Intégration d'objets connectés hétérogènes et démonstrateurs

Rapport final

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..

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

Table des matières

1. Introduction

L'informatique ubiquitaire est un des enjeux majeurs d'aujourd'hui et de demain. Depuis le milieu des années 2000, de nombreux objets connectés sont arrivés sur le marché et s'intègrent de plus en plus à notre quotidien. On les retrouve dans de nombreux domaines - téléphonie, santé, bien-être, domotique, ... - au travers d'appareils très divers tels que des téléphones, montres, tensiomètres, pèse-personnes, matelas, etc.

Cependant il existe un très grand nombre d'objets et la toile qu'ils constituent est très hétérogène. Ainsi, le défi d'aujourd'hui est d'intégrer ces objets dans l'environnement de l'utilisateur afin d'acquérir les informations propres à son contexte (heure, localisation, environnement, mobilité, ...) et lui proposer automatiquement une interaction adaptée. Il survient alors un réel besoin pour les développeurs d'avoir un cadre commun permettant de gérer l'hétérogénéité des dispositifs et de disposer d'un accès unifié à leur paramétrage et à leur utilisation. Ce cadre doit donc permettre de décloisonner l'utilisation de chaque dispositif de manière à pouvoir les associer pour créer de nouveaux usages et de nouvelles applications mettant en œuvre non plus un seul dispositif mais une constellation de dispositifs.

2. Problématique

La relation "un objet - une application" est un frein à la communication avec un ensemble d'objets connectés. En effet, chaque objet est cloisonné et restreint à sa propre application alors que, de façon instinctive, l'utilisateur préférerait n'utiliser qu'une unique application pour un groupe d'objets, en particulier s'il en réunit plusieurs d'un même domaine, dans le cadre de la santé par exemple. A la place, il doit faire face à de multiples applications et ne peut donc pas bénéficier d'une mise en commun des informations que peuvent lui fournir chacun des objets.

De ce fait, le premier obstacle que nous avons dû surmonter est l'appréhension de l'ensemble des technologies dans le but de répondre à notre objectif d'unifier le mode de communication entre objets et applications.

Notre objectif suivant était de proposer au développeur une utilisation homogène d'un ensemble d'objets hétérogènes : il nous a fallu pour cela une longue période de veille technologique et de réflexion afin d'aboutir à une architecture flexible et adaptée.

Cela nous a également permis de répondre au but fixé de simplifier l'ajout d'un nouvel objet connecté à un ensemble existant.

Notre solution permet au développeur :
— D'ajouter aisément un nouvel objet, on rentrant simplement un certain nombre d'informations basiques le concernant ;
— Quel que soit le mode de communication de l'objet, le développeur n'a plus à s'en soucier et doit juste utiliser le proxy qui résulte de notre la génération de notre système ;
— Son application peut ainsi par la suite utiliser la récupération de données provenant d'un ensemble complexe d'objets connectés.

Concrètement, on imagine un développeur voulant ajouter son objet, en l'occurrence la balance connectée Smart Body Analyzer[10] de Withings, à un ensemble d'objets connectés préexistant, contenant entre autres la Wii Balance Board, soit un objet du même type et donc possédant des fonctionnalités communes.

Ce qui diffère concrètement, et qui ne permet pas de mettre en commun ces deux objets, est la technologie de communications qu'ils utilisent. Alors que les données de la Wii Balance Board[16] sont accessibles via une communication Bluetooth, celles de la Smart Body Analyzer le sont via des appels à des services Web de l'API Withings.

Ainsi, pour ajouter cette dernière, le développeur devra simplement indiquer son mode de communication de l'objet – ici ≪ Web Service ≫, avec utilisation du protocole d'authentification OAuth[15] -, puis ajouter les adresses et paramètres de ces services mis à disposition par le constructeur.

Il pourra ensuite effectuer un mapping entre la fonction "BBMassEvent.getTotalMass()" et "WithingsBalance.getWeight()". Ainsi, l'application fournie par le développeur, pourra baser sa récupération du poids de l'utilisateur lambda sur la base de données Withings. En revanche, dès lors qu'un évènement sera détecté en provenance de la Wii Balance Board, cette nouvelle donnée viendra s'ajouter en tête de la pile des valeurs du poids. Celle-ci étant donc plus récente que le dernier relevé de la balance Withings, il est plus cohérent que l'application tienne compte de cette dernière.

Ainsi, nous voyons à quel point le développeur n'a finalement pas grand-chose à faire lorsqu'il souhaite ajouter un nouvel objet, hormis ajouter quelques informations.

De plus, grâce à notre plateforme, le développeur pourra gérer la multiplicité d'objets de même famille, voire même la substitution. En effet, il sera désormais facile de programmer une récolte d'informations, automatisées (par notifications d'évènements) ou pas, auprès de tel ou tel objet en fonction par exemple de sa présence ou son absence (sous tension ou pas).

