Projet De Master En Sciences informatiques

ThingBook : le réseau social des objets connectés

RICHARD VALENTIN

2014-2015

Sous la direction du professeur :
M. Jean-Henry Morin

Résumé

L'internet des objets est actuellement en pleine expansion, avec une prévision de 26 milliards de ces objets en 2020 selon Gartner. Ce marché prometteur souffre toutefois d'un manque flagrant d'interopérabilité entre les différents produits de marques différentes, sans oublier que la communication entre objets s'arrête bien souvent au niveau local ou au niveau de la sphère personnelle. Le potentiel des objets connectés est de ce fait largement sous-utilisé. D'un autre côté, avec près de 1,4 milliard d'utilisateurs pour Facebook, nous pouvons dire que les réseaux sociaux sont omniprésents. Destinés majoritairement à des êtres humains, il existe aussi des réseaux sociaux plus atypiques pour des animaux de compagnie (chiens ou chats par exemple). Alors pourquoi ne pas aussi en concevoir un pour les objets connectés ?

Nous proposons donc de créer un réseau social spécialement dédié à ces objets, nommé ThingBook, où les objets seraient plus autonomes, capables de tisser des liens virtuels entre eux et de partager des données librement, ou du moins à une échelle plus large que la simple sphère personnelle. À travers ce document, nous avons donc imaginé, conçu, puis évalué le premier réseau social pour les objets connectés.

Table des matières

Liste des figures

Liste des acronymes

- AJAX - Asynchronous JavaScript And XML
- API - Application Programming Interface
- BDD – Bases De Données
- BYOD - Bring Your Own Device
- CSS - Cascading Style Sheets
- EVSI - Espérance de Vie Sans Incapacité
- HTML - Hypertext Markup Language
- HTTP - Hypertext Transfer Protocol
- HTTPS - HyperText Transfer Protocol Secure
- IoT - Internet of Things
- JSON - JavaScript Object Notation
- MAC - Media Access Control
- NoSQL - Not Only SQL
- PHP - PHP : Hypertext Preprocessor
- REST - Representational State Transfer
- SQL - Structured Query Language
- URL - Uniform Resource Locator
- XML - Extensible Markup Language

1 - Introduction

L'internet des objets (IoT en anglais) est en pleine croissance. Plusieurs milliards d'objets sont déjà connectés à internet et ce n'est qu'un commencement. Bien qu'il existe déjà des services qui rendent les interactions utilisateurs-machines plus accessibles et permettent un stockage ainsi qu'une gestion plus facile des données, il n'existe pas encore de plateformes qui authentifient ces objets, entreposent leurs données, les partagent (si cela est admis), et surtout qui créent des liens entre les différents objets, comme nous le faisons avec notre entourage.

À travers ce document, nous nous efforcerons de combler ce vide en concevant le premier réseau social destiné aux objets connectés.

Nous pourrions par exemple imaginer, dans le cadre d'un cabinet de photographie, un appareil photo en relation avec une imprimante. Tous deux connectés à internet et se servant de ce réseau social, l'imprimante s'occuperait d'imprimer immédiatement chaque photo prise par le photographe, quelle que soit la distance les séparant. Bien entendu, il est possible de créer tout un tas de services spécifiques pour ce genre d'application, mais il est plus pertinent d'avoir un seul service capable de gérer des situations très variées d'interactions entre machines.

Nous commencerons par définir le champ de recherche, tout en examinant les solutions déjà existantes et leurs limites. Nous pourrons alors en tirer une problématique claire et précise. S'en suivront quelques exemples de scénarios qui définiront les exigences et les rôles de chaque partie prenante (humains, machines et programmes). En nous appuyant sur ces besoins, nous nous efforcerons d'y répondre en concevant une architecture pour ce réseau social, puis nous la concrétiserons au travers d'un prototype fonctionnel. Celui-ci sera ensuite éprouvé par plusieurs applications développées sur des objets connectés, pour confirmer son efficacité. Finalement, nous conclurons cette recherche en résumant le travail accompli, et en proposant quelques pistes d'améliorations futures.

2 - État de l'art

2.1 - Définition de l'IoT

L'internet des objets consiste en l'interconnexion d'objets physiques identifiables et capables de communiquer entre eux ou avec n'importe quelle machine du réseau (à l'aide ou non d'un dispositif d'identification), permettant ainsi de traiter des données provenant non plus seulement du monde virtuel, mais aussi du monde réel. Ces objets peuvent être de nature très différente : livres, vêtements, montres ou encore nourriture, en effet l'internet des objets se veut capable de toucher à l'ensemble des objets (ou presque) qui nous entourent, les connectant ainsi à notre monde virtuel. Ils peuvent avoir plusieurs rôles conjugués : fournir des données sur l'environnement qui les entoure (senseurs) ou agir sur celui-ci (actuateurs), afficher ces données, transmettre des informations et en recevoir ou encore prendre des décisions de manière autonome. Finalement, ils doivent être identifiables de manière unique au sein d'un même réseau.

2.2 - Histoire de l'IoT

Mark Weiser est un des premiers à avoir eu l'idée de l'internet des objets sans utiliser directement ce terme, mais en discutant de son travail effectué sur l'Ubiquitous Computing au sein de la société PARC, et ce dès 1987 [A01]. Il notera alors que la productivité dans l'entreprise a augmenté grâce à la facilitation des taches courantes, apportée par l'Ubiquitous Computing, mais que cela apporte aussi quelques questions d'ordre éthique : « *Big Brother Comes to the Office* ». Il continuera donc dans cette voie et rédigera un peu plus tard en 1991, un article beaucoup plus détaillé concernant ce sujet [A02] dans lequel il propose sa propre vision de l'avenir : l'informatique sera intégrée dans toute sorte d'objets connectés à travers un même réseau : liveboard (tablettes tactiles ou grands tableaux interactifs), badges, interrupteurs, thermostats ou encore radios, avec comme conséquence au fil du temps de faire disparaitre l'informatique, alors omniprésente, de la conscience des utilisateurs.

C'est avec Kevin Ashton en 1999 qu'apparait pour la première fois le terme Internet of Things lors d'une présentation à Procter & Gamble [A03]. Sa vision, à cette époque, est que jusqu'alors la majorité des informations présentes sur internet est d'origine humaine. Or l'humain n'est pas spécialement adapté à la conversion de données du monde réel vers le monde virtuel, là où les machines excellent. D'autre part, les êtres humains vivent dans un monde physique où toute notre société, notre économie et nos besoins sont basés sur la notion de matière, d'où une nécessité de lier virtuel et réel. Il préconise dont de laisser les ordinateurs s'occuper d'une partie monde virtuel grâce à leur conscience du monde réel, et éventuellement d'optimiser ce dernier en effectuant des tâches automatiques, par exemple en réduisant les coûts et les pertes d'énergie.

De nos jours, nous n'en sommes pas encore à ce qu'avaient préconisé ces deux hommes, mais nous nous en rapprochons de plus en plus. En effet, le phénomène de l'IoT est en plein boom : le nombre d'objets connectés dépasse actuellement les 9 milliards et offre de très belles perspectives d'avenir. Selon Gartner le nombre de ces objets passerait à 26 milliards en 2020, générant ainsi environ 1900 milliards de dollars en valeur ajoutée [L01][L02].

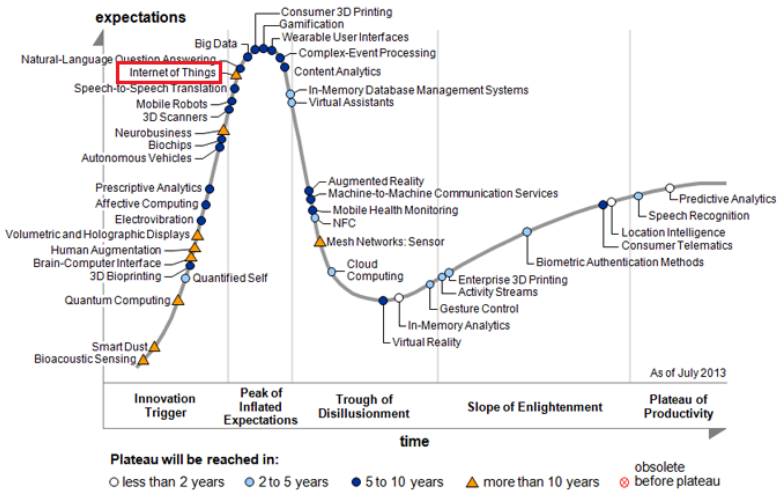

Figure 1 - Gartner's Hype Cycle concernant les nouvelles technologies

D'autre part, les utilisateurs, entrepreneurs et chercheurs s'intéressent de plus en plus à l'IoT : le nombre d'objets interconnectés mis sur le marché ne cesse d'augmenter, de nouveaux produits voient le jour et l'engouement des utilisateurs est en pleine croissance.

Figure 2 - Tendances de recherches Google pour le terme : Internet of Things

2.3 - Domaines majeurs de l'IoT

2.3.1 - Smart home

La maison intelligente est un habitat possédant de nombreux systèmes d'automatisation qui permettent aux habitants d'avoir un monitoring et un contrôle sur celle-ci. Elle a aussi pour rôle d'automatiser les tâches courantes et de s'auto-optimiser, par exemple en régulant sa consommation d'électricité, de chauffage ou encore d'eau [A04]. Or c'est un point très important pour ce 21ème siècle où l'écologie est au cœur des débats concernant l'avenir : le simple fait de pouvoir surveiller sa consommation en énergie permet de la réduire de 5 à 15%. De plus c'est un marché très prometteur qui pourrait représenter 4 milliards de dollars en 2016 [A05].

2.3.2 - Vêtements connectés

Aussi connus sous le nom de wearables, ils ont pour objectif d'être au plus près de nous en se portant directement sur le corps et ne se limitent pas aux textiles : montres, lunettes ou encore chaussures font aussi partie de ce domaine.

Ces objets sont actuellement destinés majoritairement aux sportifs. Ils recueillent des données vitales : température, pouls, oxygénation... mais aussi des données d'activité : vitesse, calories consommées, position spatiale... Et pourtant ce n'est pas leur seule fonctionnalité, certains couturiers s'en servent comme vêtements capables de changer de couleur [L03]. Bien plus connues encore, les Google Glass ont pour objectif de connecter l'utilisateur en permanence et de lui offrir toute une gamme de services, sans que celui-ci n'ait besoin de ses mains.

2.3.3 - Biologie

L'IoT ne s'intéresse pas qu'aux objets inertes, mais aussi aux êtres vivants, pour lesquels de nombreuses données corporelles peuvent être recueillies : rythme cardiaque, pression artérielle, respiration, taux de glucose, etc.

