écrit la *DTD* que doit respect le fichier *XML* d'une requête *SPARQL*. Vous la trouverez sous le nom « *sparql-results.dtd* » à la racine des sources du projet.

Bibliographie

Au-delà de ce que nous avons pu apprendre et réaliser en CM, TD et TP, plusieurs sources nous ont aidées durant la création de ce rapport, notamment :

[1] Site internet de référence pour notre rapide étude des fonctionnalités à implémenter dans un projet de gestion de film
http://allocine.com

[2] Institut d'informatique Allemand d'où la méthode UWE est issue.
http://www.pst.informatik.uni-muenchen.de/~kochn/IWWOST02-koch-kraus.pdf

[3] Développement RESTFUL
http://fr.wikipedia.org/wiki/Representational_state_transfer

SPARQL W3C
http://www.w3.org/TR/rdf-sparql-query/

Doc JENA
http://jena.sourceforge.net/documentation.html

FAQs J2EE developpez.com

http://java.developpez.com/faq/javaee/

UML 2 : Modéliser une application web.
Pascal Roques. ISBN : 978-2841773374

Article UWE de Wikipedia.
http://en.wikipedia.org/wiki/UML-based_Web_Engineering_(UWE)

Annexe

1 - DTD du résultat d'une requête SPARQL

```
<!ELEMENT sparql (head, results)>

<!ELEMENT head (variable*) >

<!ATTLIST head name CDATA #REQUIRED >

<!ELEMENT result (binding+) >

<!ELEMENT binding (uri | literal) >

<!ATTLIST binding name CDATA #REQUIRED >

<!ELEMENT uri (#PCDATA) >

<!ELEMENT literal (#PCDATA) >

<!ATTLIST literal xml:lang CDATA #IMPLIED >

<!ATTLIST literal datatype CDATA #IMPLIED >
```

2 – Exemple de fichier XSL de la transformation du fichier XML résultat en frangment HTML : Le fichier films_show.xsl d'affichage d'un film donné.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
                omit-xml-declaration="yes"
                method="xml"
                encoding="ISO-8859-1"
                indent="yes" />
        <xsl:template match="/">

        <h3> <xsl:value-of select="//binding[@name='titre']/literal" /></h3>
        <table>
        <tr>
        <td>
<img src="/moovee/film_images/{//binding[@name='path']/literal}"
width="100" ></img>
        </td>
        <td>
        Année:
<strong>
<xsl:value-of select="//binding[@name='annee']/literal" />
</strong>
        <br/>Réalisateur:
```

```
        <strong>
        <xsl:value-of select="//binding[@name='realisateurprenom']/literal" />
        </strong>
         <strong>
        <xsl:value-of select="//binding[@name='realisateurnom']/literal" />
        </strong>

                </td>
                </tr>
                </table>
        <object width="425" height="355">
        <param name="movie"
value="http://www.youtube.com/v/{//binding[@name='pathex']/literal}">
        </param>
        <param name="wmode" value="transparent">
        </param>
        <embed
src="http://www.youtube.com/v/{//binding[@name='pathex']/literal}"
type="application/x-shockwave-flash" wmode="transparent" width="425"
height="355">
                </embed>
                </object>
                </xsl:template>
        </xsl:stylesheet>
```