Master 1 - Ingénierie Informatique

Projet Android : DLNApp

Benoit Gallet
Emmanuel Herrmann
Bryce Marbois
Martin Rety

Table des matières

1 Introduction

1.1 DLNA et protocole UPnP

DLNA est une norme facilitant le partage des fichiers multimédias entre différents périphériques sur un réseau domestique. Certains de ces périphériques sont des serveurs (comme un ordinateur ou un disque dur branché sur une box), partageant des dossiers et des fichiers à des clients (comme par exemple un téléphone portable ou une tablette possédant notre application).
DLNA repose sur le protocole UPnP pour interagir avec le réseau. Ce protocole permet de connecter des périphériques afin de simplifier les échanges entre eux, et se compose de deux phases. La première est la découverte des services présents sur le réseau et permet également au périphérique d'indiquer ses propres services au serveur. La seconde est la récupération des informations des périphériques connectés. Nous pouvons à partir de cela communiquer avec les services des différents périphériques tout en informant des changements effectués.

1.2 La bibliothèque Cling-Core

La bibliothèque Cling-Core fournit une architecture complète qui aide le développeur à implémenter toutes les fonctionnalités nécessaires à UPnP dans son application. Cling-Core possède un grand nombre de classes et d'objets pour gérer la découverte et la connexion des différents périphériques, et permet aussi de naviguer dans les répertoires partagés grâce à une architecture DIDL.
Cling-Core gère de façon interne les opérations réseau entre les périphériques (client et serveur) de manière asynchrone. Cela permet à l'utilisateur de ne pas avoir à se soucier de l'implémentation via une `AsyncTask` de ces tâches. L'utilisation de cette bibliothèque est très répandue, la plupart des applications du PlayStore y ont recours, notamment la plus importante, BubbleUpnp. Cela fournit plusieurs exemples d'utilisation permettant de mieux comprendre comment marche Cling-Core, car la documentation se révèle relativement pauvre une fois passées les explications générales.

2 Structure du projet

2.1 Architecture DIDL et gestion de la structure de fichiers : Package MyObject

Le type DIDLObject (Digital Item Declaration Language) est une architecture décrivant une arborescence d'objets complexes. Celle-ci se compose de la manière suivante :

- DIDL : contient un ou plusieurs containers
- Container : contient d'autres containers ou des items
- Item : contient des items ou des components

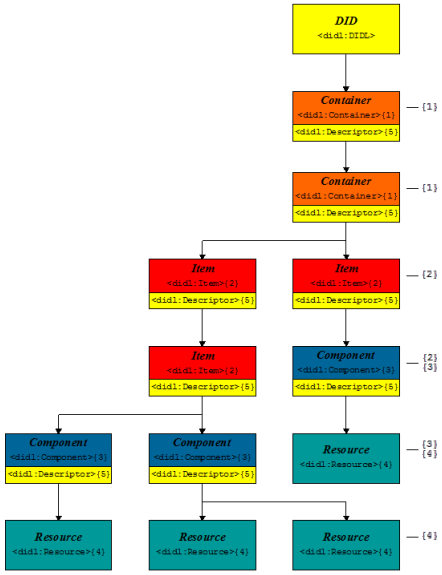

Figure 1. Architecture DIDL

— Component : contient des resources
— Resource : flux de données

Notre modélisation reprend le même schéma, mais en omettant deux niveaux de profondeur (Item et Component). Ainsi le DIDL est représenté par MyObjectDevice (le périphérique), Container par MyObjectContainer (un répertoire) et Resource par MyObjectItem (un fichier).

Le package MyObject est constitué d'une classe éponyme, regroupant les informations communes dont hériteront les trois autres classes : une icône, un titre et une description. Ces trois autres classes sont :

— `MyObjectDevice` possède deux attributs supplémentaires, urlIcon et Device. Etant donné qu'un serveur peut posséder sa propre icône, nous la récupérons si elle existe dans urlIcon afin de l'afficher à la place de l'icône par défaut. Quant à Device, ce dernier permet de regrouper toutes les informations relatives à un serveur ainsi que différents services tels que le ContentDirectory qui permet de naviguer dans le système de fichier du Device.
— `MyObjectContainer` possède un attribut de type DIDLObject, ainsi qu'un Service. Ce Service est en fait un AndroidUPnPService permettant de découvrir tous les fils du container courant. Ils servent à parcourir une architecture de type DIDL.
— `MyObjectItem` possède également un DIDLObject qui permet d'obtenir les informations d'une ressource, et un String afin d'en connaître son type. Celui-ci est récupéré via le type MIME du DIDLObject.

2.2 Gestion du protocole UPnP : Package UpnpConnection

Les classes Browse et RegistryListener gèrent toutes les interactions liées au protocole UPnP et donc les appels des fonctions de Cling-Core, comme les communications entre le client et le serveur, l'écoute des transactions et la gestion de l'architecture DIDL.