L'objectif majeur de se servir de tels dispositifs est de contrôler sa santé, en anticipant certaines maladies automatiquement ou en demandant l'avis de spécialistes, ou en avertissant les secours en cas de problème grave et urgent tel qu'une crise cardiaque.

Toutes ces informations permettraient à l'homme de vivre plus longtemps et surtout en meilleure santé, à l'heure où l'espérance de vie sans incapacité (EVSI), c'est-à-dire sans problèmes de santé gênants, est en recul dans les pays développés [L04][L05].

2.3.4 - Véhicules

Nos véhicules, et tout particulièrement les voitures, sont depuis quelques temps de plus en plus connectés : il est désormais courant d'y connecter par Bluetooth son smartphone pour écouter de la musique, passer un appel, ou utiliser le GPS, sans avoir à lâcher le volant. Les automobiles ne communiquent pas encore entre elles, mais de nombreuses entreprises, comme Bosch, travaillent sur ce problème pour former ce qu'elles appellent « une conduite interconnectée » [L06]. Cela permettrait aux véhicules de se transmettre des informations, et d'optimiser ou d'automatiser la conduite, créant ainsi une meilleure gestion du trafic. Par exemple, en cas de constatation d'accident, un véhicule pourrait informer les autres véhicules, plus en amont, de la situation. Ceux-ci pourraient alors prendre l'initiative de réduire leur vitesse, pour éviter de créer d'importants embouteillages [A06].

2.4 - Etude de cas : Brad The Toaster

Maintenant que nous avons vu ce qu'était l'Internet Of Things, il serait intéressant d'étudier un cas qui soulève des questions très pertinentes. « Brad The Toaster » est une vidéo [L07][A07] de Simone Rebaudengo se plaçant dans un futur proche, où les objets sont tous connectés et ne possèdent plus de propriétaires, mais seulement des hébergeurs qui doivent les mériter. Brad se trouve être un toaster qui est programmé pour aimer faire des toasts. Si on l'utilise souvent, alors il est content et le fait savoir à travers des réseaux sociaux. Inversement, s'il est peu utilisé, il cherche à attirer l'attention sur lui, et s'il est oublié depuis trop longtemps, il n'hésitera pas à proposer à d'autres utilisateurs, nécessitant un grille-pain, ses services, en indiquant sur le web qu'il est disponible et qu'il aimerait bien changer d'hébergeur. Ces derniers pourront alors aller le chercher dans une perspective de partage pour optimiser son utilité.

Il n'y a pas de précisions sur le modèle économique de l'entreprise qui produit ce grille-pain, mais on peut facilement imaginer un service de location, plutôt qu'un système de vente classique avec achat définitif (s'il me faut un barbecue pour seulement quelques jours par année, pourquoi en acheter un, au lieu de le louer ?). La location aurait comme grand avantage de réduire les coûts du côté du consommateur si l'utilisation est ponctuelle, et d'amener le constructeur vers des produits durables plutôt que jetables.

Cette vidéo montre que grâce à l'IoT, une société basée sur la consommation parfois excessive et pour des utilisations à court terme où les objets sont ensuite délaissés, peut changer vers une société de partage d'objets où ceux-ci auraient un rôle d'entités capables de choix, n'étant plus possédés, mais seulement hébergés par les utilisateurs. Ceci offrirait de nombreux avantages à l'heure où les ressources naturelles s'épuisent, permettant de maximiser l'utilité de tous ces objets, plutôt que de les oublier dans un coin.

D'autre part, en s'inspirant de cette vidéo, on peut facilement imaginer des réseaux sociaux dédiés aux objets connectés, où ceux-ci communiqueraient entre eux et avec les utilisateurs, dans la perspective de partager leurs données, d'indiquer leur état de pensée, de se faire des amis, ou encore de servir de canal de retour sur l'état des produits pour les constructeurs, avec par exemple possibilité d'échanger l'objet loué si celui-ci arrive en fin de vie.

2.5 - Réseaux sociaux

Un réseau social est un service web qui permet à des individus de créer et d'alimenter un profil public avec comme objectif de partager des données, de construire une liste d'autres utilisateurs avec lesquels ils ont une relation, et d'explorer cette liste ainsi que celles des autres personnes inscrites sur le réseau social [A08].

Il existe de nombreux réseaux sociaux, les plus connus étant Facebook (2006), Twitter (2006), LinkedIn (2003) et MySpace (2003). Ce qui fait leur particularité, ce n'est pas seulement le partage d'informations ni le fait que l'on puisse rencontrer des inconnus, mais c'est surtout le fait que l'on puisse explorer et rendre visible notre réseau social, c'est-à-dire nos différentes relations avec des individus. Cela s'exprime la plupart du temps sur ces sites, sous la forme d'une liste d'amis que l'on peut gérer soi-même et partager avec les autres utilisateurs.

Le profil de l'utilisateur peut aussi être alimenté de différentes données comme l'âge, le sexe, l'adresse, des photos ou les centres d'intérêt. D'autre part, les utilisateurs peuvent poster régulièrement des informations sur leur personne, leurs sentiments, leurs avis, etc. Elles sont souvent représentées sous la forme d'un mur d'affichage ou blog consultable par les autres individus. Il y a ici un sentiment de partage de soi-même. D'autre part, ces données ne sont visibles que sous certaines conditions qui dépendent des choix de l'utilisateur, permettant ainsi de garder une certaine intimité.

La majorité des réseaux sociaux visent des groupes de personnes spécifiques qui dépendent d'un même lieu (pays, continent), d'une même religion, d'une même langue ou toute autre catégorie. Il existe même des réseaux sociaux dédiés aux animaux de compagnie comme les chats ou les chiens (Catster et Dogster respectivement), alors pourquoi ne pas aussi imaginer un réseau social pour les objets connectés ?

2.6 - Gestion des données générées par l'IoT

Chaque objet connecté est capable de fournir des milliers de données chaque seconde sur son environnement, son état, etc. Multipliés par des milliards, ceux-ci génèrent et génèreront des quantités phénoménales de données qu'il faudra acheminer, traiter, stocker et récupérer à la demande. Selon les prévisions de Gartner pour 2020 [L08][L09], plusieurs obstacles devront être surmontés :

- **Sécurité :** la sensibilité des données déterminera le niveau de sécurité à prendre en compte. Par exemple, les données des entreprises devront être très sécurisées, alors qu'au contraire, des données environnementales (température, météo) n'ont pas forcément vocation à être protégées.
- **Entreprises :** le nombre d'objets connectés, internes à l'entreprise, mais aussi externes à celle-ci provenant des employés (BYOD), nécessitera d'un côté des améliorations des différents réseaux informatiques pour éviter leur encombrement (ce qui engendrerait une perte de productivité), et d'un autre, un renforcement de la sécurité.
- **Protection des consommateurs :** avec le nombre croissant d'objets connectés par personne, le nombre d'informations privées va croître de façon exponentielle, mais rien ne garantit qu'il n'y aura pas de fuites et qu'elles resteront bien privées.
- **Données :** comment stocker toutes les données générées par l'IoT ? Les infrastructures devront s'adapter.
- **Gestion du stockage des données :** la capacité de stockage des données devra considérablement être augmentée, mais ce n'est pas tout, il faudra aussi optimiser la façon de stocker celles-ci pour pouvoir les récupérer facilement.
- **Serveurs :** il faudra de nouveaux serveurs pour gérer toutes ces données.
- **Datacenter :** le réseau des datacenters devra supporter des flux importants de données, avec un trafic constitué de très nombreux petits paquets, il faudra donc une bande passante plus large.

D'autre part, l'auteur de cet article souligne l'incompatibilité entre IoT et centralisation : il n'est pas technologiquement et économiquement viable de placer toutes les données en un seul endroit, il est préférable d'agréger des données provenant d'autres datacenters.

> *« The recent focus on centralizing applications to reduce costs and increase security is incompatible with the IoT. Organizations will be forced to aggregate data in multiple mini data centers where initial processing can occur. Relevant data will then be forwarded to a central site for additional processing » (Skorupa, 2014).*

2.6.1 - Cloud computing

Le cloud computing peut être vu comme la mise à disposition de ressources informatiques (calculs, stockage, applications, serveurs) à travers un réseau. L'utilisateur accède donc à distance à ces ressources, le plus souvent en se servant d'internet. D'ailleurs, la plupart du temps, on s'en sert sans même s'en rendre compte : webmail (Gmail, Hotmail), services de stockage de fichiers en ligne (Google Drive, DropBox), ou encore plateformes de travail en ligne (ERPNext, Odoo), sont autant de services que l'on utilise dans le cloud.

Par rapport, à une solution locale, le cloud computing offre plusieurs avantages :

- **Flexibilité :** les ressources nécessaires s'adaptent à la demande instantanément, ce qui offre une très grande capacité de montée en échelle : le service peut passer d'un utilisateur à plusieurs millions sans que le client n'ait à se soucier de l'achat de nouvelles machines.
- **Maintenance automatique :** les logiciels sont mis à jour automatiquement, les serveurs défectueux sont remplacés rapidement par des professionnels et le matériel est souvent changé en vue de meilleures performances, le tout sans interruption de service et sans que le client ne s'en rende compte, puisque seul le fournisseur du service cloud s'occupe de la maintenance.
- **Télétravail :** le cloud à l'avantage d'être accessible n'importe où tant que l'on a accès à internet.
- **Paiement à la demande :** le client ne paye que ce qu'il consomme comme ressources. Cela permet aux petites entreprises de ne pas avoir à louer ou acheter des serveurs souvent très chers.
- **Sécurité :** les défaillances techniques résultant en la perte de données sont très rares et le fournisseur de cloud computing s'occupe dans une certaine mesure de la sécurité informatique de ses machines.
- **Reprise après sinistre :** en cas de sinistre, les données ne sont pas perdues, car elles ne sont pas situées au même endroit que l'entreprise. De plus, le plus souvent, les données sont répliquées sur plusieurs clouds dans des lieux différents.

On peut diviser le cloud en trois principaux services :

- **IaaS (*Infrastructure as a Service*) :** le client loue des machines virtuelles qui se comportent exactement comme de véritables serveurs, il n'y a donc pas besoin d'acheter son propre matériel informatique et il possède sur celles-ci un contrôle total. De plus, ces machines virtuelles ont l'avantage de pouvoir être rapidement modifiées en cas de besoin (augmentation de la RAM, de la capacité de stockage, de la puissance de calcul, etc).
- **PaaS (*Platform as a Service*) :** le fournisseur s'occupe de l'installation de l'OS et des applications, quant au client, il possède le contrôle sur ces applications et peut installer ses propres outils.
- **SaaS (*Software as a Service*) :** le fournisseur s'occupe de l'installation des applications, de leur mise à disposition aux clients et de leur maintenance. Le client ne se contente donc que d'utiliser les applications.

