Thomas Guérin
Florian Guif
Thomas Londechamp
Nicolas Miel
Pierre Moreau

Projet Android - Andodab

M2 Informatique -- 2014-2015

Documentation développeur

Organisation de l'équipe

L'équipe est composée de 3 étudiants en M2 Logiciels :

- Thomas Guérin
- Florian Guif
- Thomas Londechamp

Et de 2 étudiants en M2 SIAW:

- Nicolas Miel
- Pierre Moreau

Pour ce qui est de la répartition des tâches, Nicolas et Pierre se sont occupés de la gestion de la base de données, aussi bien au niveau fonctionnel que structurel (BackEnd) et Thomas G., Florian et Thomas L. se sont occupés de la partie sur la visualisation / modification des objets (FrontEnd).

Architecture

Les différents composants de l'application sont les suivants :

- ContentProvider
- Tables SQLite pour les objets
- Interface entre les activités et le ContentProvider
- Activités (modification / galaxie / synchro des objets)
- Adaptateurs pour gérer l'affichage des objets
- Client / Serveur pour la synchronisation des données

I - ContentProvider / SQLite

La base de données se divise en plusieurs tables, selon l'interprétation que nous avons fait du sujet :
Une table Objet est présente afin de représenter l'intégralité des objets qui ont un dictionnaire ainsi que ceux qui sont des parents d'objets primitifs.

Chaque objet de cette table, excepté l'objet racine, a pour ancêtre soit un objet prédéfini, soit l'objet racine (l'objet racine étant compris dans la table représentant les objets possédant un dictionnaire mais ayant son dictionnaire nul).
Dans une table séparée, nous avons stocké les objets primitifs, ayant pour ancêtre un objet parent primitif.
Dans une dernière partie, nous avons considéré les Entry (représentant les différentes entrées de dictionnaires d'un objet). Chaque entry a un nom mais aussi une valeur, qui peut, soit représenter un Objet (un objet avec dictionnaire ou un objet parent primitif) soit représenter un objet primitif. Nous utilisons donc une hiérarchie à la fois pour les objets mais aussi pour les entry.
Nous associons ensuite dans une table les objets possédant un dictionnaire avec les entry lui correspondant (ce qui représente les propriétés d'un objet).

En plus des contraintes imposées par le sujet, nous avons choisi de représenter les parents des objets primitifs (Integer, Float et String) avec des identifiants en "dur", ce qui permet de faciliter la synchronisation. Nous avons utilisé le même principe pour l'objet racine.
Les identifiants des autres objets sont des Long générés aléatoirement afin de ne pas avoir de duplication possible.

Nous avons fait en sorte que le ContentProvider autorise la création d'un objet et gère automatiquement dans quelle table (en plus de la table Object) insérer le nouvel objet créé (soit dans la table DicoObject, soit dans la table OPPrimitive). Le même principe est également utilisé pour la création d'une Entry.

Une gestion de la suppression d'objets est implémentée : dans un premier temps, nous supprimons tout d'abord toutes les entry ayant pour valeur l'objet à supprimer. Dans un second temps, nous supprimons toutes les associations entre les entry et l'objet à supprimer, puis, une suppression est faite dans la table correspondante à l'objet supprimé (table DicoObject).
Enfin, l'objet est supprimé de la table Objet (une fois que toutes les références lui étant faite dans la base sont supprimées).

Nous implémentons également dans l'interface des fonctions facilitant la manipulation de la base, comme des fonctions de récupération d'un objet (récupérant ses caractéristiques dans la base), une fonction de récupération de tous les objets, une fonction de récupération des enfants d'un objet (en se basant sur le champ "ancestor" de la table ainsi qu'une fonction de récupération des propriétés d'un objet (qui récupère

en réalité, avec nommage, les associations entre un objet et les entry de son dictionnaire).
La fonction la plus importante et la plus compliquée étant celle de mise à jour d'une propriété d'un objet, étant donné toutes les vérifications et les contraintes sur les tables à respecter.

