MOVING Modification BackOffice

*Spécifications*

Ce document synthétise les spécifications logicielles relatives aux changements à mettre en œuvre dans le BackOffice.

Benjamin Verney – Thomas Badin

06/05/2014

MOVING Modification BackOffice

Spécifications

Les besoins exprimés pour le (les) fronts offices demandent certaines modifications dans le BackOffice. Les spécifications de ces changements sont consignées dans ce document. Les changements dans le back offices se feront directement en modifiant le projet [Zend Framework 2](http://framework.zend.com) (projet hébergé au 6 mai 2014 sur [Github](https://github.com/thbadin/cervin.git)).

Table des matières

Authentification via les API 2

Gestions des droits User de l’API 3

Historique des requêtes Consultation 4

Historique des requêtes Contribution 4

Recommandations 4

Correspondance Tag 🡺 Elément 4

Correspondance Tag (non GPS) 🡺 Eléments 6

Correspondance GPS 🡺 Eléments 7

Extension de la gestion des media 8

INTEGRATION DU MULTILANGAGE 9

# Authentification via les API

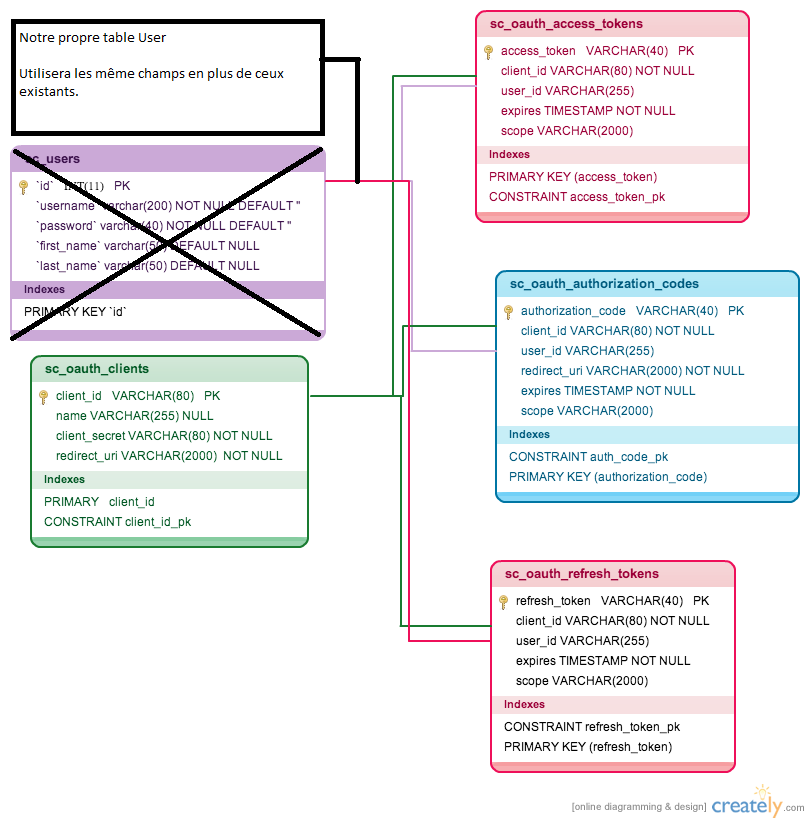
Priorité : 1 Très haute

Le BackOffice doit être modifié afin de stocker des informations d’authentification pour les API.

Nous utiliserons la base de données utilisateurs existante pour définir les utilisateurs autorisés à contribuer par le moyen des webservices POST.

Aussi, nous stockerons les tokens d’accès et les tokens de refresh associés aux users.

Le module d’authentification OAuth2 que nous implémenterons (voir WebServicesReadOnly.docx) met à disposition tout cela mais pose un problème concernant la table utilisateurs, le module est fait de façon à utiliser sa propre table d’users,  nous allons donc le modifier pour utiliser notre propre table.



# Gestions des droits User de l’API

Un utilisateur pourra se voir attribuer le rôle de Contributeur, un contributeur peut créer des éléments sur l’IHM du BackOffice ou en utilisant les webservices POST.

