Intégration d'objets connectés hétérogènes et démonstrateurs

Conception du framework et du démonstrateur

Adrien Casanova Victor Sallé
Marie-Catherine Turchini
Polytech Nice Sophia, SI5
{acasanov,salle,turchini}@polytech.unice.fr

Alexia Llorens
Université Nice Sophia Antipolis, M2
la002769@etu.unice.fr

Résumé

On assiste actuellement à une expansion et une démocratisation phénoménale des objets connectés qui envahissent notre quotidien. Beaucoup ont ainsi fait l'acquisition d'un bracelet sportif, d'une station météorologique connectée ou encore d'une montre intelligente. Au-delà de la multitude d'applications hétérogènes avec laquelle doit interagir l'utilisateur, se cache une autre problématique, plus technique : comment permettre aux développeurs d'homogénéiser l'accès aux objets et à leurs données ? La réponse à cette problématique se fera en fournissant un cadre de travail permettant de décloisonner leur usage afin de remonter l'ensemble des informations à un même niveau.

Suite à la classification des objets connectés en catégories faite auparavant, ce document décrit la phase de conception du framework en lui-même. Une vidéo présentant ce document sous une autre forme est également disponible sur YouTube[1].

Mots-clés Internet of Things, Objets connectés, API, Framework

1. Introduction

Jusqu'à présent, le travail de l'équipe a consisté en la compréhension du problème ; de cette phase en est sortie une classification des objets, c'est-à-dire un regroupement en familles. Le but de cette étape était de faciliter la connexion d'un nouvel objet connecté ; en effet, désormais le développeur qui voudra en ajouter un au framework aura juste à communiquer les informations basiques de celui-ci, tel que la protocole d'échange.

Après avoir ainsi classifié les objets connectés[2], les trois semaines qui ont suivi ont été consacrées à la conception du framework. Au cours de cette période, nous avons découpé le framework en différents blocs : chacun d'entre eux représente une étape essentielle. Certaines d'entre elles nécessitent une interaction avec le développeur qui ajoute son objet connecté, par exemple pour entrer les caractéristiques de celui-ci, les autres sont des actions effectuées par le système et seront donc exécutées en interne.

Ce document présente donc la description du fonctionnement du framework et explicite les raisons des choix qui ont été faits.

2. Suivi de projet

2.1 Budget

A ce stade du projet, la moitié du budget a été consommée. Les ressources humaines ont été placées différemment par rapport au plan initial : trois membres du groupe ont travaillé sur la conception du framework tandis qu'un dernier membre s'est intéressé à la conception de l'application qui servira à démontrer les résultats du projet.

2.2 Déviations par rapport au plan initial

La charge de conception du framework ayant été sous-estimée dans le cahier des charges, nous connaissons un retard d'une semaine sur le début de l'implémentation puisqu'à l'heure actuelle aucun code n'a été produit. Ce décalage n'est pas vraiment gênant dans la mesure où une conception plus poussée offrira une meilleure modularité et un ajout de familles d'objet simplifié, d'autant plus que la quantité de code ne sera à priori pas très conséquente.

3. Conception du framework

3.1 Point d'entrée (exposé)

Le point d'entrée de notre système représente le moyen pour le développeur de nous communiquer l'ensemble des informations nécessaires au système pour son bon fonctionnement, entre autres :

- le mode de communication : la technologie utilisée par l'objet pour renvoyer les données collectées, par exemple : Bluetooth, Wi-Fi ;
- l'URL à laquelle se trouve l'API de l'objet s'il en existe une ;
- l'appartenance à une famille déjà existante, par exemple lors de l'ajout d'un nouvel de la marque Withings.

L'ensemble des ces informations seront transformées en un fichier au format JSON. A noter que dans un premier temps, le site web ne sera pas développé. En effet, nous préférons nous axer sur le bon fonctionnement du framework, l'interface utilisateur est donc secondaire. Nous créerons donc des fichiers JSON de tests manuellement et nous les enverrons en local ; le bouchon sera levé au cours de la suite du projet, si le temps nous le permet.

