Rapport

TRAVAIL DE BACHELOR

Quentin Forestier

TB – 08.04.2022

Table des matières

Cahier des charges

Problématique

Le but de ce travail de Bachelor est de répliquer et améliorer les fonctionnalités de Slyum, un éditeur de diagramme de classes UML développé en Java à la HEIG-VD. Il est notamment souhaité que l'application puisse proposer une collaboration sur les diagrammes, sans devoir passer le fichier entre les différents utilisateurs.

Solutions existantes

- Slyum
- StarUML
- Umletino
- …

Objectif

L'objectif de ce travail est de permettre d'avoir un éditeur de diagramme de classe sur le web, avec une collaboration simplifiée. Une collaboration instantanée est envisagée, mais dépendra du temps à disposition.

Jalon

Dans un premier temps, il faudra définir un meta-schéma au moyen d'un diagramme de classes.

Dans un deuxième temps, il s'agira de se familiariser avec le Framework Play!, et de voir dans quelle mesure il est possible d'utiliser les websockets afin de pouvoir collaborer. Une fois cela fait, il faudra choisir si oui ou non, il est possible d'avoir une coopération en direct sur l'édition de diagrammes. Un second diagramme viendra s'ajouter afin de modéliser les messages échangés entre le serveur et le client.

Dans un troisième temps, il faudra effectuer une recherche sur les différentes librairies JavaScript permettant un affichage et des modifications simples et intuitives du diagramme. Puis, un autre diagramme de classes viendra compléter le meta-schéma, représentant les informations graphiques.

Ensuite, il faudra pouvoir sauvegarder les diagrammes de classes dans une base de données PostgreSQL. Pour cela, il faudra rechercher l'ORM le plus adapté et le mettre en place.

Et enfin, l'ajout de la gestion de projets, d'utilisateurs et de groupes sera mis en place. Il sera alors possible de s'inscrire et se connecter, afin d'accéder à ses propres projets, ainsi qu'aux projets des équipes dont l'utilisateur est membre.

Une fois la conception du meta-schéma, ainsi que la familiarisation avec Play! effectué, l'application et ses fonctionnalités seront codés en Java pour la majeure partie, et en JavaScript pour l'affichage du diagramme. L'évolution de l'application suivra les jalons, de sorte que chaque étape soit déjà un résultat.

Fonctionnalités de l'application

Fonctionnalité principale du diagramme

- Création de classes, interfaces, énumération, classes abstraites
- Création d'attributs et de méthodes (abstraites ou non)
- Création d'associations (binaire, n-aire, compositions, agrégation, classes d'association)
- Création de relation d'héritage
- Création de relation de dépendances

- Affichage des éléments sous format graphique
- Possibilité de modifier graphiquement le diagramme
- Exportation sous format graphique
- Sérialisation/Désérialisation des diagrammes

Fonctionnalité supplémentaire du diagramme

- Possibilité d'effectuer une annulation de la dernière action
- Possibilité d'avoir différentes vues du diagramme
- Possibilité de dupliquer une entité du diagramme
- Mise en place de raccourcis clavier
- Possibilité d'écrire les attributs/fonctions sous forme de texte, qui sera ensuite transformé en entité

Fonctionnalité de gestion

- Inscription/Connexion
- Création de projets
- Création de groupes d'utilisateurs
- Gestion des droits dans le groupe
- Gestion des membres du groupe
- Quitter un groupe
- Ouverture des diagrammes de classes
- Coopération directe / indirecte sur un diagramme
    - Dans le cas d'une coopération instantanée, l'accès aux autres utilisateurs sera bloqué sur l'élément du diagramme sélectionné.
    - Dans le cas d'une coopération indirecte, l'accès au diagramme sera bloqué si un autre utilisateur travaille déjà dessus

Échéance

Date	Tâche
14 avril 2022	Cahier des charges
16 mai 2022	Rapport intermédiaire
29 juillet 2022	Rendu des livrables
26 août 2022	Résumé publiable