3. Description du travail fourni

3.1 Travail préliminaire

L'un des mots clés de notre projet de fin d'études, d'ailleurs contenu dans son titre, est "hétérogène". En effet, nous avons dû traiter de nombreux objets connectés, de différentes marques, utilisant différents modes de communication, différents protocoles d'échanges, à l'image de la diversité du marché actuel.

Afin donc de maîtriser cette hétérogénéité, il nous a fallu faire une étude, en amont, des technologies généralement utilisées ainsi que des objets mis à notre disposition.

Pour ces derniers, nous avons tout d'abord cherché, au travers des supports proposés, le moyen de communication utilisé, celui-ci représentant le plus gros du challenge. Concernant les objets Withings par exemple, le Pulse O2[11] utilise exclusivement du Bluetooth alors que la balance communique par Bluetooth et Wi-Fi. En revanche, l'API publique du constructeur stocke les résultats sur son propre serveur ; ainsi, le Bluetooth n'est utilisé que pour envoyer les données vers le smartphone, servant d'intermédiaire pour le transit d'informations vers le serveur en question. Aucune documentation n'est cependant disponible afin de reproduire la communication Bluetooth/périphérique. Le pèse-personne Smart Body Analyzer envoie lui directement les données en Wi-Fi.

Le Nabaztag[12] a quant à lui nécessité des recherches supplémentaires, puisque c'est le seul objet mis à notre disposition qui propose non seulement la récupération d'informations, mais également une écriture. La compréhension de l'objet a été plus longue que prévu, puisque, le Nabaztag n'étant plus commercialisé, les serveurs officiels ne sont plus actifs ; heureusement il existe des serveurs alternatifs. Leurs fonctionnements et fonctionnalités proposées sont assez similaire, la différence se faisant plutôt sur le contenu des messages échangés. Par exemple il est possible de créer une séquence d'actions pour le Nabaztag (par exemple déplacer son oreille droite d'un certain angle puis faire clignoter une de ses LEDs). Pour un serveur, par exemple, la partie de message pour bouger l'oreille droite sera sous la forme suivante : "0507", 05 correspondant à l'oreille droite et 07 à sa position (une valeur pouvant aller de 0 à 16) alors que pour un autre serveur alternatif pour faire bouger l'oreille droite dans le sens horaire d'un angle de 20 degrés il faudra envoyer un morceau du message sous cette forme : "0,motor,0,20,0,1". Quel que soit le serveur alternatif utilisé, il est généralement permis de faire agir son Nabaztag, réagir à des tags ou à d'autres Nabaztags.

Une fois cette étude effectuée, il nous a fallu dégrossir le problème afin de faciliter la construction de la solution mais également son utilisation à venir par les développeurs. En effet, la prise en main doit être la plus instinctive possible et un minimum de code doit être regénéré inutilement.

Nous avons donc axé la logique de notre système sur un principe de regroupement en "familles". Cependant, la constitution de ces dernières nous a été bien difficile, en raison du grand nombre d'objets à classer, leurs nombreuses caractéristiques, se recouvrant souvent, et parmi lesquelles il est très facile de trouver plusieurs regroupements. La difficulté résidait principalement dans le fait que ces derniers soient très subjectifs ; en effet, aucune solution n'était bonne ou mauvaise, il nous a fallu faire des choix qui nous semblaient plus judicieux que d'autres, pour la scalabilité par exemple.

Nous avons ainsi opté pour les regroupements qui nous semblaient les plus intuitifs, par exemple :

- Par marque : on imagine que le développeur d'un constructeur va ajouter les blocs pour l'ensemble des objets de la marque en question. Il est donc judicieux de mettre en place une famille lui permettant de réutiliser un bloc commun à l'ensemble de ces objets. Nous avons pris conscience de cela grâce aux objets Withings mis à notre disposition, puisque ces derniers utilisent principalement un service web REST[22] et possèdent par exemple tous une étape d'authentification sécuritaire (API OAuth) ;
- Par fonctionnalités : on imagine le cas d'un environnement connecté possédant plusieurs objets de même type, du moins ayant en commun des fonctionnalités identiques. Pour notre projet, on peut prendre l'exemple des deux balances mis à notre disposition, la balance connectée Withings et la Wii Balance Board. On aimerait ainsi pouvoir récolter les données provenant de ces deux objets mais en stipulant que les fonctions de récupération de poids sont bel et bien les mêmes sur les deux objets.

D'autres regroupements sont bien évidemment possibles et sont explicités plus amplement dans le document rendu pour le jalon #1 de ce projet.