Cependant chacune de ses contributions restera sous la forme d’un brouillon tant qu’un admin ne l’aura pas validé.

De plus, les webservices de consultation ne permettrons pas d’exporter les éléments qui ne sont pas encore publiés, le contributeur ne pourra alors pas consulter un élément qu’il vient de créer (contribution) en utilisant les webservices GET.

# Historique des requêtes Consultation

On veut pouvoir garder une trace chaque fois qu’une requête est effectuée contre les webservices.

Pour les services GET, seules les scènes consultées nous intéressent pour pouvoir recommander des scènes en relation avec les scènes déjà visitée.

On stockera alors, pour chaque requête, dans une nouvelle table d’historique **:**

**Id User, id Scene, timestamp**

**Shéma :**

|  |
| --- |
| **HistoriqueConsultationScene** |
| **(PK)(int)-Id**  **(FK)(int)-IdUser**  **(FK)(int)-IdScene**  **(timestamp)-DateTime** |

# Historique des requêtes Contribution

Pour les services POST, on veut connaître les médias créés par le contributeur.

Le modèle actuel de la base de données nous permet de faire ceci sans ajouter de table, la table média possède un attribut IdUser c’est l’Id du créateur.

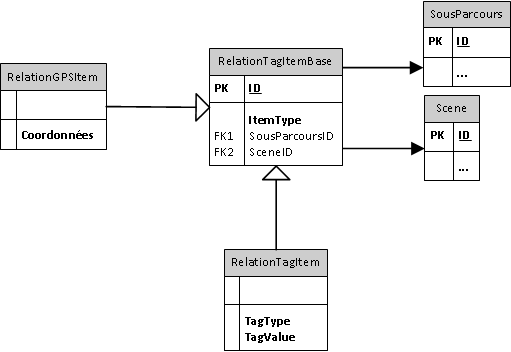
Nous pouvons donc utiliser ce modèle pour avoir l’historique des contributions. Il suffira de créer une fonction getMediaByUserId qui retournera les médias créés par un utilisateur en spécifié.

# Recommandations

On veut pouvoir recommander des scènes à un user en fonction de son historique de consultation.  
La méthode (algo) reste à définir.

# Correspondance Tag 🡺 Elément

Pour les besoins de l’application mobile, il sera nécessaire de rajouter un mécanisme de correspondance entre l’ID d’un Tag (NFC, QRCode, iBeacon) ou d’une coordonnée GPS et d’un élément. Pour cela nous avons besoin d’étendre la base de données de la manière suivante :



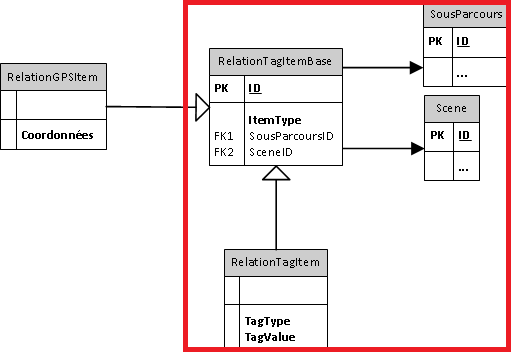
Le fonctionnement sera simple : par exemple l’application demandera au Back Office via un Web Services (cf document de conception des WebServices) a quoi correspond cet Identifiant NFC. Le WebService lui répondra qu’il correspond à la Scène (ou au Sous parcours) avec tel Identifiant de base de donnée.

La complexité entre la relation Tag 🡺 Eléments et position GPS 🡺 Eléments étant vraiment différentes nous avons choisi de séparer les deux développements.