II - Galaxie des objets

L'activité de galaxie des objets utilise un layout et des view personnalisés. Chaque view représentant un rectangle avec en titre le nom de l'objet et en contenu la liste de ses propriétés. Pour paramétrer la taille de la fenêtre, chaque vue se mesure en fonction de la plus grande chaîne de caractères qu'elle contient (soit le titre de l'objet, soit la taille d'une ligne au niveau des propriétés) pour la largeur, la hauteur est calculée par rapport au nombre de propriétés.

Le layout calcule alors la largeur maximale en prenant comme valeur la plus grande largeur d'un des niveaux de l'arbre et comme hauteur la somme des hauteurs de tous les niveaux de l'arbre. Modulo les marges définies comme constante de la classe, on calcule la taille de la fenêtre nécessaire à l'affichage de l'arbre.

Un fois les vues disposées sur le Layout, ce dernier se charge de dessiner les lignes pour donner l'impression d'un arbre en utilisant les positions des vues.

Il est possible de se déplacer et de zoomer au niveau de la Galaxie, cela se traduit simplement par une translation et un facteur d'agrandissement (scale) que le layout applique sur les vues lors de l'affichage.

La galaxie est paramétrable via deux valeurs envoyées dans l'intent :

-allowChoose : un booléen vrai par défaut qui définit si l'utilisateur peut choisir un objet avec un appui long.

-rootId : l'id de l'objet à afficher à la racine de l'arbre permet de restreindre l'arbre à une certaine partie de la base de données.

III - Création et édition des objets

Lors de la création d'un objet, on demande à l'utilisateur de sélectionner un Objet parent au travers de l'activité Galaxy. La Galaxy des objets nous renvoie l'id du Parent sélectionné, on insère alors un nouvel objet dans la base de données avec comme père l'objet sélectionné.

On dirige alors l'utilisateur vers l'activité EditObjetActivity qui récupère dans l'intent l'id de l'objet à éditer.
L'activité propose à l'utilisateur une interface lui permettant d'éditer facilement toutes les propriétés d'un objet.
L'activité affiche toutes les informations de l'objet au travers d'editView, TextView et d'une listView qui affiche les propriétés d'un objet grâce à custom adaptater.

Lorsqu'un utilisateur souhaite ajouter une propriété, on fait appel à une nouvelle activité proposant à l'utilisateur une liste de propriétés provenant de ses parents. Il peut, soit redéfinir cette liste soit créer une nouvelle propriété. Cette activité propose aussi un système de recherche sur la listeView des propriétés des parents, fonction très utile lorsque la liste des propriétés devient très importante.

Lorsque l'utilisateur crée / édite / redéfini une propriété il doit choisir la valeur de propriété au travers de l'activité Galaxy, ou bien si la propriété est une primitive (String, Float, Integer), l'utilisateur peut alors directement indiquer la valeur de la propriété.

L'activité EditObjetActivity effectue les contrôles "logiques" avant d'insérer des données grâce au ContentProvider, cela garantit l'intégrité de la base de données.
Par exemple, si le parent d'un Objet est "sealed", l'utilisateur n'a pas la possibilité de changer la valeur "sealed" de cet Objet. Ou bien encore, lors de la redéfinition d'une propriété, l'utilisateur ne peut pas choisir une valeur moins spécialisée dans la base de données.

L'utilisateur peut aussi supprimer un objet à l'aide d'un bouton situé au bas de l'écran. Celui-ci fait appel à une méthode du ContentProviderUtil qui va assurer l'intégrité de la base de données lors de la suppression.

Difficultés rencontrées

Au niveau de la base de données, une des grosses difficultés que nous avons eu est que l'option “delete on cascade” ne fonctionnait pas. De ce fait, nous avons dû implémenter manuellement ce procédé de cascade à travers les tables.