3.2 Conception du framework

3.2.1 Conception générale

Une fois l'étude préalable effectuée, soit l'ensemble des technologies et des objets appréhendés, nous avons pu entamer la phase de conception du framework.

Pour cela, nous avons tout d'abord dégrossi le problème en essayant de schématiser un scénario complet, et ainsi identifier les

étapes essentielles, allant de la saisie des informations de l'objet jusqu'à la récupération du proxy généré.

La compréhension du système dans son ensemble a été longue et a nécessité plusieurs réunions avec nos encadrants afin d'apporter un certain nombre de précisions.

Notre premier obstacle a été rencontré lors de l'élaboration du proxy à générer. Il nous aura entre autre fallu effectuer une étude des technologies utilisées généralement pour l'utilisation des objets connectés. Nous sommes rapidement arrivés à la conclusion que l'UPnP[13] était le standard le plus courant. Il présentait en plus l'avantage de proposer des fonctionnalités nécessaires, voire essentielles, à notre framework, à savoir détecter la présence ou l'absence de l'objet dans l'environnement. Aussi, il gère la notion d'événement, requise pour ce projet.

Après des recherches plus approfondies, nous avons découvert le protocole DPWS[14] : il s'agit là d'une version améliorée de l'UPnP, puisque possédant une couche de sécurité supplémentaire et se reposant uniquement sur des standards. Cependant, le protocole est de ce fait plus pénible à prendre en main et les solutions d'implémentation disponibles sont beaucoup moins nombreuses que pour UPnP. Ce choix aurait donc demandé un investissement supplémentaire pour un résultat qui aurait pu être obtenu à moindre coût. De plus notre PFE ayant pour but d'établir une preuve de concept, nous avons jugé bon de nous passer de cet aspect sécuritaire supplémentaire et gagner du temps en appréhension de technologie afin de nous concentrer sur l'essentiel.

Notre deuxième obstacle a été de prévoir un framework proposant des fonctionnalités bien trop poussées ; en effet, nous avions eu une optique d'automatisation trop complexe. Ainsi, notre première conception demandant à l'utilisateur de ne saisir que le strict minimum d'informations, et le cœur du système effectuant le reste du travail. L'implémentation d'une telle solution aurait demandé une élaboration très poussée et un nombre considérable d'heures.

Nous sommes finalement arrivés au schéma général suivant (pour plus de précisions, cf. document rendu pour le jalon #1) :

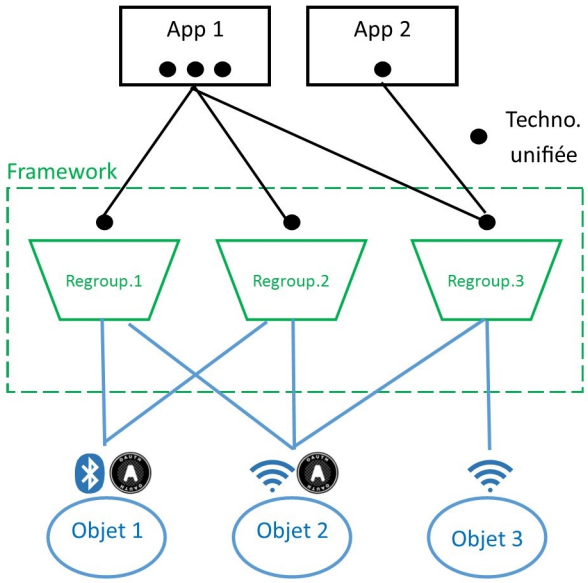

FIGURE 1. Schéma représentant la solution à obtenir

Ce schéma permet de voir de façon claire à quel niveau les différents développeurs vont pouvoir intervenir :

- Les développeurs d'objet connectés seront les acteurs de la couche « Middleware », une couche finalement assez bas niveau, puisque ce sont eux qui seront en charge d'ajouter les nouveaux blocs technologiques permettant d'abstraire les contraintes matérielles et technologiques de leur objet ;
- Les développeurs d'application quant à eux se situent au niveau supérieur, celui contenant l'ensemble des applications. On voit bien que quel que soit les objets qui alimenteront le contenu de celles-ci, le développeur n'aura plus qu'à se soucier de connaitre la technologie unifiée, et de rien d'autres.

L'étape suivante a été d'approfondir l'ensemble des entrées et sorties du système : quelles informations doit saisir le développeur ? Quels services doit proposer le proxy ?

3.2.2 Approfondissement

Nous avons adopté une méthodologie inversée, puisque nous sommes partis de ce que nous devions obtenir, afin de pouvoir définir avec plus de précisions et de certitude les entrées requises pour le bon fonctionnement de la génération du proxy.