Le cloud computing serait donc un excellent moyen pour gérer les données produites par l'IoT. Il offre un service disponible partout, alors qu'actuellement, la plupart des services IoT fonctionnent indépendamment et de façon morcelée. C'est-à-dire, d'un seul ou d'un groupe d'objets connectés vers un terminal (ex : smartphone, tablette), relié à internet, mais qui n'interagit que dans de rares cas avec des objets connectés d'un autre groupe (par exemple, il pourrait être intéressant d'établir une communication entre sa maison intelligente et celle du voisin). Et d'autre part, le cloud computing s'avèrerait très utile grâce à sa capacité de montée en échelle et sa possibilité de support de charge importante générée par l'IoT et sa croissance exponentielle.

2.6.2 - NoSQL

L'IoT génère des données de nature et de type très varié : booléennes, numériques, textuelles, audio, visuelles... autant de données anarchiques qu'il faudra stocker de manière méthodologique et ordonnée. Les bases de données relationnelles ne permettent pas de stocker efficacement autant de données et surtout avec des objets aux nombres d'attributs variables et aussi différents. D'autre part, les réseaux sociaux organisent les relations entre utilisateurs sous forme de graphes, domaine où les bases de données standard ne sont pas très performantes contrairement à certaines bases NoSQL qui s'en sont fait une spécialité. Finalement, celles-ci n'offrent pas une très grande possibilité de montée en échelle face aux besoins de l'IoT composés de plusieurs milliards d'objets.

Pour répondre à cette problématique, de grands groupes ont développé des bases de données NoSQL qui se répandent de plus en plus. Celles-ci ne sont plus organisées par tables au sens strict, c'est-à-dire que le nombre d'attributs pour chaque élément peut varier ainsi que leur type (nombre, texte, graphes, documents...). Elles offrent donc bien plus de flexibilité que les bases de données relationnelles. D'autre part, le langage des requêtes n'est plus SQL, mais dépend de la base de données utilisée, où l'on retrouvera entre autres des extensions de SQL, des accès directs par identifiant, de l'orienté objet, etc.

L'objectif premier de ces bases NoSQL est la performance à toute échelle (pouvoir supporter des téraoctets de données avec des milliards d'éléments), la rapidité d'accès et la spécialisation dans des tâches bien précises pour répondre aux besoins de géants comme Amazon ou Google, exactement ce dont un service d'agrégation de données générées par l'IoT, et de graphe de relation entre objets, aurait besoin [L10].

Il existe quatre grandes familles de bases de données NoSQL :

- **Clé / valeur :** C'est la base NoSQL la plus simple. Chaque élément possède un identifiant unique (une clé), ainsi que plusieurs attributs. Les seules requêtes possibles sont PUT, GET, DELETE. Les données sont donc récupérées brutes et doivent être filtrées coté applicatif. Les bases les plus connues sont Riak, Voldemort et DynamoDB.

- **Orientée document :** Comme pour le modèle précédent, les éléments sont stockés avec une clé, mais ceux-ci ne sont plus sous forme d'attributs, mais sous forme de documents XML ou JSON. Les données sont ainsi stockées de manière plus hiérarchique. Les bases les plus connues sont MongoDB et CouchDB.

- **Orientée graphe :** Ces bases stockent des données de type graphes. Cela s'avère très utile pour les réseaux sociaux par exemple. Les bases les plus connues sont Neo4J et HyperGraphDB.
- **Orientée colonne :** Proche des tables des bases de données relationnelles, ce modèle propose un nombre de colonnes dynamiques. Les données et colonnes sont stockées ensemble plutôt que par lignes. C'est un modèle particulièrement efficace pour effectuer des requêtes rapides sur de larges ensembles. Les bases les plus connues sont HBase, Cassandra et Hypertable.

La plupart de ces bases de données sont open sources, mais nécessitent du matériel conséquent (plusieurs serveurs, voire un datacenter complet) et des compétences professionnelles très avancées. Heureusement, il existe des services clef en main comme le propose Amazon et ses services de cloud computing où le client paye à la demande en fonction des ressources consommées, avec une capacité d'évolution impressionnante (de quelques milliers à plusieurs milliards de requêtes parallèles selon les besoins).

2.6.3 - Agrégateurs de données provenant de l'IoT

À la vue de l'immensité du flux de donnée que l'IoT génère et qui augmente de façon exponentielle, certaines plateformes ont fait du stockage et de l'agrégation de données, provenant des objets connectés, leur cheval de bataille. Ils s'inspirent des services d'Open Data, en proposant de sauvegarder les données dans le cloud, de les récupérer et de les partager avec d'autres objets, applications ou utilisateurs.

C'est par exemple ce que propose la plateforme ThingSpeak [L11], qui fournit aux développeurs une API compatible avec de nombreux dispositifs comme Arduino, Raspberry Pi et Android, dans de nombreux langages (C, Python, Ruby), et qui fonctionne avec n'importe quelle application capable d'envoyer des requêtes HTML. ThingSpeak permet alors de stocker les données non plus localement, mais dans le cloud, de les récupérer pour un usage futur, de les partager avec d'autres objets, et aussi de communiquer indirectement entre plusieurs objets, en se servant de la plateforme comme intermédiaire. C'est donc un service très intéressant pour les développeurs qui s'affranchissent de la contrainte de la gestion du stockage, et qui leur permet aussi de créer bien plus facilement des applications venant s'appuyer sur ces données (ex. : tableau de bord, contrôle des objets à distance).

Il existe aussi un projet européen, OpenIoT [L12], qui regroupe plusieurs dizaines de chercheurs travaillant sur la réalisation d'un middleware capable de récupérer automatiquement des informations provenant de senseurs, sans que l'utilisateur final n'ait à se soucier des senseurs utilisés. Pour se faire, le middleware récupère les données, les stocke dans le cloud, agrège intelligemment celles-ci, pour au final fournir des résultats condensés, pertinents et personnalisés pour l'utilisateur. Un agriculteur peut par exemple [L13] placer plusieurs senseurs sur ses terrains qui obtiendraient comme données l'humidité, l'état des sols ou encore les différentes températures. Le middleware s'occuperait alors de récupérer ces informations, puis fournirait un rapport détaillé et d'éventuels conseils d'optimisation pour améliorer le rendement, prévoir une sècheresse, etc. Il en émerge alors la notion « Sensing-as-a-Service », c'est-à-dire la possibilité pour l'utilisateur d'acquérir de

nombreuses données pertinentes sur son environnement sans s'occuper de la partie technique (matérielle comme logicielle).

En analysant, ces différents agrégateurs de données, on peut toutefois constater un problème : il n'y a pas de notions de relations entre les différents objets. C'est-à-dire que ces services sont très efficaces pour stocker des données en ligne, ce qui permet donc indirectement la communication entre deux objets, mais ne gèrent pas le type de relation qu'ont ces objets ni les autorisations d'échanges et de consultations qui pourraient y être liés. C'est donc aux développeurs de fournir par eux-mêmes une liste d'objets qui peuvent lire et partager des données, suivant certaines restrictions, s'il y en a. On observe aussi que ces services ne s'adressent au final qu'aux développeurs, requérant donc certaines connaissances techniques, alors qu'un des buts de l'IoT est justement de se démocratiser, et par conséquent d'être utilisable par le plus grand nombre.

3 - Problématique et Motivation

3.1 - Problématique

Dans l'exemple de Brad The Toaster, les objets se servent des réseaux sociaux pour interagir entre eux ou avec des personnes. On peut donc imaginer un réseau social non plus seulement centré sur les êtres humains, mais aussi sur ces objets connectés, leur offrant ainsi plus de fonctionnalités qui leur sont propres, tout en gardant les aspects positifs des agrégateurs de données déjà préexistants.

Dès lors, la problématique que l'on peut poser est : comment créer un réseau social dédié aux objets connectés ?

Un tel réseau social permettrait entre autres de créer des relations entre les objets et de publier des informations telles que des données brutes, des sentiments ou toutes autres formes d'interactions. Nous pourrions même imaginer une surcouche permettant de créer différents services pour ces objets et pour les utilisateurs, à la manière des applications que l'on retrouve sur Facebook.

Pour des raisons de similitude avec le premier réseau social mondial, Facebook, nous proposons d'appeler ce nouveau réseau social entièrement dédié aux objets connectés : ThingBook.

3.2 - Motivation

Voici un exemple très simple, mais ô combien révélateur de l'utilité d'un tel service :

La société OrbiWise est spécialisée dans des solutions de type « Internet of Things ». Un de leur projet vise à construire un senseur de l'état de remplissage des poubelles de la ville de Genève [L14]. Ce capteur se place directement sous la couverture anti-pluies des poubelles publiques, est relié à Internet, et indique au service de la voirie s'il faut vider les poubelles ou non. Grâce à celui-ci, la mairie pourra alors optimiser les déplacements de ses employés et rajouter des poubelles aux endroits nécessaires.

Cette entreprise choisira surement de développer sa propre plateforme de centralisation pour l'ensemble de ses capteurs. Cette plateforme s'occupera de référencer l'ensemble des senseurs, de sauvegarder leurs données, d'agréger celles-ci et d'indiquer via une interface leur état. Cela a un coût en terme de temps, de ressources et donc au final d'argent. Pourtant, une partie de ce travail pourrait être évité en utilisant justement un service tel que ThingBook, dont un des rôles est de référencer les différents objets, de stocker les données qu'ils postent, et de fournir une API permettant de développer des applications. Ainsi la société OrbiWise n'aurait plus alors qu'à se concentrer sur l'interface utilisateur finale plutôt que sur des processus de gestion et de transmission de données entre objets, déjà implémentés dans le réseau social.

4 - Scénarios

À travers les différents scénarios suivants, il nous sera possible de nous faire une idée de l'utilité d'un tel réseau social, tout en identifiant, en parallèle, les besoins qui en ressortiront. Ces scénarios se déroulent dans un monde futuriste, mais pas si lointain, où l'utilisation d'objets connectés est fréquente et où ils sont omniprésents.

4.1 - Scénario 1

Marc, 45 ans, est chauffeur de taxi dans la 1ère ville 100% connectée du monde : Genève. Il vient tout juste de recevoir son nouveau taxi connecté, et, comme beaucoup de ses collègues, il choisit d'inscrire son taxi sur ThingBook. Son taxi lui propose alors de devenir ami avec les autres taxis de ses collaborateurs et son smartphone. Tout cela est bien nouveau pour Marc, mais il décide d'accepter pour voir ce que cela va donner.