Livrables

- Une application Java utilisant le framework Play! déployée sur une machine virtuelle
- Un protocole de tests afin de garantir la fonctionnalité de l'application
- Un rapport comprenant :
    - Les choix de conception
    - Les problèmes rencontrés
    - Les solutions envisagées
    - La synthèse du résultat obtenu
- Un manuel utilisateur décrivant les fonctionnalités de l'application

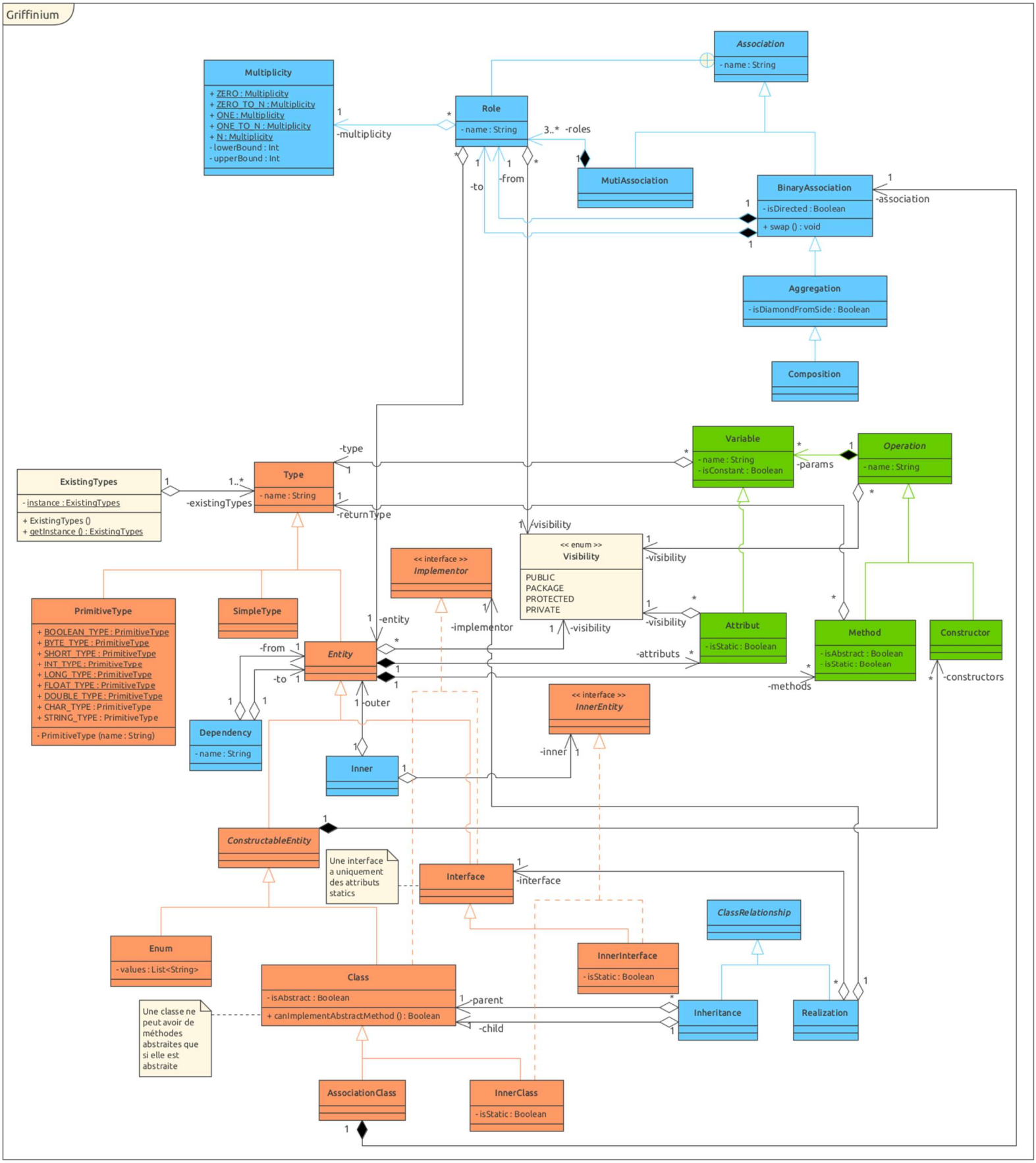
Planning

Tâches	Échéance
Meta-schéma	1er avril 2022
Familiarisation avec Play! et application de test	22 avril 2022