## Correspondance Tag (non GPS) 🡺 Eléments

Priorité : 5 Moyenne

Le back office doit être modifié afin de stocker la correspondance ID tag (NFC, QRCode, iBeacon) et un point d’intérêt. Un point d’intérêt est soit une scène, soit un sous parcours. Cette partie représentante la portion suivante du diagramme :



Descriptions des attributs :

|  |  |  |  |
| --- | --- | --- | --- |
| Table | Attributs | Type | Description |
| RelationTagItemBase | ID | Automatique | Entier auto incrémenté |
| RelationTagItemBase | ItemType | Enumération | 2 valeurs possible : SousParcours / Scene |
| RelationTagItemBase | SousParcoursID | Entier (FK) | Si ItemType == SousParcours : Id du sous parcours ; Sinon : null |
| RelationTagItemBase | SceneID | Entier (FK) | Si ItemType == Scene : Id de la scène; Sinon : null |
| RelationTagItem | TagType | Enumération | 3 valeurs possibles : NFC/QRCode/iBeacon |
| RelationTagItem | TagValue | String | Valeur du Tag (Id de la puce NFC par exemple) |

La base de donnée et Doctrine (l’ORM utilisé dans ZF2) devra être étendu, en respectant la logique du schéma ci-dessus.

Une interface d’administration devra être rajoutée à Moving BO, permettant d’ajouter / supprimer / modifier une correspondance.

Pour l’ajout et la modification, l’interface devra comporter les éléments suivants :

* Le choix du type de Tag (via une liste déroulante par exemple). Dans cette première version la liste proposera NFC / QRCode / iBeacon
* La valeur du Tag (TextBox classique, valeur non nulle)
* Le choix entre Scène et SousParcours. Une fois le choix Scène / SousParcours fait, l’interface proposera une interface de selection de Scène / de SousParcours. (Avec un champs de recherche)

Les validations suivantes devront être effectuées :

* La valeur du tag doit être alphanumérique (sans espace), et non null
* Le couple Tag Type / Tag Valeur doit être unique
* Si ItemType correspond à Scene, SceneID doit être non nul
* Si ItemType correspond à SousParcours, SousParcoursId doit être non nul
* L’intégrité référentielle doit être respectée

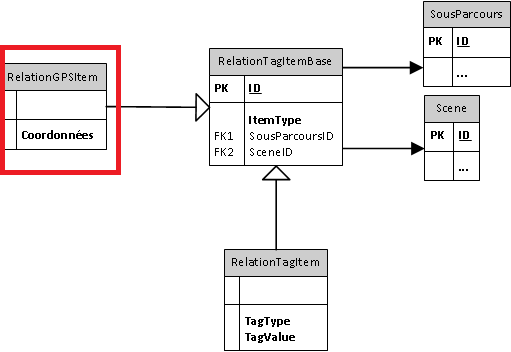
## Correspondance GPS 🡺 Eléments

Priorité : 7 Basse

Cette partie est assez complexe et utilise des technologies peu courantes. N’étant pas un expert en Système d’Information Géographique, et cette fonctionnalité n’étant pas primordiale pour la première version, les spécifications ne seront pas très poussé, et la conception assez libre. (Il faudra a priori tester différentes technologies).

Pour stocker les coordonnées GPS, il faudra se baser sur l’extension GIS de MySQL (<http://dev.mysql.com/doc/refman/5.0/fr/gis-introduction.html>). Pour doctrine, une extension existe (<https://github.com/tvogt/doctrine2-spatial/>), à tester !

La base de données devra être étendue afin de stocker les données géographiques :



La table RelationGPSItem (héritant de RelationTagItemBase) stockera une coordonnée au format [Polygon](http://dev.mysql.com/doc/refman/5.0/fr/gis-class-polygon.html). Le Polygon correspondra à une partie de carte liée à un élément. Exemple : Une scène décrit l’histoire de l’Inovallée. On stockera dans la table de correspondance un polygon correspondant au contour de l’Inovallée dans la base de données, et on le liera à la scène décrivant l’histoire de l’Inovallée. Lorsque l’utilisateur entrera dans le polygon représentant l’Inovallée, l’application lui proposera de consulter la scène associée.