Le but principal du projet étant d'unifier la technologie de communication, afin de produire un proxy qui n'en utilise qu'une seule, c'est évidemment une des informations requises lors de la saisie par le développeur utilisant le framework. Ce dernier devra également renseigner le mode de communication utilisée. Les autres informations, telles que la marque de l'objet, son identifiant, sa description, etc, seront des données secondaires, utilisées principalement pour faciliter la recherche plus tard. En effet, dans le cas où un nouveau développeur ajouterait un objet, il peut être utile pour lui de pouvoir chercher les travaux déjà effectués par ses collègues afin d'en réutiliser une partie s'il s'agit de technologie(s) similaire(s).

Dans cette perspective d'unification technologique, après avoir opté pour un protocole de sortie UPnP, il nous a également fallu réfléchir au langage du système, soit celui en charge de la génération du proxy. Nous avons choisi d'utiliser le langage Java, car le mieux maîtrisé par l'ensemble des membres du groupe tous les outils nécessaires au développement du projet, au travers de diverses librairies. De plus, le but de notre framework étant de proposer une solution générique, utilisable par le plus grand nombre, il semble judicieux de faire le choix d'un des langages les plus utilisés et les mieux documentés à ce jour.

Nous avons à ce stade-là les informations à recueillir auprès du développeur, le langage de programmation du cœur du système ainsi que la technologie utilisée par le proxy à générer, reste à savoir : comment effectuer cette génération ?

3.2.3 Génération du proxy

Une fois encore, le temps de réflexion et de conception a été très conséquent : il s'agit là d'un exercice auquel nous avons rarement été confronté, encore moins avec un tel degré de difficulté. En effet, jusqu'ici, sur l'ensemble des projets que nous avons réalisés nous avons été plus ou moins guidés quant aux choix technologiques ou conceptuels, mais jamais un développement ne nous a confrontés à tant d'obstacles.

Nous avons donc tout d'abord réfléchi à la forme exacte que devait avoir notre proxy : doit-il être un exécutable ? Un ensemble de fichiers modifiables ? Puis sur sa génération propre : créons-nous une entité "Proxy" générique prenant un fichier de configuration en paramètre ? Ou est-ce que chaque proxy généré est une entité bien spécifique ?

Dans le cas par exemple de la possible généricité de l'entité "Proxy", nous avons rapidement songé à mettre en place de la réflexivité Java afin de créer un squelette, piste longuement creusée jusqu'à réaliser que l'ajout dynamique de fonctions était difficilement possible. Cette solution a alors été définitivement écartée car non adaptée à notre problème.

Afin de palier à ce désagrément, nous nous sommes par la suite redirigés vers une résolution par le biais de template qui pourra générer l'interface UPnP (Cling), à partir d'une représentation objet du fichier de mapping.

L'utilisation de templates offre l'avantage d'être plus propre que du printout en étant presque aussi rapide à mettre en place. Le choix s'est porté sur la bibliothèque Apache Velocity[17] puisqu'il s'agit d'une bibliothèque reconnue et modulaire. Nous avons ainsi défini plusieurs templates : serveur UPnP, service, et méthode. Le serveur utilise les services, et les services définissent une ou plusieurs méthodes. Ces templates permettent de générer le squelette des classes et les signatures de méthodes avec les annotations Cling. Le corps des méthodes quant à lui est rempli par du "print".

En bout de chaîne, un projet compilable avec Maven est généré.

Afin cependant de ne pas seulement nous contenter d'une solution trop "facile", ou de ne regarder exclusivement dans les compétences que nous avons acquises, nous avons fait une recherche dans le but de trouver si d'autres solutions pouvait exister. Or, certains membres du groupe ayant suivi le cours d'IDM avaient étudié les DSL[18]. Cependant ces derniers font intervenir des notions complexes requérant beaucoup plus de temps et d'investissement pour être appréhendées. Une fois encore notre PFE ne constitue qu'une preuve de concept et un tel engagement ne serait pas alors pas justifié ; surtout au vu de la mise en place de techniques aussi performantes. La piste des DSL représente cependant une belle perspective d'amélioration ou d'alternative.

3.2.4 Interactions développeur/système

Finalement, les interactions avec le développeur se feront par le biais d'un portail basique : il s'agit là de notre démonstrateur, décrit en détails par la suite.

Celui-ci aura pour rôle de réceptionner les données basiques décrivant l'objet ainsi que de permettre à l'utilisateur d'effectuer son mapping des fonctions.