Une fois les informations entrées et validées, un objet JSON les contenant est crée et envoyé au système afin qu'il analyse son contenu et déclenche des actions dans certains cas : par exemple, si l'URL d'une API a été indiquée, nous irons récupérer l'ensemble des fonctionnalités proposées à l'adresse communiquée. Si aucun WSDL/WADL n'est précisé pour un fonctionnement par web services, le développeur peut ajouter lui-même des fonctions manuellement.

Une fois cette étape finie, le développeur est à nouveau sollicité puisqu'une interface lui permettra d'effectuer le mappage entre les fonctions récupérées par exemple à l'URL de l'API et les fonctions ajoutées manuellement par le développeur, et les fonctions déjà présentes dans le système. L'un des intérêts majeurs de cette étape réside dans le fait de réunir des fonctionnalités identiques. Prenons l'exemple du pèse-personne Withings et du Nabaztag, tous deux renvoyant entre autre la température ambiante : on peut alors associer leurs fonctions respectives de récupération de la température,

de manière à ce que les fonctions *getTemperatureBalance()* et *getTemperatureNabaztag()* correspondent à une même fonction *getTemperature()*. Dans le cas où le Nabaztag serait éteint, la fonction getTemperature() renverrait donc automatiquement le résultat de getTemperatureBalance().

Le système contiendra une base de données des objets ayant déjà été configurés. Cela présente plusieurs avantages concernant les performances :

— Rapidité : on peut imaginer que dans certains cas, du code pourra être réutilisé, générant ainsi plus rapidement le proxy ;
— Robustesse : du code déjà existant est supposé être fiable, s'en servir est donc un gage de qualité supplémentaire.

3.2 Génération du proxy (interne)

Il s'agit là d'une étape exécutée en interne : elle consiste en la création et le stockage des différentes informations du système : les protocoles et modes de communication déjà pris en charge, les méthodes contenues dans l'API unifiée -l'API finale qui sera exposée-. Ce bloc sera développé en Java, car il s'agit du langage de programmation le mieux maîtrisé par l'ensemble du groupe, et qui propose des bibliothèques permettant d'effectuer des migrations de code vers un autre langage. Chaque famille d'objets sera ainsi représentée par une classe mère ou bien une interface ; quant aux différentes catégories de cette famille, elles seront instanciées par le biais de classes filles. Ainsi, le système sera facile à maintenir puisqu'il sera aisé d'y greffer un nouvel objet ou bien même une nouvelle famille, etc. A partir de ces données, nous allons pouvoir générer notre proxy.

3.3 Sortie du framework (proxy généré)

Il était important de réfléchir au protocole de communication utilisé entre l'application cliente et le proxy généré avant de concevoir ce dernier, afin de définir la manière dont seront exposés les services. Le framework doit générer un proxy, qui sera déployable sur plusieurs instances matérielles (ordinateur, box internet, etc) et pourra être mobile. L'application cliente ne peut alors pas faire d'hypothèse sur le point d'entrée du proxy : il faut donc le rendre découvrable. Nous avons d'abord pensé à un annuaire de services web local, mais cette solution implique une instance matérielle supplémentaire, de point d'entrée connu. Après plusieurs recherches et analyses, nous avons découvert le protocole UPnP, qui permet la découverte de services sur un réseau local, tout en offrant une gestion des évènements que nous recherchons également.

Le protocole UPnP[3] n'est pas standardisé mais est très répandu, et de nombreuses bibliothèques sont disponibles dans de nombreux langages. Il permet d'exposer des services web, qui sont décrits avec un format de description en XML propre au protocole (schéma UDA). Son successeur, le protocole DPWS décrit lui les services web avec un fichier WSDL tout en étant sécurisé. Malheureusement, moins d'outils sont disponibles, or, le but de ce projet étant de fournir une *preuve de concept* nous préfèrerons utiliser le protocole UPnP. L'aspect sécuritaire pourra être retravaillé plus tard au cours d'un autre projet.

