Hugo PEREIRA, Clément MARTINS, Cyril PETER, Hans HOOKOOM, Alexandre BIDAUX

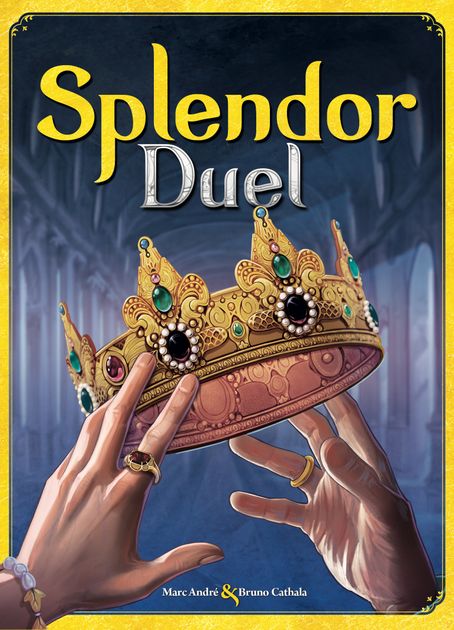
Une image contenant texte, Police, Graphique, logo

Description générée automatiquement

AI22/LO21 – Splendor Duel

Rapport Final

Une image contenant Graphique, cercle, symbole, capture d’écran

Description générée automatiquement

Une image contenant symbole, Graphique, Police, logo

Description générée automatiquement

Table des matières

[Résumé de ce que permet notre application 2](#_Toc154225782)

[Description de notre architecture 3](#_Toc154225783)

[Description générale 3](#_Toc154225784)

[Composants logiques 3](#_Toc154225785)

[Composants UI 3](#_Toc154225786)

[ModulaRité de l’architecture et améiliorations possibles 3](#_Toc154225787)

# Résumé de ce que permet notre application

Depuis début septembre, nous travaillons sur une application graphique permettant de jouer au jeu Splendor Duel sur une machine.

Cette application est aujourd’hui terminée et permet les actions suivantes :

* Jouer contre un joueur adverse
* Jouer contre une IA rudimentaire
* Changer la langue du jeu

Une fonctionnalité que nous n’avons pas eu le temps de finir à cause d’une trop grosse dette technique est la possibilité de sauvegarder sa partie. Les composants de sauvegarde existent et fonctionnent mais afin de gagner du temps dans la conception nous avons fait un découplage trop fort entre l’UI et les composants logiques du jeu ce qui empêche une modification en direct des composants logiques impactant l’UI.

# Description de notre architecture

## Description générale

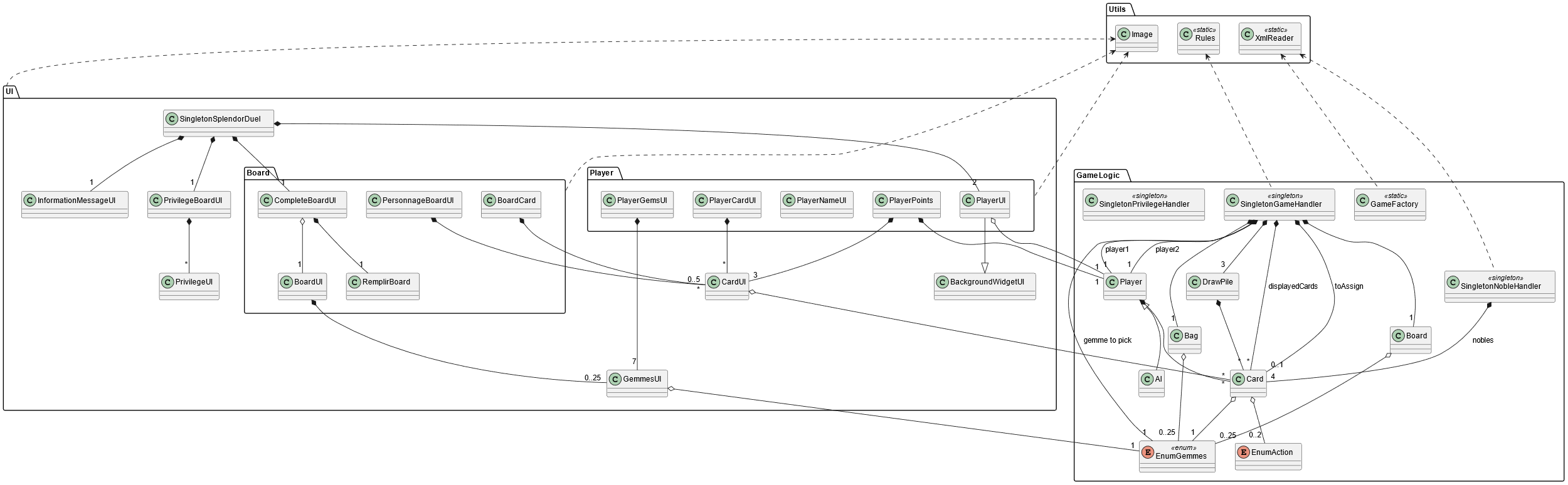
Notre architecture est décomposée en 3 grandes briques :