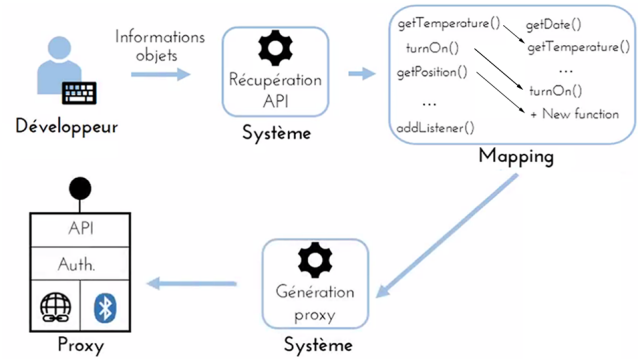

FIGURE 2. Schéma représentant la solution à obtenir

3.2.5 Serveur embarqué

S'est ensuite posé la question du serveur embarqué. Nous travaillons sur des objets connectés, donc de petits systèmes, il faut donc pouvoir mettre en place un serveur léger. La preuve de concept ne nécessite pas de création concrète, c'est-à-dire que nous considérons l'hypothèse où notre proxy sera déployé sur une plateforme possédant toutes les caractéristiques (logicielles, techniques, etc) nécessaires à son fonctionnement.

Une nouvelle et longue veille technologique a une fois encore due être effectuée afin de trouver une solution adaptée. Afin de développer un serveur léger, nous avons envisagé d'utiliser l'outil Node.js[19]. Cependant, après une étude et des tests approfondis, il est apparu qu'il n'était que partiellement compatible avec le protocole UPnP, en grande partie car il repose sur du JavaScript, langage de haut niveau.

3.3 Implémentation du framework

3.3.1 Utilisation de modules

L'intérêt pour le développeur d'utiliser notre framework réside également dans le fait de ne pas avoir à redévelopper chaque couche du protocole de communication. Nous avons donc fait le choix de développer des modules génériques, qui seront embarqués dans l'archive générée. Ceux-ci seront alors appelés par le serveur UPnP avec les bons paramètres, résumant dans de nombreux cas cet appel à une simple ligne.

Actuellement, deux modules permettant une communication via services web sont développés. Le premier permet de se connecter à un service web standard, alors que le second offre l'utilisation de la surcouche de sécurité OAuth. Pour utiliser ce dernier, le développeur de l'application cliente devra tout de même fournir lui-même les quatre informations relatives au fournisseur de service (API key, API secret, access token et secret token).

Lorsqu'un module générique n'est pas disponible, le développeur peut fournir lui-même son propre module à intégrer dans l'archive.

3.3.2 Utilisation de templates

Comme précisé précédemment, le code est généré en utilisant le moteur de templates Apache Velocity. Trois templates sont définis pour le serveur UPnP (serveur, service, méthode), et ils sont remplis à l'aide d'un objet de type dispositif UPnP. Nous avons cherché à séparer la logique UPnP de la génération via moteur de templates. Pour ce faire, nous manipulons en premier lieu un objet de type dispositif UPnP, et nous adaptons nos templates à celui-ci.

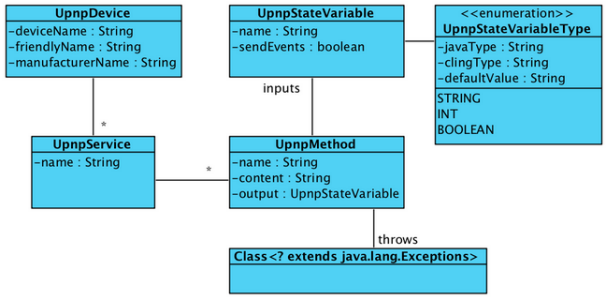

FIGURE 3. Diagramme de classes représentant un dispositif UPnP

3.3.3 Déroulement

De l'entrée du fichier texte à la génération de l'archive, trois étapes principales se déroulent.

Lors de la récupération du fichier d'entrée au format JSON, celui-ci est parsé pour être transformé en POJO qui sera plus aisément manipulable. Pour cela, un singleton a été mis en place, avec gestion de l'héritage au sein du fichier JSON. Par exemple,

lorsqu'il faut spécifier les informations qui lient méthode générique et méthode de l'objet, il est évident que celles-ci seront différentes en fonction qu'il s'agisse d'un service web ou d'une communication en Bluetooth. Ainsi, l'utilisateur ne devra spécifier que ce qui est utile pour l'objet concerné.

Ensuite, il faut créer un objet correspondant à un dispositif UPnP et extraire les informations requises. L'idée est d'extraire deux informations du POJO créé à partir du fichier passé en entrée : premièrement un objet correspondant à un dispositif UPnP et deuxièmement la liste des éléments requis (JAR local, dépendance Maven...).

La dernière étape consiste à créer l'archive Maven correspondante à partir des informations extraites lors de l'étape précédente. La création se fait en deux étapes : ajout du code généré puis ajout des dépendances. Comme précisé précédemment, le code est généré à l'aide d'un système de templates. Dans cette version naïve, seuls le serveur d'un dispositif UPnP et son unique service correspondant sont créés. Ensuite, les dépendances (i.e. module prédéfini, JAR...) sont ajoutées. Lorsqu'une dépendance au format JAR est ajoutée, un repository local Maven est créé dans le projet.