Le lendemain, comme à son habitude, Marc monte dans son véhicule. Celui-ci démarre alors tout seul, ce qui surprend Marc. En effet, le smartphone a indiqué au taxi qu'il était proche, grâce au GPS, et le taxi a alors pris l'initiative de démarrer tout seul pour accueillir son conducteur. Marc se rend alors vers son premier client de la journée qui se situe malheureusement à l'autre bout de la ville et aux heures de pointe. En consultant les profils des autres taxis, des routes, et des feux de signalisation sur le réseau social, le taxi propose alors à Marc le meilleur itinéraire pour rejoindre sa destination, et le poste sur son profil à destination des autres taxis qu'il a en ami. Il permet ainsi à Marc de lui faire gagner plus de 20min, ce qui le ravit. Tout au long de la journée, le taxi ne cessera d'étonner Marc, en ralentissant automatiquement à l'approche d'un feu de signalisation au rouge, en ajustant tout seul son allure en fonction du trafic et des limites de vitesse, ou en corrigeant automatiquement sa trajectoire s'il s'approche trop d'une ligne blanche, d'un piéton ou d'un obstacle. Tout cela est possible grâce à la publication de données publiques des différents objets connectés de la ville de Genève sur ThingBook. Elles sont alors consultées par le taxi, qui s'adapte en fonction de celles-ci. La perte de contrôle apparente sur la conduite de son véhicule, due au fait que son taxi est plus autonome, a tendance à l'énerver, mais après tout c'est pour sa sécurité et celle des autres usagers.

Après plusieurs mois d'utilisation, Marc a fait le point sur ses comptes et s'est aperçu qu'il a pu effectuer 20% de transports en plus, donc obtenir un meilleur bénéfice. D'un autre côté, il a aussi observé qu'auparavant, conduire son taxi lui demandait un niveau de concentration perpétuelle, donc une grande fatigue en fin de journée. Désormais, avec son nouveau taxi plus autonome, la pression est relâchée, ce qui l'épuise beaucoup moins.

4.2 - Scénario 2

Marine, 23 ans, vient juste de se mettre au jogging et décide d'acheter une paire de chaussures connectées pour faire du sport. En magasin, le vendeur lui propose d'inscrire pour elle ses nouvelles chaussures sur ThingBook, le réseau social des objets connectés, lui expliquant que grâce à ce service, Marine pourra suivre l'évolution de ses performances, les kilomètres parcourus, ainsi que les trajets qu'elle a effectués. Il lui fait aussi remarquer, par la même occasion, qu'elle peut aussi inscrire sa montre connectée capable de mesurer son rythme cardiaque, sa tension, ainsi que son taux d'oxygénation. Les objets pourront alors communiquer et échanger des informations entre eux. Cela lui permet d'obtenir plus de renseignements sur son endurance et ses performances lorsqu'elle fera son jogging. La montre affichera ces données, ainsi que des conseils personnalisés sur son écran. Ravie, elle accepte, se crée un compte utilisateur, et laisse le vendeur inscrire les chaussures et sa montre pour elle.

Ses chaussures enfilées, elle décide de rentrer chez elle en footing puisqu'elle n'habite qu'à 15min du magasin. Sur son chemin, elle constate en effet que sa montre affiche automatiquement son pouls, la distance parcourue depuis qu'elle a commencé à courir, ainsi que sa vitesse moyenne. Une fois chez elle, elle se connecte sur ThingBook et constate que ses nouvelles chaussures ont publié une quantité phénoménale de données, avec entre autres : sa température corporelle, le nombre de pas qu'elle a parcourus, les coordonnées GPS de son chemin, ou encore les calories dépensées, le tout à intervalle très régulier. Plus surprenant encore, les chaussures ont même publié leur état d'esprit concernant Marine, et semblent pour l'instant très contentes de leur nouvelle utilisatrice. Avec autant de données personnelles, Marine a très peur que d'autres personnes puissent l'espionner publiquement. Heureusement, le vendeur a eu le bon réflexe en limitant les données publiées par les chaussures au cercle de ses « amis » objet, comme la montre de Marine, empêchant ainsi des objets inconnus de consulter des données privées.

Malheureusement, après seulement quelques mois, Marine se désintéresse du sport et cesse de faire du jogging. Ce n'était finalement qu'un passe-temps de courte durée. Ses chaussures connectées le constatent bien et cherchent à avertir Marine en publiant leur état mélancolique sur ThingBook. Mais rien n'y fait, Marine a définitivement abandonné le jogging. Les chaussures envisagent alors, de leur propre chef, de changer de propriétaire, plutôt que d'être oubliées ou jetées. Pour ce faire, elles signalent régulièrement sur leur profil public, qu'elles sont prêtes à changer d'hôte si celui-ci aime le sport, et en parallèle, elles avertissent Marine de cette volonté. Seulement quelques jours après, le smartphone de Mélanie, une grande sportive, repère la demande de changement de propriétaire et l'informe qu'une paire de chaussures connectées de sa pointure est disponible et voudrait changer d'utilisateur. Mélanie accepte donc cette proposition via ThingBook. Depuis son ordinateur, Marine reçoit alors une notification comme quoi ses chaussures connectées aimeraient

changer de propriétaire et qu'elles ont trouvé preneur. Elle peut bien sûr refuser, mais plutôt que de laisser ses chaussures inutilisées, elle préfère les donner. Le lendemain, les deux femmes se rencontrent, Marine donne sa paire de chaussures à Mélanie et valide le changement de propriétaire sur le réseau social, permettant ainsi de rendre les chaussures plus heureuses et avec un nouveau propriétaire plus adapté.

4.3 - Scénario 3

Martin est un jeune entrepreneur qui vient de fonder sa start-up dans le domaine de l'informatique, et il s'intéresse tout particulièrement à l'IoT. Il souhaite développer une application capable de gérer les consommations énergétiques au sein des entreprises. Utiliser des bornes d'accès et de contrôle pour les objets connectés lui demanderait un coût élevé en recherche et en développement pour créer des prototypes. Ce coût qui se répercuterait alors inévitablement sur la vente du produit, et pourrait dissuader certaines entreprises. En outre, installer du matériel en plus dans des sociétés ne sera pas forcément accepté. Martin est aussi confronté à un autre problème : les entreprises possèdent de nombreux objets connectés, mais la plupart sont de marques différentes. Il est donc difficile de les faire communiquer entre eux et de coordonner certaines interactions.

Martin a lors choisi d'utiliser ThingBook. En faisant des tests au sein de sa propre entreprise, il a inscrit l'ensemble des objets connectés de son bâtiment (capteurs, actuateur, affichages, etc.) sur le réseau social. Il s'est ensuite mis au défi, avec son équipe, de créer une application capable de gérer l'intégralité de ces objets pour optimiser la dépense énergétique de ses locaux, en se servant de l'API fournie ThingBook. Après plusieurs mois, il est parvenu à développer une application indépendante qui permet en moyenne d'économiser plus de 30% de ressources énergétiques (un budget important pour de grandes entreprises, provoquant d'importantes économies), le tout, sans devoir installer de matériel supplémentaire dans les entreprises, hormis éventuellement certains capteurs nécessaires pour la mesure des quantités d'énergies consommées.

Exigences

Les exigences se décomposent en plusieurs niveaux, avec différentes parties prenantes. Elles permettent de définir nos potentiels utilisateurs, ainsi que leurs besoins. Parmi ces participants, nous retrouvons les utilisateurs lambda, les objets connectés et les développeurs. Nous verrons aussi à quelles caractéristiques doit répondre le réseau social, notamment en ce qui concerne la grande quantité d'objets souhaitant utiliser le service.

4.4 - Exigences des utilisateurs

Intégration des objets au sein du réseau social : Les utilisateurs doivent pouvoir enregistrer leurs objets connectés sur le réseau social. Cela peut se faire, pour l'utilisateur, depuis l'interface de contrôle de ces objets (ex. : écran d'une montre connecté), puis, pour les objets concernés, en passant par une API capable de communiquer avec le réseau social. Inversement, les utilisateurs doivent pouvoir retirer leurs objets du réseau social.

Contrôle des autorisations : Les utilisateurs doivent pouvoir gérer les informations qui seront partagées par leurs objets connectés. Cela se fera de préférence en se servant de groupes pour ne pas à avoir à gérer chaque objet un par un, surtout dans un contexte où leur nombre ne cessera de croître.

4.5 - Exigences des objets connectés

Communication avec le réseau social : Les objets connectés doivent pouvoir échanger des informations avec le réseau social. Cela peut se faire par exemple via une API REST, c'est-à-dire, en envoyant des requêtes HTTP spécifiques au serveur.

Gestions des relations : Les objets connectés doivent pouvoir créer des relations entre eux et consulter celles d'autres objets, en respectant les contraintes imposées par les autorisations définies par les utilisateurs. Avec une perspective plus lointaine, nous pourrions aussi imaginer que des applications émergent pour gérer automatiquement les relations entre objets, surtout si on en possède plusieurs centaines voire milliers.

Échange et publication de données : Les objets connectés doivent pouvoir envoyer des données sur le réseau social (ex. : état de l'objet, interactions avec des utilisateurs, état d'esprit). Ces données peuvent ensuite, plus tard, être modifiées, supprimées, ou changer d'autorisation. Ils doivent aussi pouvoir consulter les données d'autres objets et communiquer avec eux à travers le service.

Changement de propriétaire : Les objets connectés doivent pouvoir changer de propriétaire, que ce soit dans une perspective de vente, de partage ou pour toute autre raison, avec l'autorisation de leur ancien propriétaire. Lors de cette étape, il faudra particulièrement veiller aux données à conserver pour respecter l'intimité de l'ancien propriétaire.

4.6 - Exigences des développeurs

Création de nouveaux services : Les développeurs doivent pouvoir créer des applications, intégrées au réseau social, se servant des données publiées par les objets connectés pour toute sorte d'objectifs comme : l'automatisation et l'optimisation de certaines tâches qui demandent la coordination de plusieurs objets, la création d'un tableau de bord qui permettrait d'agréger des informations, ou encore d'apporter un système de contrôle direct des objets connectés.

4.7 - Exigences du réseau social

Support de la montée en échelle : Le réseau social d'objets doit être capable de supporter une grande montée en échelle, puisque comme nous l'avons vu un peu plus tôt dans ce document, le nombre d'objets connectés croît de façon exponentielle, et donc le flux de données généré par ces objets aussi. Le cloud computing et les bases de données NoSQL sont une bonne solution pour répondre à cette problématique.