Recherche et mise en place dans l'application de test de la librairie graphique JavaScript	13 mai 2022
Fonctionnalités principales du diagramme	17 juin 2022
Ajout de la base de données pour sauvegarder les diagrammes	1er juillet 2022
Fonctionnalité de gestion	15 juillet 2022
Fonctionnalité supplémentaire du diagramme	29 juillet 2022

Les tests et la documentation seront en accord avec les différentes étapes mentionnées ci-dessus.

Meta-schéma


Griffinium
Association
- name : String
Multiplicity
+ ZERO : Multiplicity
+ ZERO_TO_N : Multiplicity
+ ONE : Multiplicity
+ ONE_TO_N : Multiplicity
+ N : Multiplicity
- lowerBound : Int
- upperBound : Int
-multiplicity
Role
- name : String
3..* -roles
-to
-from
MutiAssociation
BinaryAssociation
- isDirected : Boolean
+ swap () : void
-association
Aggregation
- isDiamondFromSide : Boolean
Composition
-type
Variable
- name : String
- isConstant : Boolean
-params
Operation
- name : String
ExistingTypes
- instance : ExistingTypes
+ ExistingTypes ()
+ getInstance () : ExistingTypes
-existingTypes
Type
- name : String
-returnType
<< interface >>
Implementor
-visibility
<< enum >>
Visibility
PUBLIC
PACKAGE
PROTECTED
PRIVATE
PrimitiveType
+ BOOLEAN_TYPE : PrimitiveType
+ BYTE_TYPE : PrimitiveType
+ SHORT_TYPE : PrimitiveType
+ INT_TYPE : PrimitiveType
+ LONG_TYPE : PrimitiveType
+ FLOAT_TYPE : PrimitiveType
+ DOUBLE_TYPE : PrimitiveType
+ CHAR_TYPE : PrimitiveType
+ STRING_TYPE : PrimitiveType
- PrimitiveType (name : String)
SimpleType
-entity
-implementor
Attribut
- isStatic : Boolean
-attributs
Method
- isAbstract : Boolean
- isStatic : Boolean
-methods
Constructor
-constructors
-from
-to
Entity
-outer
Dependency
- name : String
<< interface >>
InnerEntity
-inner
Inner
ConstructableEntity
Une interface a uniquement des attributs statics
Interface
-interface
ClassRelationship
Enum
- values : List<String>
Class
- isAbstract : Boolean
+ canImplementAbstractMethod () : Boolean
InnerInterface
- isStatic : Boolean
Inheritance
Realization
-parent
-child
Une classe ne peut avoir de méthodes abstraites que si elle est abstraite
AssociationClass
InnerClass
- isStatic : Boolean

Description du meta-schéma

En orange

Toutes les représentations de types et entités possibles.

PrimitiveType représente les types primitifs, les types sont donc exhaustifs.

SimpleType permet d'ajouter un type, qui est simplement une String, ce qui permet donc d'avoir un type qui n'est pas représenté par une entité (utile dans le cadre d'une utilisation de librairie).

L'interface Implementor sert uniquement pour le lien de réalisation. Elle permet de spécifier quelles entités peuvent impléme nter une interface.

L'interface InnerEntity sert uniquement pour le lien Inner. Elle permet de spécifier quelles entités peuvent être dans une autre entité.

En vert

Tous les attributs et méthodes disponible dans les entités.

En bleu

Tous les liens possibles d'un diagramme de classe.

Role permet, pour chaque entité de l'association, d'avoir un nom, une visibilité et une multiplicité.

En blanc

ExistingTypes est un singleton qui permet d'avoir une liste des types existants. Le changement de nom d'un type se répercutera sur toutes les variables et opérations l'ayant pour type.