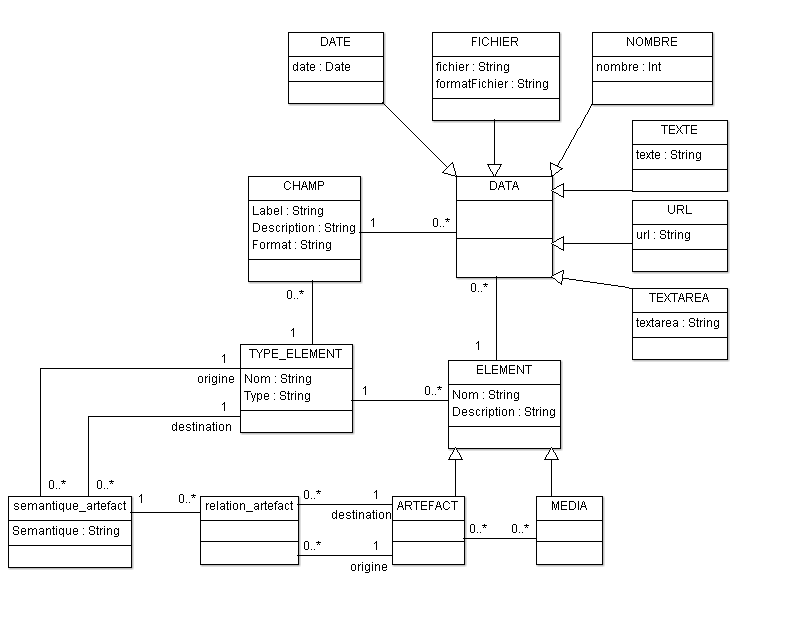
Note : On pourra remplacer l’utilisation des [Polygon](http://dev.mysql.com/doc/refman/5.0/fr/gis-class-polygon.html) par des [MultiPolygon](http://dev.mysql.com/doc/refman/5.0/fr/gis-class-multipolygon.html) afin de permettre d’associer directement plusieurs régions à un élément. (Plus complexe à gérer).

La principale difficulté se trouve au niveau de l’interface d’administration. Après avoir choisi un Tag de type GPS (a rajouter dans les choix), l’utilisateur devra définir le polygon. Il faudra donc développer une interface de sélection géographique (regarder du coté d’OpenStreetMap).

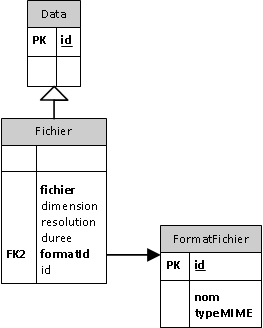
# Extension de la gestion des media

Priorité : A définir

Actuellement les médias sont gérés de la manière suivante (cf. [2013/Documentation Technique [MOVING-BO].doc](../../2013/Documentation%20Technique%20%5bMOVING-BO%5d.doc) :



Pour le type de fichier, nous aurons besoin de stocker plus d’information, en fonction du type du fichier. Ces informations permettront d’afficher correctement le fichier au client. Nous rappelons que le front office devra permettre d’afficher les médias de type suivants :

* Images (type a définir, mais au moins jpeg et png)
* Vidéo (type à définir)
* Audio (type à définir mais au moins mp3)
* PDF

Nous pensons avoir besoin des éléments suivants pour l’affichage (liste non exhaustive) :

* La durée (pour de l’audio, de la vidéo)
* Les dimensions (pour de la vidéo, des images)
* Une résolution (pour de la vidéo ? vraiment utile ?)

Pour le format de fichier, il sera préférable de créer une table séparée avec les différents types de fichier recensé. Au final le fichier devra être représenté dans la base de données de manière ci-contre.

Au niveau des changements à réaliser dans l’interface d’administration, il faudra modifier la fonction d’upload afin de d’extraire les métadonnées (durée, dimensions, résolutions…) des fichiers. Actuellement, seule le type MIME est détecté en fonction de l’extension du fichier.

# Internationalisation

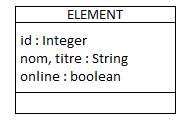
Priorité  A definir

## Changements dans la collection et les parcours

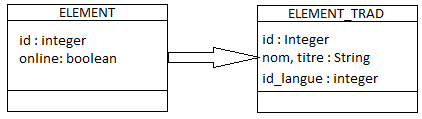
Pour pouvoir intégrer le changement de langue à l’application, il est nécessaire de modifier les classes pour qu’elles prennent en paramètre la langue sélectionnée.

La solution choisie est, pour chaque classe dont les attributs changent en fonction de la langue, nous créons un clone de cette classe contenant les attributs en question plus un attibut id\_langue.