3.4 Démonstrateur

3.4.1 Conception

Tout d'abord, il est important de préciser que le framework nous a demandé bien plus de temps que prévu, en raison d'une étude préliminaire et de veilles technologiques très longues. C'est pour cela que les ressources ont été réparties différemment ; ainsi, la part du démonstrateur a diminué comparé aux prévisions faites pour le Gantt initial. En effet, il ne s'agit là que d'une partie servant à appuyer notre démonstration, à mettre en avant l'utilisation de notre framework et du proxy généré au travers d'un exemple concret, plus présentable qu'un résultat en console, et non pas une part primordiale de notre PFE. Cela ne nous a donc été en aucun cas préjudiciable de réduire cette tâche.

Afin de concevoir ce démonstrateur, il a fallu tout d'abord choisir les objets qui allaient être utilisés. Ce choix s'est fait selon divers critères, tels que la facilité d'utilisation et d'intégration au framework, l'appartenance à une "famille" précise, la pertinence des données et fonctionnalités utilisées, etc. Ainsi, le but de l'application étant de démontrer l'utilisation et la faisabilité d'un tel framework, nous avons cherché à exploiter différents moyens de communication avec des objets pouvant être regroupés dans une même catégorie d'utilisation. Au vu des objets à notre disposition, notre choix s'est naturellement porté sur les deux balances, à savoir le Smart Body Analyser de Withings et la Wii Balance Board. A partir de ce choix, l'idée était donc de déterminer comment exploiter les données de ces deux objets de façon pertinente et intéressante pour l'utilisateur. Après une étude de leurs caractéristiques, une première maquette a été établie :

Ainsi, les deux balances permettant d'obtenir la masse de l'utilisateur, nous avons décidé de recouper ces informations sur une même vue afin de les comparer et d'en établir une moyenne. La balance Withings Smart Body Analyzer permet quant à elle d'obtenir des informations supplémentaires, telles que la masse grasse ou le rythme cardiaque, qui seront affichées dans des vues séparées.

Cette maquette a ensuite été améliorée afin d'aboutir à une interface implémentée plus évoluée, plus pertinente et surtout plus concrète. Devant le peu de temps disponible pour réaliser ce démonstrateur et afin d'obtenir un résultat final convaincant et abouti, nous avons choisi de développer une application Web, pouvant être visualisée sur plusieurs dispositifs (Ordinateur, SmartPhone, tablette, ...).

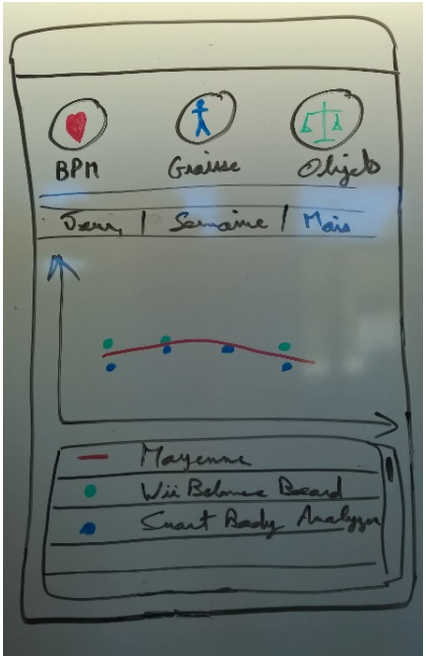

FIGURE 4. Première version de la maquette du démonstrateur

3.4.2 Implémentation

Le démonstrateur est constitué de deux composants majeurs.

Le premier est une application Java J2EE[20] composée d'un client UPnP Cling et d'un système de persistance de données, utilisant le framework Spring[21]. Cette application expose également une API Rest permettant à la vue d'accéder aux données.

Le deuxième composant est un client Web, développé en HTML5, CSS3 et JavaScript (avec jQuery[23]). Celui-ci utilise la librairie AmCharts.js[24] permettant de réaliser des graphiques. Notre choix s'est porté sur cette librairie pour sa simplicité d'utilisation, son design attrayant et son aspect Responsive Web Design.

Cette décomposition s'est faite afin de maintenir une architecture MVC[26] permettant le développement de différentes vues autour d'un même noyau fonctionnel, apportant ainsi une plus grande flexibilité, un plus grand découplage et donc une meilleure maintenabilité.