Ce package contient les classes suivantes :

- `Browser` possède un Service permettant de faire la jonction avec le Service Cling-Core androidUpnpService. Ce dernier gère une pile de Device, et l'ajout du RegistryListener sur celle-ci nous permet d'ajouter ou de supprimer un Device ou de rafraîchir la pile. Celui-ci gère également l'architecture DIDL contenue dans chaque Device en permettant de l'explorer à des niveaux de profondeur donnés. On peut par exemple naviguer à travers l'arborescence d'un serveur grâce à browseDirectory. La classe permet au final d'initialiser, de gérer et de terminer les différentes connexions à des Devices. Elle s'occupe, en outre, de créer les MyObject fils du Device ou du Container.
- `MyHandler` récupère un DIDLContent et en extrait une liste de MyObject.
- `RegistryListener` gère les évènements qui surviennent sur le réseau tels que la connexion ou la déconnexion d'un serveur, etc.

Pour effectuer différentes actions sur AndroidUpnpService nous utilisons la méthode getControlPoint(), donnant accès à différentes actions, comme search() qui recherche de manière asynchrone la liste des Device.

2.3 Lecteurs multimédias : Package MediaActivities

Lors du parcours de l'arborescence, si l'utilisateur sélectionne une ressource, alors il a le choix de sélectionner une application tierce présente sur son téléphone, ou à défaut d'ouvrir le fichier sur un des lecteurs multimédia que nous fournissons.
L'activité MyImageViewActivity permet ainsi d'ouvrir toutes les ressources de type image présentes dans le dossier courant. Il est possible de swipe à gauche ou à droite pour faire défiler les images.
MyVideoMusicViewActivity sert quand à elle à ouvrir les fichiers audio et vidéo. Nous rencontrons malheureusement quelques problèmes avec certaines extensions de fichiers qui ne sont pas compatibles de base avec Android. Ainsi un fichier dont l'extension est .mkv n'aura pas de son par exemple.

Pour savoir quel lecteur sera utilisé, nous procédons de la manière suivante : Si la ressource est une image alors elle sera lue par notre lecteur pour pouvoir bénéficier des fonctionnalités que nous avons implémentées. Sinon, si la ressource peut être ouverte avec une application déjà installée sur le téléphone, celle-ci est lancée. Sinon, si la ressource est de type vidéo ou audio, elle sera lue via la classe MyVideoMusicViewActivity. Sinon, si aucune de ces conditions n'a été vérifiée, on ouvre la ressource avec le navigateur web par défaut du smartphone.

2.4 Algorithmique générale

L'activité principale de notre application est la classe MainActivity. Elle permet d'initialiser les différentes variables inhérentes à l'application, comme l'ArrayList de MyObject correspondant à la liste affichée à l'écran. Cette liste est mise à jour grâce à MyAdapter (qui étend ArrayAdapter). MyAdapter permet également de mettre en relation les différents éléments de la vue avec les attributs d'un MyObject.

MainActivity initialise un fragment : Browser. Ce fragment possède ServiceConnection qui permet de gérer AndroidUpnpService. Lors de son initialisation, une instance du RegistryListener est attaché à chacun des Devices trouvés. Ce fragment permet également de trouver tous les fils d'un MyObject suivant son type. Pour pouvoir mettre à jour la liste de MyObject, il utilise la classe MyHandler. Celle-ci permet de mettre à jour l'ArrayList en créant les MyObject par rapport à un DIDLContent.

Toutes ces classes peuvent communiquer avec MainActivity via l'interface Notification, qui permet d'effectuer certaine action dans le thread principal, comme l'affichage des différents Devices ou le rafraîchissement d'une page.

2.5 Autres fonctionnalités

Diverses fonctionnalités ont été implémentées dans notre application afin de la rendre plus ergonomique pour l'utilisateur.

2.5.1 Rafraîchissement de la page Il existe deux manières de rafraîchir l'affichage du contenu d'une page. L'utilisateur peut soit appuyer sur un bouton dans le menu, soit faire un swipe vers le bas. Ces deux actions appelent la méthode refresh() de Browser qui met à jour l'ArrayList de MyObject. Toutefois, il existe un rafraîchissement automatique de la liste de Device, incluse dans la bibliothèque Cling-Core qui le fait de manière optimale.

2.5.2 Bibliothèque Picasso Comme l'affichage d'une image à partir d'une URL n'est pas réalisable nativement avec Android, nous avons dû utiliser une bibliothèque externe pour le faire. Nous avons choisi Picasso car c'est une des plus répandue du fait de sa légèrete et de sa simplicité d'utilisation.

2.5.3 Gestion du bouton "back" Par défaut le bouton Back d'Android réduit l'application en la mettant en arrière-plan. Nous l'avons modifié pour que celui-ci remonte dans l'architecture DIDL.