Exemple : La classe Element

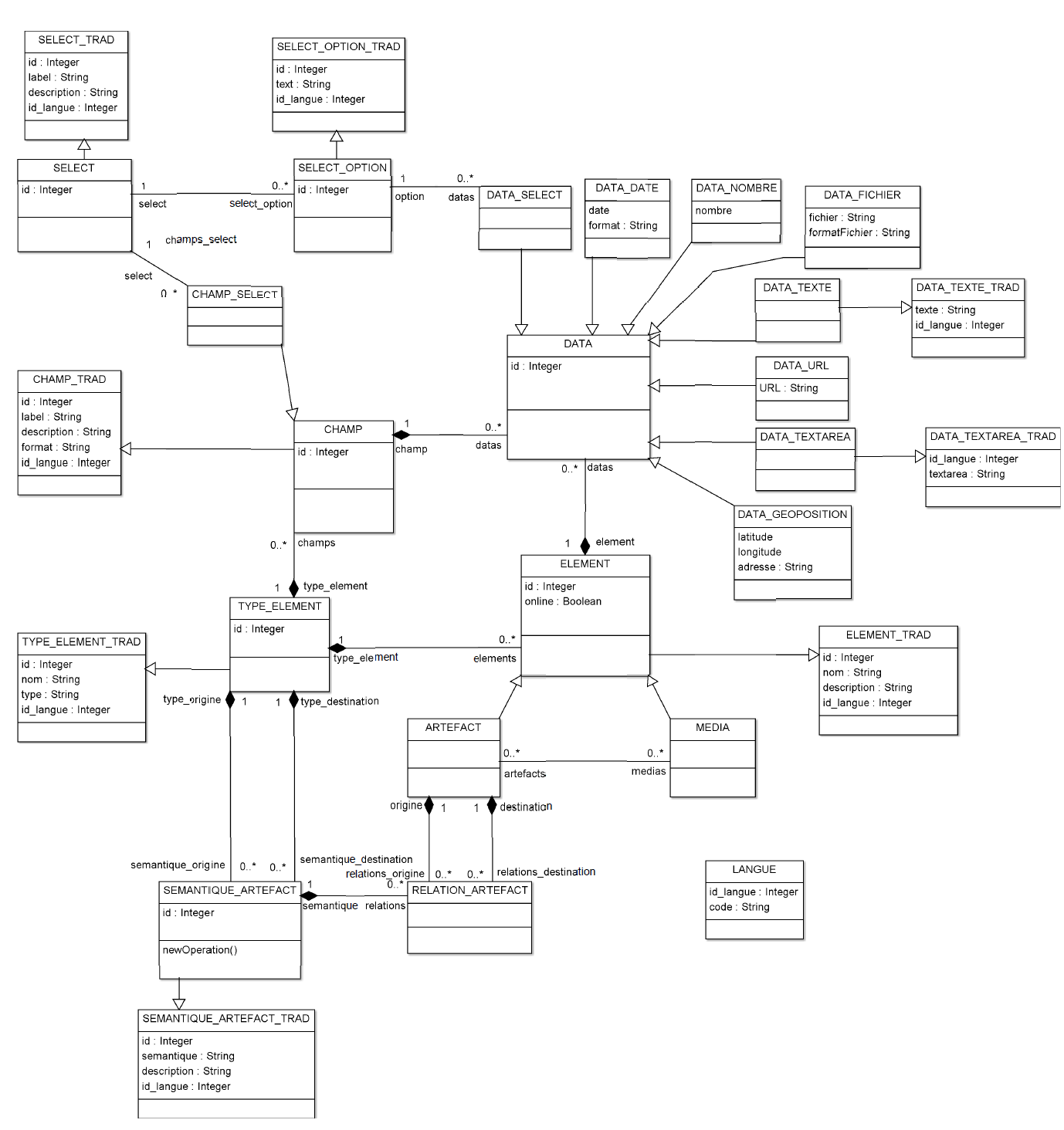


Devient :



Cela permet de conserver un seul id par media, plutôt que d’avoir des id différents pour un même media existant en plusieurs langues.