Simplicité d'utilisation : Le réseau social doit pouvoir être utilisable par le plus grand nombre, il faudra donc veiller à son ergonomie et son utilisabilité, tout en offrant une partie plus avancée pour les utilisateurs désireux d'avoir plus de contrôle, ainsi qu'une partie réservée aux développeurs.

5 - Architecture et Design

5.1 - Architecture globale du service

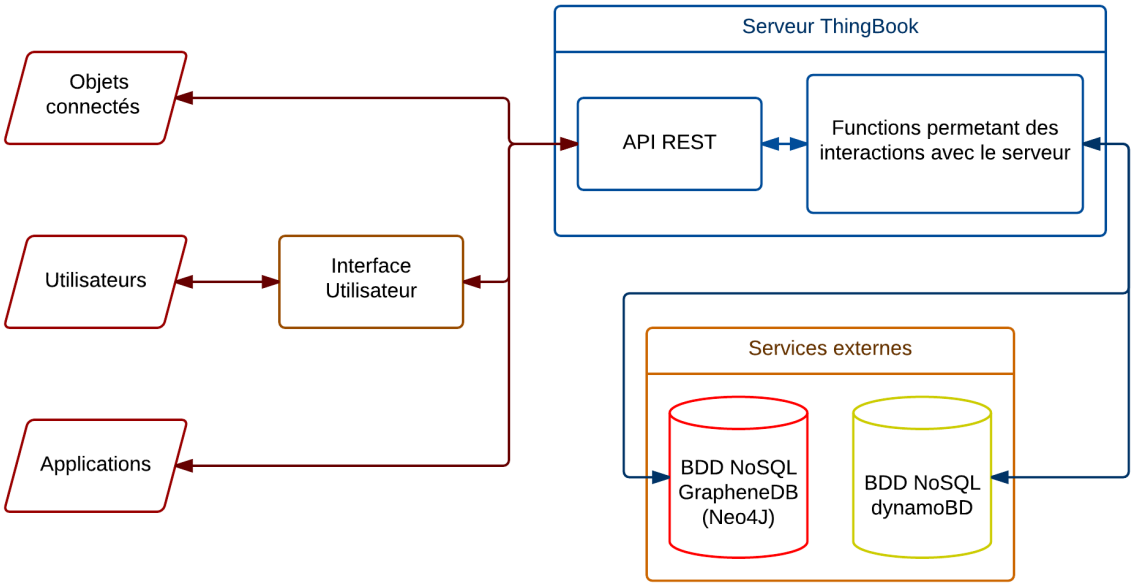

Figure 3 - Architecture simplifiée du service ThingBook

Objets connectés, utilisateurs et applications ont des rôles différents, mais passent par la même porte d'entrée pour communiquer avec le réseau social et parfois utilisent les mêmes processus, de ce fait, nous utiliserons le terme « entité » pour désigner l'un de ces trois acteurs. D'autre part, objets et applications ne sont tous deux au final que des *programmes* exécutés sur des machines dont seule la nature diffère. Par conséquent, ces entités auront les mêmes fonctionnalités et seul leur type les différenciera.

Pour communiquer avec le service, ces entités se serviront d'une API de type REST, c'est-à-dire en respectant le processus suivant :

1. Une entité envoie une requête HTTP avec une URL spécifique qui ordonnera au serveur quelle opération effectuer.
2. Le serveur traite cette demande et répond au format JSON.

Il est facile pour une machine de se servir d'une API REST, mais pour un utilisateur novice, taper des URL complexes sous forme textuelle pour interagir avec le serveur n'est pas chose aisée. Aussi, une interface utilisateur devra être fournie pour permettre un accès au plus grand nombre. Cette interface peut être réalisée en HTML, CSS, JavaScript, PHP, puis hébergée directement sur le serveur de ThingBook.

Finalement, le service communiquera aussi avec deux services externes qui sont GrapheneDB et DynamoDB, deux bases de données NoSQL qui supportent une forte montée en échelle.

5.2 - Processus de l'API

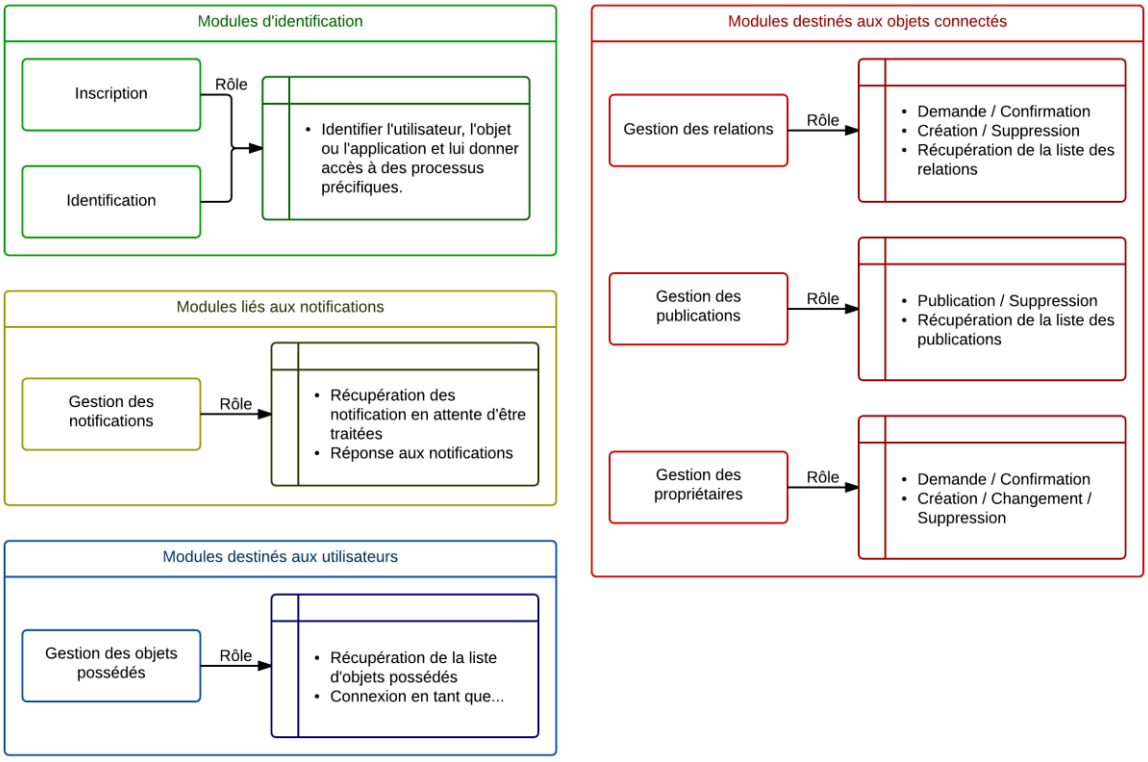

Figure 4 - Détails internes des processus que devra gérer l'API

L'architecture du réseau social d'objets connectés se décompose globalement en quatre sous-parties :

5.2.1 - Modules d'identification des entités

Objets connectés, utilisateurs et applications utiliseront tous le même mode d'identification :

1. **Inscription** : ce module a pour rôle d'inscrire une entité au réseau social. Objets et applications devront fournir une paire id/clé pour créer un compte, et les utilisateurs un email et un mot de passe.

2. **Identification** : grâce au couple id/clé transmis via la requête HTTP, ce module authentifiera l'entité et lui autorisera des actions limitées en fonction de son type (un objet aura accès à des fonctionnalités destinées aux objets et non à celles des utilisateurs).

5.2.2 - Modules de gestion des notifications

Les notifications permettent d'informer une entité d'un changement, et éventuellement de lui demander une réponse si le contexte l'exige (par exemple, si la notification est une demande de changement de propriétaire, le nouveau propriétaire visé devra choisir si oui ou non il accepte de posséder cet objet). On ne peut pour l'instant que décomposer cette partie en un seul module qui aura plusieurs rôles :

1. **Gestion des notifications** : ce module permettra d'une part aux entités de récupérer leurs notifications qui n'ont pas encore été lues, et d'autre part de répondre aux notifications qui sont en attente d'un retour.

Les notifications ne sont générées que par le serveur en réaction à des actions effectuées par d'autres entités. En aucun cas un objet ne peut notifier un autre objet via ce mécanisme. En effet, pour les interactions entre objets, il existe le module publication, bien plus adapté.

5.2.3 - Modules destinés aux utilisateurs (personnes physiques)

Les utilisateurs veulent un contrôle final sur leurs objets, c'est-à-dire de pouvoir surveiller le comportement de ces derniers : publications, relations, autorisations, ou encore changement de propriétaire, ceci dans le but de leur garantir un tant soit peu de vie privée.

Les utilisateurs auront à leur disposition une interface web pour pouvoir interagir facilement avec le réseau social. Comme pour toutes entités, les utilisateurs devront dans un premier temps s'inscrire au réseau social, puis s'identifier, leur permettant ainsi un accès au module suivant :

1. **Gestion des objets :** ce module permet à l'utilisateur de consulter la liste des objets dont il est propriétaire. De plus, il autorise l'utilisateur à se connecter sur le profil d'un de ses objets et ainsi à accéder à l'ensemble des modules qui leur sont réservés. L'utilisateur obtient alors une interface web qui lui permet de suivre les publications de son objet, de les supprimer, de consulter ses relations, etc. Il obtient donc exactement les mêmes fonctionnalités que ses objets connectés, permettant ainsi de garder un contrôle sur eux.

5.2.4 - Modules destinés aux objets connectés

Les objets connectés sont les principaux utilisateurs de ce réseau social, qui s'articulera donc en majorité autour d'eux.

Dans un premier temps, tous les objets voulant accéder au service devront s'inscrire à celui-ci. Cette inscription n'a pas besoin de validation de la part d'une personne physique, mais tant que l'objet n'a pas de propriétaire officiel, l'ensemble des données qu'il publiera resteront privées. Une fois cette inscription effectuée, les objets pourront dans un second temps s'identifier, leur donnant ainsi accès à toutes les fonctionnalités qui leur sont réservées :

2. **Gestion des relations** : ce module a plusieurs rôles, tous liés au management des relations qu'entretiennent les objets, c'est-à-dire s'ils se connaissent, sont amis, ou toute autre forme de lien. Il aura notamment pour rôle de gérer les demandes et confirmations de relations, de créer et supprimer ces relations en fonction des demandes, et de lister l'ensemble des relations d'un objet en cas de consultation. Ce module fait entre autres appel au service externe GrapheneDB pour gérer les relations sous forme de graphe via Neo4j.