3 Analyse du travail

3.1 Universalité

Dans la réalisation de cette application, nous avons essayé d'être le plus "universel" possible notamment via :

— L'affichage des chaînes de caractères dans différentes langues suivant les paramètres régionaux de l'utilisateur. Pour l'instant seuls le francais et l'anglais sont intégrés dans l'application.
— Les images faisant partie de l'application sont disponibles dans différentes tailles afin de s'adapter à tous les écrans. Nos affichages personnalisés s'adaptent automatiquement aux changements d'orientation de l'appareil afin de toujours garder un rendu ergonomique.
— Toutes les images utilisées dans l'application (comme celles du répertoire et du fichier) sont libres de droits, hormis le logo de l'application. En effet, celui-ci sera surement changé par la suite selon l'évolution de l'application.

3.2 Organisation du travail

Notre travail s'est décomposé en deux grandes parties. La première a été la phase de documentation. En effet, il a fallu apprendre à utiliser la bibliothèque Cling-Core grâce à la documentation disponible sur le site, et à différents exemples présents sur GitHub et StackOverflow. Cela a pris la moitié du temps passé sur le projet, car la documentation a rapidement montré certaines limites, et que nous prenions soin de toujours rapporter par écrit ce que nous en avions compris. La deuxième partie fût le codage à proprement parler, même si le travail de documentation continua de façon plus ténue. Nous avons partagé le projet en différents modules, afin d'avoir un noyau solide sur lequel ajouter petit à petit différentes couches de fonctionnalités. Nous avons donc commencé par faire un squelette de l'application, puis à établir une connexion avec un périphérique, puis à parcourir ses fichiers et dossiers partagés, et enfin à lire correctement les différents flux. Une phase finale a tout de même été consacrée à des améliorations et débogages divers.

3.3 Problèmes rencontrés et améliorations possibles

3.3.1 Images différentes suivant le type de fichier Nous avions dans l'idée d'afficher pour chaque type de fichier (vidéo, musique, image, texte ou autre) une icône correspondante afin de pouvoir identifier rapidement ce que l'utilisateur s'apprêtait à ouvrir. Malheureusement, cette technique nécessitait de parcourir la liste des fichiers présents à chaque navigation dans l'aborescence. Cette opération nécessitant un certain temps lorsqu'il y avait beaucoup d'éléments dans la liste, le rafraîchissement de l'écran avait lieu avant la fin du calcul. Cela engendrait dans le meilleur des cas une disparition des icônes, et dans le pire un plantage complet de l'application. Cette idée a donc été abandonnée au profit d'une seule icône pour les fichiers et une autre pour les dossiers.

3.3.2 Noms de fichiers Nous avons utilisé les identifiants MIME afin de savoir de quel type est le fichier courant. Comme les noms de fichier sont des String et que la classe MIME doit parser des URI, les caractères spéciaux et les espaces empêchent le parsing de la chaîne de caractères. Nous avons donc

dû effectuer un formatage préalable remplaçant les caractères spéciaux par des caractères compatibles avec une URI.

3.3.3 Fichiers vidéos Plusieurs formats vidéo ne sont pas reconnu nativement par Android, comme le .mkv ou le .avi. On ne peut afficher que l'image et pas le son pour le .mkv, et pour le .avi cela dépend du fichier. Il était possible d'inclure une bibliothèque (Vitamio) pour pouvoir lire n'importe quel flux vidéo, mais, outre sa difficulté d'installation et le fait qu'elle ne soit plus maintenue depuis un certain temps, elle pèse 50 Mo, soit vingt-cinq fois plus que notre application.

3.3.4 Amélioration de la galerie Actuellement, la galerie ne permet que de visionner les images en les faisant défiler de droite à gauche et inversement. Une amélioration possible serait de pouvoir zoomer ou dézoomer sur l'image actuelle (pinch to zoom).

3.3.5 Télécharger les médias Une deuxième amélioration serait de pouvoir télécharger le média affiché suite lorsque l'utilisateur effectue une pression longue sur l'écran. Cela permettrait une consultation hors ligne des éléments téléchargés.

3.3.6 Préférences utilisateur Une autre amélioration importante serait d'enregistrer certaines préférences liées à l'utilisateur, comme un dark mode, l'affichage de miniatures des médias, ou de choisir nos lecteurs en plus de ceux installés sur l'appareil.

4 Conclusion

Dû au manque de temps pour développer l'application (trois semaines), nous n'avons pû faire que la première partie de l'application (le côté client), les deux autres étant la construction d'une partie serveur et le transfert de la lecture d'un flux en cours. Nous avons également effectué un gros travail de documentation, que ce soit dans un sens d'apprentissage (le fonctionnement de Cling-Core et d'UPnP en général), ou de retranscription de nos acquis à travers une javadoc conséquente ainsi que des commentaires dans le code et un rapport les plus complets possibles.

Ce projet nous a permis de consolider nos connaissances en Android, comme par exemple au niveau des notions de Fragment ou de Service, ainsi que de nous ouvrir sur un sujet particulier : les connexions dans un réseau domestique via le protocole DLNA.