* Composants logiques
* Composants UI
* Classes utilitaires

Le cœur de l’application repose sur les composants logiques (et certains utilitaires desquels dépendent certaines classes logiques) qui ont les capacités de gérer leurs interactions avec les autres classes logiques ainsi que leurs propres attributs.

**En théorie**, nos composants UI viennent agréger nos composants logiques et servent d’intermédiaires pour l’affichage et l’interaction avec ces classes. Cependant, nous avons accumulé une dette technique sur nos composants UI en ne les faisant que s’appuyer légèrement sur les classes logiques. Par exemple, la classe servant à afficher les cartes possédées par un joueur ne se met pas à jour à travers les cartes possédées par la classe Player mais à travers le click de l’utilisateur. Il est tout à fait logique que lors de l’achat d’une carte, nous agrégions la carte sélectionnée à notre classe logique, ce qui se répercutera dans la classe UI. Ici, nous ajoutons la carte à notre classe logique et directement à notre classe UI qui ne dépend donc pas de la classe UI pour connaitre les cartes à afficher. Cela nous a permis de plus rapidement développer notre application mais a mené à de la redondance d’information et surtout un découplage entre les composants logiques et UI qui nous ont bloqué pour l’ajout de certaines fonctionnalités.

Voici un MCD représentant notre architecture actuelle :



## Composants logiques

Nos composants logiques sont chargés au lancement de l’application avec le nombre par défaut de chaque élément du jeu.

A l’aide d’un fichier XML nommé cards.xml, on génère l’intégralité des cartes du jeu. Nous répartissons ensuite ces cartes dans les différentes piles de cartes. Les piles de cartes sont ensuite mélangées et distribuées sur le plateau de jeu.

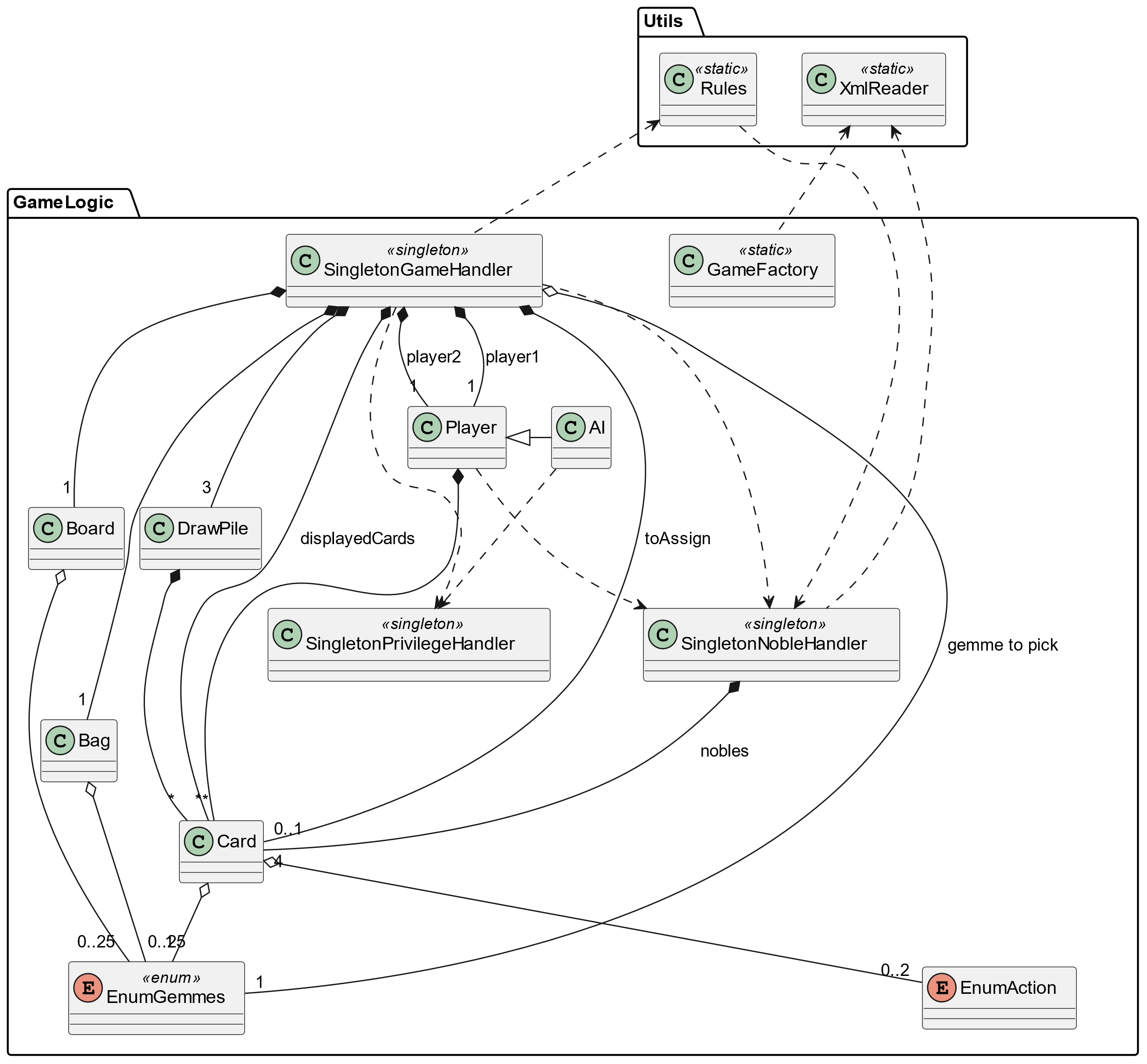
On génère également les 25 gemmes du jeu dans les proportions de base (4 gemmes bleu, blanc, rouge, vert, noir ; 3 gemmes or ; 2 gemmes perles) et on les met dans le sac du jeu. On mélange le sac pour ensuite remplir le plateau du jeu avec les gemmes. Le plateau du jeu étant complexe à naviguer (en spirale à partir du centre), on a créé un itérateur qui permet de naviguer correctement le plateau tout en ne partageant pas la structure de données.