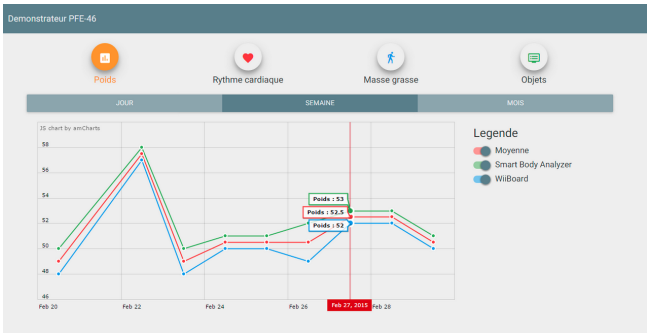

FIGURE 5. Interface final, site web – Evolution du poids

La figure ci-dessus présente l'interface finale. Sur cette vue, une zone supérieure contient quatre boutons permettant respectivement l'affichage de l'évolution du poids, du rythme cardiaque ou de la

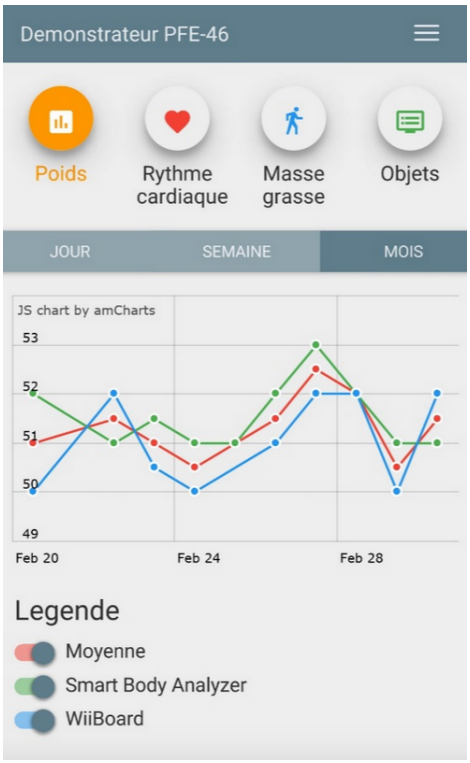

FIGURE 6. Interface finale Responsive, version mobile

masse grasse en fonction d'une période de temps définie, ou encore des informations sur les objets desquels proviennent ces données (variable pour chaque objet), telles que le statut – ON, OFF -, le niveau de la batterie, etc. Celles-ci sont alors affichées dans la zone principale sur une courbe ou dans des panels.

Ce démonstrateur présente donc une exploitation de données provenant de plusieurs objets, utilisant un mode de communication différent, et récupérées directement via le proxy grâce à une technologie de communication unique, à savoir UPnP.

4. Validation

Afin de mettre en avant le travail effectué de la meilleure des façons qu'il soit à l'aide des objets connectés mis à notre disposition, nous avons opté pour une démonstration basée sur les deux balances : la balance connectée Withings et la Wii Balance Board. Cela nous permet de montrer la puissance de l'outil que nous proposons pour :

- Des modes de communication différents : la balance connectée Withings communique par le biais du Wi-Fi avec un service web contrairement à la Balance Board qui utilise le Bluetooth ;
- Des marques différentes ;
- Des objets de mêmes fonctions : cela va nous mettre en avant l'étape de mapping des fonctions ainsi que le travail qui est fait par la suite par le proxy lorsqu'il trouve deux objets concurrents.

5. Positionnement vis-à-vis de l'état de l'art

A ce jour, l'ensemble des constructeurs d'objets connectés développent des applications propres à ces derniers. Dans le cas où un même fabriquant commercialise plusieurs produits, un effort d'uniformisation au sein de ceux-ci est souvent fourni.

Actuellement aucun framework ayant pour rôle de faciliter l'intégration d'objets connectés hétérogènes n'existe, seule la firme Google s'est officiellement positionnée sur ce manque[27]. Cependant, certaines plateformes permettant de mettre en commun l'utilisation de différents services Web fleurissent. Mais celles-ci se situent à un niveau plus élevé, plus applicatif et non matériel. En effet, leur but n'est pas d'unifier la technologie de communication mais de croiser le résultat de services choisis afin de définir des comportements spécifiques.

Comme explicité plus largement dans le document "Description of Work", contrairement à l'Internet qui possède un standard, à savoir le protocole HTTP, les objets connectés ne connaissent pas encore de réel protocole universel. Cependant, celui qui s'en rapproche le plus est l'UPnP, c'est pour cela qu'il s'agit du choix que nous avons fait pour le protocole de génération de proxy.

6. Élargissement et perspectives

Nous sommes donc parvenus à fournir un framework permettant l'intégration facile de certains objets mis à notre disposition. Certains d'entre eux n'étant pas assez documentés ou difficiles d'accès (aucune API connue), nous avons dû les écarter et nous concentrer sur les autres.