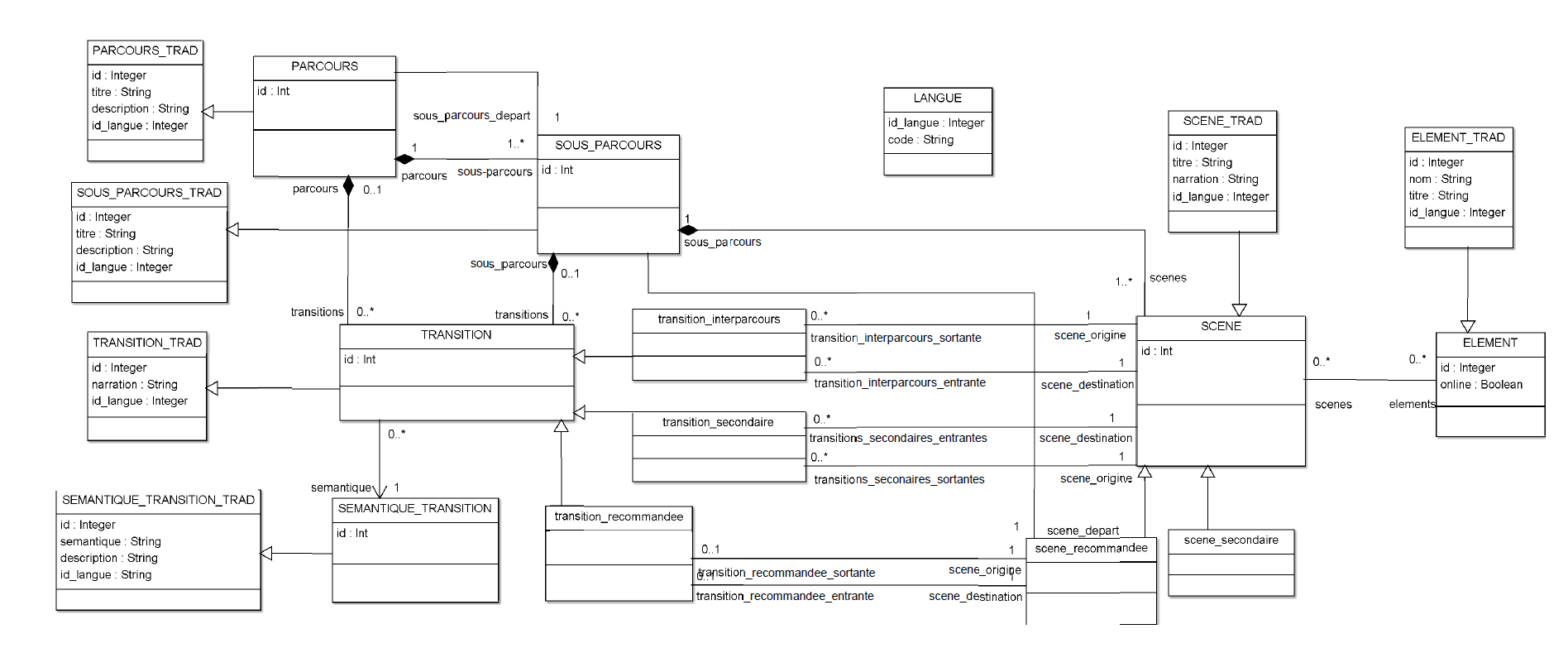
Nouvelle modélisation de la collection :



La classe LANGUE est reliée aux classes SELECT\_TRAD, SELECTOPTION\_TRAD, DATA\_TEXTE\_TRAD, DATA\_TEXTEAREA\_TRAD, ELEMENT\_TRAD, SEMANTIQUE\_ARTEFACT\_TRAD, CHAMP\_TRAD et TYPE\_ELEMENT\_TRAD.

Nouvelle modélisation du parcours :

La classe LANGUE est reliée aux classes PARCOURS\_TRAD, SOUS\_PARCOURS\_TRAD, TRANSITION\_TRAD, SEMANTIQUE\_TRANSITION\_TRAD, SCENE\_TRAD et ELEMENT\_TRAD.



Changements dans la gestion des utilisateurs

changements lors de la creation de contenu

changements dans l’ihm