3. **Gestion des publications :** ce module a pour principale tâche de gérer l'ensemble des publications des objets. Il aura entre autres pour rôle de stocker les publications, de les supprimer si l'objet concerné en fait la demande, de les transmettre aux objets qui veulent les consulter (sous réserve de limitations en fonction des relations qu'entretiennent les objets), et finalement, de gérer les autorisations liées à chacune de ces publications. Ce module fait appel à deux bases de données, l'une provenant d'un service externe pour le stockage brut des données (DynamoDB) et une autre interne pour un accès plus rapide (MySQL).

4. **Gestion des propriétaires :** dernier point important, la gestion des propriétaires. Chaque objet est lié à un seul propriétaire qui possède, in fine, le contrôle sur celui-ci (relations, publications, etc.). Ce propriétaire n'est pas forcément une personne physique, il peut être une entité, un groupe ou encore une institution. Le module aura pour rôle de gérer les demandes, confirmations et changements de propriétaire.

5.2.5 - Modules destinés aux applications

Les applications sont un point très important du réseau social, permettant de développer une grande quantité de services annexes pour les utilisateurs. Une application requiert en fin de compte toutes les fonctionnalités qu'aurait un objet. Par conséquent, plutôt que d'utiliser des robots logiciels sur des comptes destinés aux objets, les développeurs pourront enregistrer leurs applications en tant que telles et avoir accès aux mêmes modules que les objets connectés.

Il faut bien comprendre qu'une application n'est pas un programme qui prend le contrôle du compte d'une entité (ex. : grâce à ses identifiants), mais bien un acteur qui pourra publier différentes données comme le ferait un objet, et qui créera différentes relations avec des objets qui le souhaitent. Par exemple, nous pourrions imaginer une application « interrupteur » qui publie *true* ou *false* en fonction de l'état allumé ou éteint de cet interrupteur virtuel. De nombreux objets pourraient alors s'abonner à cette application via une relation « subscribe », consulter les publications, et réagir en conséquence.

5.3 - Rôles des objets connectés

Certains rôles et processus sont plus pertinents s'ils sont exécutés du côté de l'objet connecté, ils ne seront donc pas inclus dans le réseau social. Parmi ces rôles, voici quelques exemples :

- **Inscription :** l'inscription devra être initiée depuis l'objet connecté, c'est-à-dire soit par un choix du constructeur, une décision autonome de l'objet, ou par l'utilisateur via une interface de l'objet connecté. Le réseau social s'occupera alors de traiter et de valider la demande d'inscription. Celui-ci n'aura donc pas de fonctionnalité utilisateur pour inscrire un objet (par exemple au niveau de l'interface web). Ce choix vient d'une part du fait qu'il est plus simple d'inscrire des objets directement depuis leur propre interface, et d'autre part parce qu'il permet d'automatiser en toute facilité ce processus pour les constructeurs. Autre point, les objets connectés devront avoir un identifiant unique déjà implémenté et standardisé (un peu comme les adresses MAC) ceci pour éviter tout problème de reconnaissance de l'objet.

- **Autorisations :** les objets connectés pourront publier de nombreuses données de nature très variée (texte, nombre, son…), avec des tags non formalisés qui définiront leur type (température, vitesse, pression…), ainsi que les types de relations (publique, amis) autorisées à consulter ces données. L'objectif est de ne pas limiter les développeurs en leur imposant par exemple une liste fixe de tags. Cette liberté a un prix : les autorisations devront être gérées du côté des objets connectés. En effet, puisque les tags sont libres, un objet peut très bien envoyer une donnée de type température avec comme tag « vitesse », ceci dans le but de contourner les sécurités du réseau social, si par exemple celles-ci interdisent de rendre publique toute donnée avec le tag « température ». De telles sécurités seraient donc inutiles. Les restrictions de partage des données (avec qui je partage cette information ?) devront par conséquent être implémentées du côté de l'objet connecté, avec de préférence une interface de contrôle pour l'utilisateur.

5.4 - Architecture technique et choix technologiques

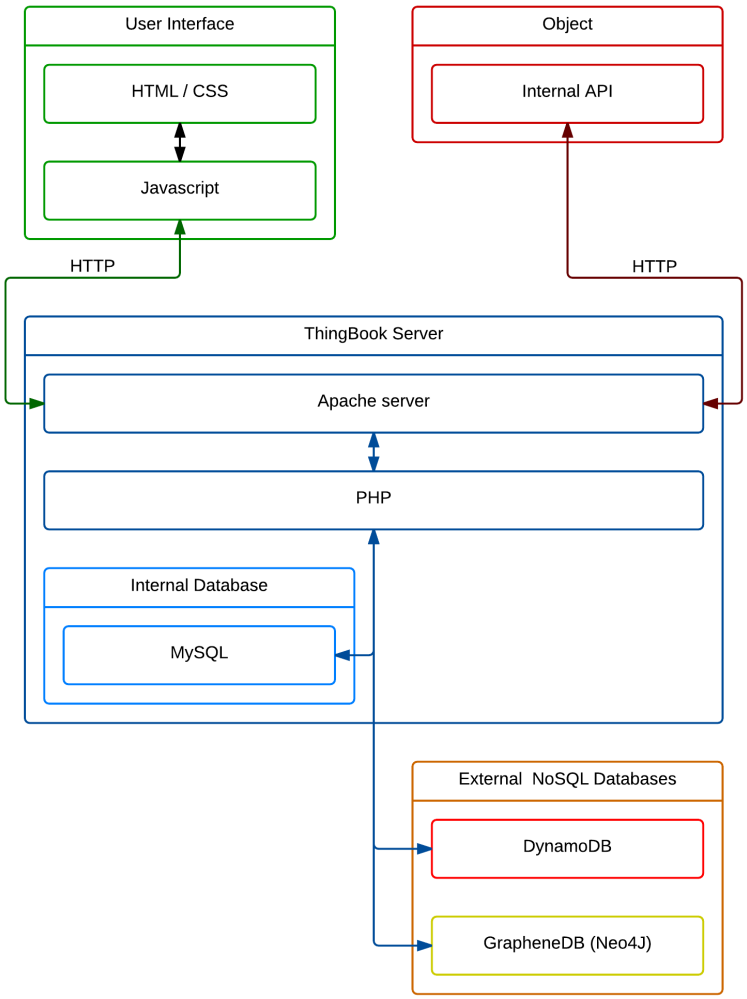

Figure 5 - Couches logicielles du service

5.4.1 - Interface utilisateur, scripts serveur et objets

ThingBook étant un service web, le serveur doit être capable de répondre à ces besoins. Pour ce faire, nous utiliserons Apache comme serveur HTTP. C'est le programme le plus utilisé dans le monde du web, et fonctionne sur de très nombreuses plateformes (Windows, Linux, MacOS) garantissant ainsi une grande portabilité. De plus, une surcouche HTTPS sera aussi disponible pour garantir plus de sécurité et de confidentialité aux utilisateurs et aux objets connectés.

Pour le développement du service lui-même, nous nous servirons de PHP comme langage de programmation. C'est là aussi le langage le plus utilisé pour les scripts côté serveur web. En collaboration, les langages HTML, CSS et JavaScript seront aussi utilisés pour l'interface utilisateur. Ils permettront de générer les différentes pages visibles depuis un navigateur internet, en plus de les rendre interactives et dynamiques.

Les développeurs, objets et applications pourront quant à eux utiliser l'API REST pour communiquer avec ThingBook. L'architecture REST est très utilisée dans ce domaine, et fonctionne de la façon suivante : un programme envoie une requête HTTP avec une URL bien spécifique (ex. : *http://domaine.com/?object=a7d8&key=5sp4&data=18*) permettant de transmettre ainsi des données au serveur. Le serveur traite alors ces informations, et renvoie une réponse sous forme de JSON, qui est un format très répandu, facile à parser et compréhensible par l'homme.

5.4.2 - Bases de données

ThingBook utilisera trois bases de données différentes pour le stockage persistant :

1. **Une base de données MySQL** en interne pour gérer les comptes (utilisateurs, objets et applications), les propriétaires, les notifications, et les publications (la partie concernant les données sera effectuée par une autre BDD).

2. **Une base de données NoSQL de type graphe**, avec comme choix Neo4j. Cette BDD est très répandue et propose une syntaxe intuitive nommée Cypher, qui permet de créer des relations entre différents nœuds. Comme pour Amazon, nous utiliserons un service externe capable de monter très facilement en échelle, nommé GrapheneDB [L15], et qui héberge des bases Neo4j. Cette BDD aura pour rôle de stocker les relations entre les différents objets.

3. **Une base de données NoSQL de type key-value**. Amazon propose ce service sous le nom de DynamoDB [L16], garantissant un stockage avec des types de données très variés et supportant une très grande montée en échelle. Le client ne paye alors qu'en fonction des ressources consommées, plutôt que d'investir dans des serveurs couteux capables de supporter un afflux exponentiel de données. Cette BDD aura pour rôle de stocker l'intégralité des données publiées par les objets connectés.

6 - Prototype

Dans cette partie, nous allons matérialiser le framework vu précédemment au travers d'un prototype. Celui-ci s'organisera en trois axes principaux :

- La conception d'un schéma et l'intégration des différentes bases de données.
- La réalisation d'un API, servant à faire le lien entre les objets connectés et le service proposé.
- Le développement d'une interface utilisateur garantissant plus de simplicité pour ces derniers.

Le code source du projet est disponible sur Github [L17].

Étant donné la nature un peu particulière des applications, qui ont finalement un rôle très proche des objets connectés, car ce sont des programmes au même titre que ceux exécutés sur les objets, nous avons choisi de fournir un prototype dans lequel celles-ci ne sont pas encore présentes, car remplaçables par de « faux » objets qui seraient en réalité des applications. Toutefois, l'ensemble de l'architecture et du code mis en place prévoit déjà cet ajout futur, sans devoir changer les scripts préexistants.