Toutefois, nous rappelons qu'il s'agit là d'une simple preuve de concept. Ce travail vient s'inscrire dans la recherche scientifique de nos encadrants, Messieurs Gaëtan Rey et Stéphane Lavirotte ; nous fournissons ainsi une étude approfondie du travail qui devrait être effectué afin de mettre en place une plateforme facilitant le travail des développeurs d'objets connectés ainsi que la récupération de données dans des environnements complexes.

Comme décrit précédemment dans ce rapport, nous avons opté pour une solution de génération du proxy par le biais d'un framework. Cependant, il ne s'agit peut-être pas de la solution optimale et utiliser un DSL aurait sûrement été plus adapté. Le temps imparti ne nous permettant pas une mise en place aussi poussée, nous ne l'avons pas fait mais dans le cas de la reprise du projet par un autre groupe, ce serait l'une des perspectives à explorer et représenterait une valeur ajoutée au PFE actuel.

Autre élargissement qu'il serait possible de faire : l'aspect sécuritaire des échanges. Ce côté-là du projet a également été évoqué auparavant, il aurait fallu, dans un premier temps, opter pour le protocole DPWS et non pas UPnP, afin d'ajouter une couche de sécurité. Cependant nous n'en avions pas le temps, de plus, la sécurité dans les environnements connectés est un des problèmes majeurs actuels et peu de solutions et d'outils existent de nos jours ; c'est pour cela que nous n'avons pas persisté dans cette voie mais qui mériterait attention, en particulier pour les échanges de clés de sécurité (clés d'API, etc) entre l'application et le proxy.

En conclusion, c'est une longue période d'étude et de recherche qui nous a permis d'aboutir à un résultat final qui constitue une implémentation possible - d'autres ayant été explorées et détaillées précédemment -. Ainsi, cette preuve de concept vient confirmer la faisabilité d'un tel framework mais c'est cette étude faite en amont et durant toute la durée du projet qui est à poursuivre et à mettre en perspective avec l'état de l'art à un instant t. Au vu de l'état embryonnaire de l'Internet of Things[30], ce projet présente une belle avancée vers le décloisonnement des objets connectés. Cependant, afin de ne pas être dépassé, celui-ci doit s'adapter aux évolutions de ce domaine, tâche qui sera effectuée par nos successeurs.

Références

Documents produits

[1] A. Casanova, A. Llorens, V. Sallé et M.-C. Turchini. *D1.1 - Description of Work*, 2014

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

[3] A. Casanova, A. Llorens, V. Sallé et M.-C. Turchini. *D2 - Conception du framework et du démonstrateur*, 2014

[4] Démonstrateur URL `http://users.polytech.unice.fr/~acasanov/pfe46`

[5] Diaporama de la soutenance URL `http://users.polytech.unice.fr/~salle/pfe/PFE-46_soutenance.pdf`

[6] Vidéo finale URL `http://youtu.be/qJO4amjegAA`

Encadrants

[7] Stéphane Lavirotte URL `http://stephane.lavirotte.com/`

[8] Gaëtan Rey URL `http://users.polytech.unice.fr/~rey/`

Objets connectés utilisés

[9] Withings URL `http://www.withings.com/fr/`

[10] Withings Smart Body Analyzer URL `http://www.withings.com/fr/smart-body-analyzer.html`

[11] Withings Pulse O2 URL `http://www.withings.com/fr/withings-pulse.html`

[12] Nabaztag URL `www.karotz.com`

Technologies

[13] UPnP URL `http://fr.wikipedia.org/wiki/Universal_Plug_and_Play`

[14] DPWS URL `http://en.wikipedia.org/wiki/Devices_Profile_for_Web_Services`

[15] API OAuth URL `http://oauth.net/`

[16] API pour l'utilisation de la Wii Balance Board URL `https://github.com/micromu/WiiRemoteJ`

[17] Apache Velocity URL `http://velocity.apache.org/`

[18] DSL (définition) URL `http://fr.wikipedia.org/wiki/Langage_d%C3%A9di%C3%A9`

[19] NodeJS URL `http://nodejs.org/`

[20] Java J2EE URL `http://fr.wikipedia.org/wiki/Java_EE`

[21] Spring framework URL `http://projects.spring.io/spring-framework`

[22] API Rest URL `http://fr.wikipedia.org/wiki/Representational_State_Transfer`

[23] jQuery URL `http://jquery.com/`

[24] AmCharts URL `http://www.amcharts.com/`

[25] Site web adaptatif URL `http://fr.wikipedia.org/wiki/Site_web_adaptatif`

[26] MVC (Modèle-Vue-Contrôleur) URL `http://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur`

Etat de l'art

[27] Physical Web URL `https://github.com/google/physical-web/tree/master/documentation`

[28] Busit URL `https://www.busit.com/`

[29] IFTTT URL `https://ifttt.com/`

[30] IoT (Internet of Things) URL `http://fr.wikipedia.org/wiki/Internet_des_objets`