Aussi, il fallait connaître le format de ce qui allait être généré. Deux solutions s'offraient à nous : la génération d'un exécutable, ou la génération d'un fichier de configuration utilisable sur un exécutable universel dont le paramètre d'instanciation serait ce fichier, toutes deux ayant ses avantages et inconvénients. La première permet d'obtenir un exécutable final beaucoup plus léger puisque seul le nécessaire est embarqué ; de même, il n'y a pas besoin de mettre à jour un gros exécutable lorsque de nouvelles familles apparaissent. La seconde permet d'obtenir un fichier très

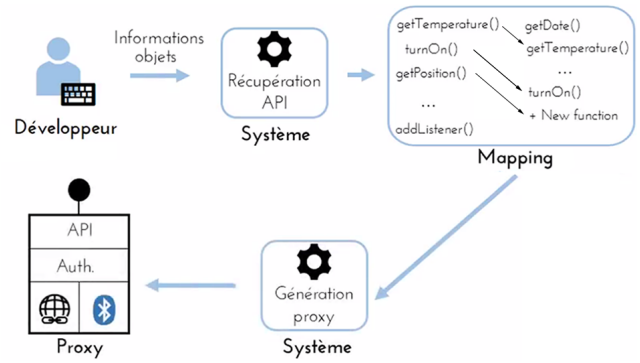

FIGURE 1. Fonctionnement du framework

léger, éventuellement modifiable à la main dans le cas de besoins spécifiques, mais oblige à embarquer un exécutable plus lourd, souvent sous-exploité. Par soucis d'assurer le passage à l'échelle, nous choisissons la première solution.

Le proxy généré sera donc un exécutable, écrit dans un code managé pour plus de simplicité, offrant des services web découvrables via UPnP et décrits à l'aide d'un fichier XML.

4. Conception du demonstrateur

Nous avons commencé la conception du démonstrateur. Le but de celui-ci est de montrer le bon fonctionnement de notre framework. Nous voulons vérifier que nous disposons bien d'un accès simplifié à un ensemble d'objets (un réseau d'objets) et que via cet accès nous pouvons visualiser toutes les informations concernant un objet. Les besoins se résument donc à la visualisation d'un ensemble de services accessibles, une fois un service choisi, nous voulons accéder aux données des objets qui composent ce service.

Pour le moment le choix de la technologie s'orienterait vers du web, plus adapté à l'utilisation de n'importe quel dispositif pour la visualisation des données. L'affichage de celle-ci se ferait via une interface sous la forme d'un tableau de bord, c'est-à-dire une zone délimitée pour un objet et ses informations mais avec la possibilité d'avoir des éléments regroupés, ceci à la convenance de l'utilisateur.

5. Conclusion

Ces étapes de conception nous ont donc pris plus de temps que prévu, mais elles s'avèrent indispensables pour continuer le projet de manière efficace.

Le plan d'action jusqu'au prochain jalon fixé au 25 janvier 2015 va être d'implémenter le framework, sans réelle interface graphique dans une première mesure. De même, nous finaliserons la conception du démonstrateur puis l'implémentation suivra. Cette dernière se fera donc pour le framework et le démonstrateur en parallèle et de manière itérative, en implémentant dans un premier temps les blocs permettant de faire interagir les objets connectés dont nous disposons actuellement (Withings Pulse O2, Withings Smart Body Analyzer et Nabaztag).

Références

[1] A. Casanova, A. Llorens, V. Sallé et M.-C. Turchini. *Vidéo de présentation* URL `http://youtu.be/Ov7bSU5JejU`

[2] A. Casanova, A. Llorens, V. Sallé et M.-C. Turchini. *D2 - Étude des API et explication des regroupements*, 2014

[3] Stéphane Lavirotte. *From Service Oriented Middleware to SoM for Devices*, 2014