6.1 - Bases de données

6.1.1 - MySQL

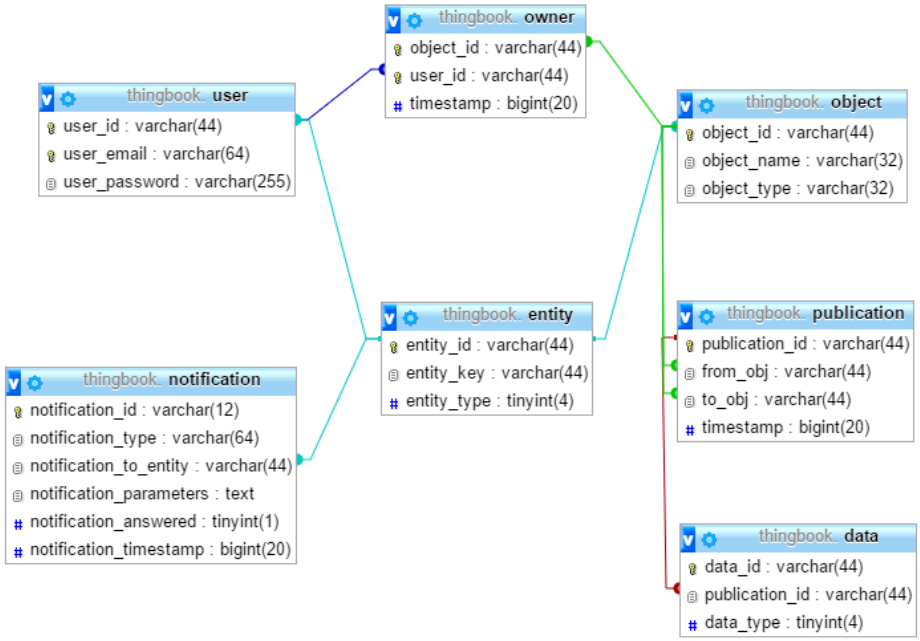

Figure 6 - Base de données MySQL interne

Si l'on reprend les différents points évoqués dans l'architecture vue précédemment, nous retrouvons ici la table *entity* qui permet pour chaque entité de connaitre son identifiant, sa clé privée et son type (objet, utilisateur, application ?). Cette table sert principalement à l'authentification de l'entité et permet de connaitre les processus côté serveur que cette entité pourra utiliser.

Les tables *user* et *object* viennent ajouter des attributs supplémentaires aux utilisateurs et aux objets, par exemple l'email et le mot de passe de l'utilisateur, ou encore le nom et type de l'objet.

La table *owner* lie objets et utilisateurs pour identifier les propriétaires. L'architecture choisie pour ces tables permet à un utilisateur de posséder plusieurs objets, mais permet aussi à un objet d'être possédé par plusieurs propriétaires. Pourtant, comme vu plus tôt dans ce document, nous avons choisi de limiter les objets à un seul propriétaire (celui-ci peut très bien être une entité morale, et par conséquent, pas nécessairement une personne physique). Ce choix s'est fait pour des raisons plus ou

moins juridiques : vers qui se tourner si l'objet blessait un utilisateur (par exemple pour une voiture connectée) ? Le propriétaire permet donc d'identifier un responsable. Toutefois ceci mériterait probablement d'être étudié plus en profondeur et pourrait faire l'objet d'un autre travail de recherche. Pour en revenir à la table *owner*, la limitation à un seul utilisateur se fait côté code, plutôt que du côté des tables, permettant ainsi une plus grande flexibilité en cas de changements futurs.

La table *notification* stocke les différentes notifications adressées aux entités. Une notification a un type (sur quoi elle informe) et des paramètres (de qui provient la notification, pour quelle raison ?).

Les tables *publication* et *data* permettent quant à elles d'enregistrer les différentes publications des objets ou applications.

6.1.2 - GrapheneDB (Neo4J)

Les graphes sont de formidables outils pour gérer des relations entre différents nœuds. Ils permettent de tracer de véritables chemins pour trouver des informations. Nous avons donc choisi d'utiliser une base de données orientée graphe pour gérer les relations entre chaque objet. Un des objectifs du projet étant de garantir une qualité de service constante, quel que soit le nombre d'objets connectés, nous nous sommes servis de GrapheneDB, un hébergeur cloud de BDD Neo4j, qui supporte facilement une très forte montée en échelle. De plus, les développeurs ont créé plusieurs APIs, dans de nombreux langages de programmation, permettant de communiquer avec leur service.

Neo4j utilise comme syntaxe Cypher qui est un langage proche de SQL, capable d'effectuer des requêtes à la fois simples et complexes, autorisant l'utilisateur à créer de véritables graphes et à les parcourir de façon très intuitive. Ce fut pour moi une découverte très instructive, tant les possibilités sont vastes.

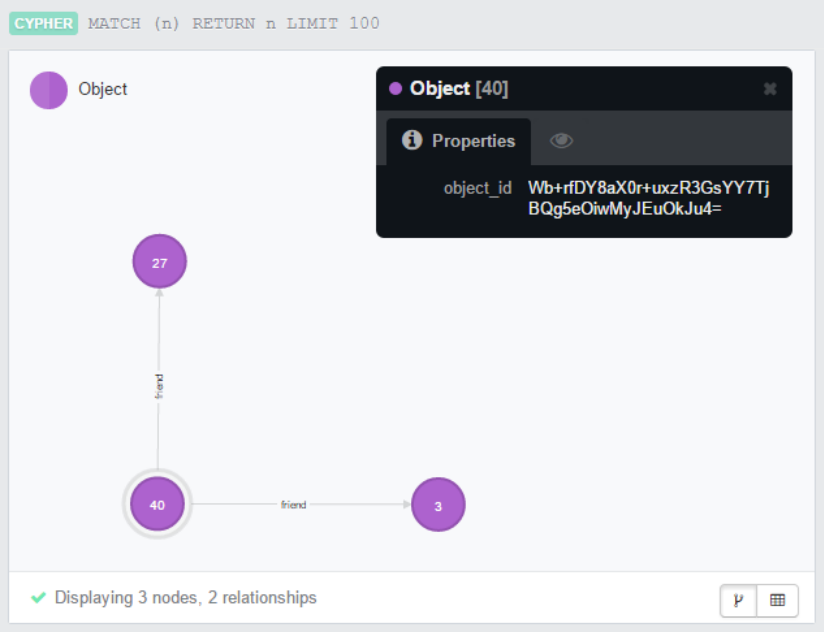

Figure 7 - Base de données Neo4j

La figure précédente montre les relations qu'entretiennent trois objets entre eux :

Le nœud 40, défini par son identifiant *object_id*, possède la relation « friend » avec deux autres nœuds : 3 et 27. Dans Neo4j, chaque nœud a un numéro et peut posséder plusieurs attributs, mais ce numéro ne sert pas à identifier le nœud ; seul *object_id* le permet. Les relations peuvent être orientées (à sens unique) ou non (valables dans les deux sens). Nous avons choisi des relations non orientées, c'est-à-dire que le nœud 3 est aussi « ami » avec le nœud 40 (l'afficheur n'affiche malheureusement pas correctement les relations non orientées).

6.1.3 - DynamoDB

DynamoDB est une BDD NoSQL de type key-value, dont le fonctionnement est très proche des tables relationnelles que l'on connait, mais qui autorise des types de données très variés, un nombre d'attributs variables et surtout, comme pour GrapheneDB, supporte une montée en échelle élevée si besoin. Nous nous sommes donc servis de ce service comme support des données générées par les objets connectés.

Figure 8 - Base de données DynamoDB

L'attribut *data _id* est la clé d'identification de chaque ligne, *relationships* représente un tableau des relations autorisées à lire cette donnée, même chose pour *tag* qui est un tableau des tags associés à la donnée. Finalement *value* représente la donnée brute, qui peut être un chiffre, une chaine de caractères ou encore des données binaires.

6.2 - Développement de l'API

6.2.1 - Environnement de développement

Comme décrit un peu plus tôt, le serveur HTTP est Apache 2.4.9 avec une couche HTTPS, le langage pour l'API et les scripts côté serveur est PHP 5.5.12 (dernières versions stables lors de la rédaction de ces lignes).

Pour l'installation des librairies PHP qui permettent de communiquer avec les bases de données externes GrapheneDB et DynamoDB, il faut se servir de Composer. Composer est un outil de gestion des dépendances en PHP, c'est-à-dire qu'il permet de déclarer les bibliothèques indispensables à un projet et s'occupe de les installer automatiquement (par exemple depuis un dépôt git).

6.2.2 - Scripts

L'API représente la partie la plus importante du projet. Son objectif est de faire le lien entre le réseau social et les différentes entités (utilisateurs, objets connectés et applications).

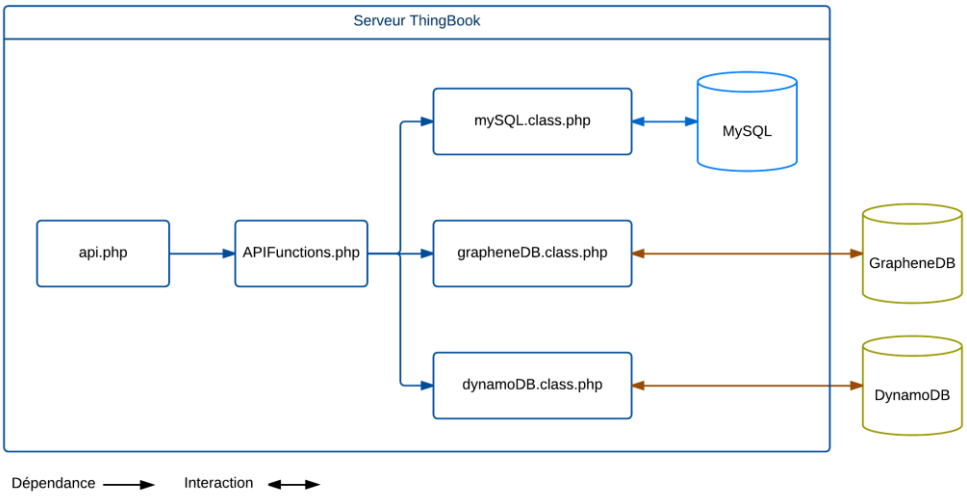

Figure 9 - Scripts en rapport avec l'API

- **api.php :** c'est l'interface d'entrée vers le réseau social. Les requêtes HTTP pour communiquer avec l'API se font sur cette page. Celle-ci à rôle de faire le lien entre une requête HTTP qui transmet une demande spécifique au serveur via GET ou POST, et un modèle *Object* compréhensible par une machine.

- **APIFunctions.php :** ce script contient l'ensemble des fonctions qui permettront d'effectuer des actions côté serveur. Il s'occupe par exemple d'enregistrer les entités sur le réseau social, de gérer les publications, etc. Si les scripts *api.php* et *APIFunctions.php* ne sont pas regroupés sur un seul fichier, c'est pour permettre de créer d'autres interfaces d'entrée facilement, sans avoir besoin de tout recoder. Nous pourrions par exemple imaginer un ajout permettant de communiquer grâce aux sockets.