On crée ensuite les joueurs avec leurs noms et aucune possession. Si l’utilisateur a choisi d’affronter une IA, on remplace un des joueurs par une IA nommée « IA ».

A partir de ce moment, on instancie les classes SingletonSplendorDuel et SingletonGameHandler qui sont respectivement les classes principales pour l’UI et la logique du jeu.

Cette génération à lieu dans une Factory, le but est de pouvoir plus tard remplacer cette Factory par une autre dans le cas ou le jeu serait amené à évoluer.

Voici un MCD présentant notre architecture logique :



## Composants UI

Nos composants UI servent à gérer les interactions entre l’utilisateur et la logique.

Ainsi, nous répartissons nos composants UI en 4 parties :

* Les composants « Généraux »
* Les composants « Board » qui concernent le plateau
* Les composants « Menu » qui concernent les différents menus du jeu
* Les composants « Player » qui concernent le joueur

Les composants « Généraux » sont appelés un peu partout dans l’UI (on peut penser aux cartes par exemples qui sont affichées chez le joueur et sur le plateau de jeu). On y retrouve des classes abstraites comme CardContainerGUI et GemmesContainerGUI qui imposent la gestion d’évènements comme click ou le scroll sur d’autre composants UI. On y retrouve la classe SingletonSplendorDuel qui est la classe mère de toute l’UI du jeu.

Les composants « Board » concernent l’affichage du plateau, on y retrouve les classes UI chargées de gérer les cartes affichées sur le plateau (BoardCardUI), le plateau avec les gemmes (BoardUI) ou encore la gestion de l’affichage des cartes nobles (PersonnageBoardUI).

Les composants « Menu » concernent les petits menus rapides qui s’affichent au lancement de l’application et à la fin d’une partie.

Les composants « Player » concernent l’affichage des informations concernant les joueurs. On y retrouve des classes chargées d’afficher les cartes du joueur (PlayerCardUI), ses points (PlayerPointsUI) ou plus simplement son nom (PlayerNameUI). La classe qui compose toutes les classes de cette catégorie est PlayerUI.

Voici un MCD présentant notre architecture UI :

## ModulaRité de l’architecture et améiliorations possibles

Notre architecture est assez permissive en ce qui concerne la modification des classes. Cela est principalement dû à la GameFactory qui à comme responsabilité l’instantiation des classes. Si par exemple je modifie la classe Card (qui a de nombreuses dépendances) de telle sorte a ce que son interface reste la même mais que ses attributs ou sa construction change, nous n’avons qu’à modifier la GameFactory pour que la modification soit faite correctement. De plus, nous pouvons ajouter/retirer des cartes simplement en modifiant le fichier cards.xml qui les gère.

Cependant, comme déjà mantionné dans ce rapport, nousa