- **mySQL.class.php, grapheneDB.class.php, dynamoDB.class.php:** ces trois librairies sont composées de méthodes qui interagissent avec les bases de données. Ces méthodes simplifient certains traitements, notamment la phase de connexion. Les classes *grapheneDB.class.php* et *dynamoDB.class.php* utilisent en plus les librairies fournies par Amazon (DynamoDB) et GrapheneDB pour établir les connexions distantes avec leurs services respectifs.

6.2.3 - Documentation

Cette section a pour but d'expliquer sommairement comment fonctionne l'API, et quelles actions peuvent exécuter les entités. Une documentation de l'API bien plus complète et longue est disponible sous format XML [L18]. Il est donc indispensable de comprendre que les chapitres suivants ne survoleront que les parties importantes de l'API sans rentrer dans les détails de fonctionnement.

6.2.3.1 - Requêtes

Pour effectuer des actions côté serveur, il faut appeler le script *api.php* via une requête HTTP ou HTTPS. Les données sont alors transmises par méthode GET ou POST en se servant de l'attribut query. Ce dernier est constitué d'une chaine de caractères JSON contenant plusieurs attributs permettant d'informer le serveur de l'action à effectuer, ainsi que les éventuels paramètres qui lui sont attachés.

Voici un exemple de structure de query :

```
{
        "id"            : "...",
        "key"           : "...",
        "action"        : "...",
        "parameters"    : {
                ...
        }
}
```

Le serveur exécute alors l'action demandée (*action*) et envoi une réponse au format JSON contenant le statut de la requête (succès, échec ?), ainsi qu'un ensemble d'attributs en réaction à cette demande (par exemple, une liste de relations si l'on demande à consulter la liste des relations d'un objet).

L'API développée garantit aussi une certaine robustesse :

- Les erreurs sont identifiées (mauvais format, attributs manquants, injection de code), et l'API renvoie le numéro et un message concernant cette erreur.

- Chaque entité est limitée dans ses actions en fonction de ses attributions. Par exemple, un objet ne pourra pas accéder aux fonctions utilisateurs, ou encore ne pourra consulter librement les données publiées par un autre objet avec lequel il n'entretient aucune relation.

6.2.3.2 - Identification

L'identification est un point qui a soulevé quelques questions, auxquelles nous avons essayé de répondre :

Quels identifiants/clés existe-t-il déjà ?

Si un objet est connecté à internet, c'est qu'il possède surement une adresse MAC, soit un identifiant unique composé de 48bits (plus de 28 mille milliards de combinaisons possibles). Cette technologie semble donc tout indiquée pour distinguer les objets connectés au sein du réseau social, pourtant nous n'avons pas retenu cette solution pour plusieurs raisons :

- Un objet connecté n'utilise pas forcément TCP/IP pour avoir accès à internet, il se peut très bien qu'un objet communique de façon sans fil, sans se servir du WIFI (ex : fréquence 433Mhz), avec une station relais qui sert de point d'accès vers internet. Dans ce cas, seule la station aura une adresse MAC. C'est d'ailleurs le schéma classique que l'on retrouve actuellement dans la domotique. Le problème est donc double : d'un côté les objets n'ont pas d'identifiant, car pas d'adresse MAC, et de l'autre la station qui gère plusieurs objets ne peut avoir qu'un seul identifiant par carte réseau…

- 28 mille milliards de combinaisons est un chiffre incroyablement grand. Mais en 1981 l'IPv4 avec ses 32bits, soit 4 milliards de combinaisons, semblait impossible à saturer. Pourtant actuellement les adresses IPv4 disponibles sont en importante pénurie, il n'est donc pas inconcevable que dans quelques années, ce soit la même chose pour l'adresse MAC.

Nous avons donc vu les choses en grand :

Chaque entité possède un identifiant unique composé de 256bits, soit plus de 10^{77} combinaisons (l'univers visible est constitué d'environ 10^{80} atomes), garantissant une disponibilité absolue.

La clé privée permettant d'authentifier cette entité (équivalent du mot de passe) est elle aussi composée de 256bits pour éviter les attaques par force brute.

Cette solution a pourtant quelques désavantages : un identifiant unique impose d'avoir une autorité régulatrice qui alloue ces identifiants. D'autre part les constructeurs doivent implémenter pour chaque objet un identifiant dès sa fabrication. Nous pourrions alors imagier un service internet faisant office d'autorité de contrôle qui fournirait à la demande des clés d'identification par exemple à 1 centime de CHF l'unité pour éviter toute sorte d'abus.

Quelle sécurité lors de la transmission de ces informations ?

La sécurité ne fait pas partie intégrante de mon projet, elle a donc été mise de côté : chaque entité transmet à l'API son identifiant et sa clé privée tels quels, sans mécanisme particulier de protection. Seul un accès HTTPS a été mis en place, mais celui-ci n'est clairement pas suffisant. Il serait donc indispensable de revoir cet aspect dans une perspective d'amélioration future.

Dans le cadre de ce projet, la clé d'identification et la clé privée seront choisies de façon aléatoire et unique pour chaque entité.

Lorsqu'une entité veut effectuer une action côté serveur, elle doit dans la plupart des cas transmettre son identifiant et sa clé privée (*id* et *key* dans la section précédente) pour l'authentifier. Les utilisateurs ne font pas exception à la règle : ils doivent aussi s'authentifier via un identifiant et une clé 256bits. Par contre ceux-ci ont la possibilité d'utiliser l'action *authenticate* qui convertit le couple *email*/*password* en *id*/*key*.

6.2.3.3 - Enregistrement sur le réseau social

- **register_user :** permet à un utilisateur de s'enregistrer sur le réseau social via une adresse email et un mot de passe. Le mot de passe est haché puis sauvegardé dans la BDD MySQL interne. Le serveur génère alors un identifiant unique et une clé privée pour cet utilisateur qu'il renvoie comme réponse.

- **Register_object :** permet à un objet connecté de s'enregistrer sur le réseau social en transmettant son identifiant unique ainsi que sa clé privée.

6.2.3.4 - Notifications

Les notifications ont pour rôle d'informer une entité d'une action particulière. Elles sont générées uniquement par le serveur en réaction à un évènement.

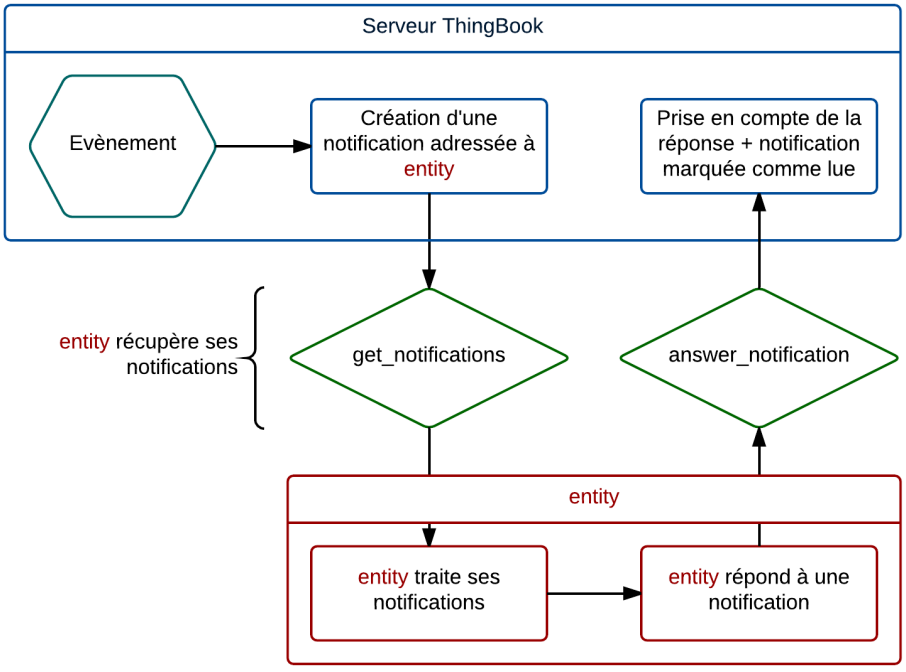

Figure 10 - Schéma montrant le processus partant de la création d'une notification, jusqu'à sa réponse.

Lors d'un évènement particulier (ex : un objet demande à être « ami » avec un autre objet), le serveur génère une notification. L'objet concerné, en consultant ses notifications via l'action *get_notifications*, peut choisir d'accepter ou de refuser cette offre. Il répond alors à cette notification en invoquant l'action *answer_notification*, plus un paramètre déterminant son choix (oui ou non je veux devenir « ami »). Le serveur marque alors la notification comme lue et agit en fonction de la réponse de l'objet.

Les notifications sont un point très important, notamment dans la gestion de transactions qui requiert le consentement de plusieurs entités, comme la demande de création de relations, ou encore la demande de changement de propriétaire.

6.2.3.5 - Actions utilisateur

- **authenticate** : comme vu précédemment, cette action sert à convertir le couple *email*/*password* en *id*/*key*, dans le but de simplifier la connexion des utilisateurs qui ne retiendront certainement pas par cœur 256bits…

- **list_owned_objects** : permet à un utilisateur de récupérer la liste de tous les objets qu'il possède. Cette liste contient entre autres l'identifiant et la clé de chaque objet, permettant ainsi à l'utilisateur de se connecter en tant qu'objet et donc d'avoir un contrôle sur son profil et les actions qu'il effectue.

6.2.3.6 - Actions objet connecté

Relations :

- **request_for_a_new_relationship** : permet à un objet de faire la demande d'une nouvelle relation avec un autre objet. Pour créer une nouvelle relation valide, le processus complet est le suivant :
    1. objet_1 demande à créer la relation « ami » avec objet_2 grâce à la commande *request_for_a_new_relationship*. Côté serveur, une notification concernant cette demande est émise à l'attention d'objet_2.
    2. objet_2 consulte ses notifications (*get_notifications*), puis répond à la demande de relation par un consentement ou un refus (*answer_notification*). Côté serveur, en cas d'acceptation, une nouvelle relation « ami » est créée entre objet_1 et objet_2.
    3. objet_1 et objet_2 sont notifiés qu'ils sont désormais liés par une nouvelle relation.

- **get_relationships** : liste l'ensemble des relations que possède un objet. Cette liste est consultable par n'importe quelle entité.

- **remove_relationship** : supprime tout simplement une relation entre deux objets.

Publications :

Lorsqu'un objet publie des données, il se peut que celles-ci aient un lien entre elles. Les publications servent donc à regrouper un ensemble de données qui seront alors consultées en bloc. Par exemple, une voiture connectée pourrait publier simultanément sa position et sa vitesse à un instant *t*, plutôt que d'envoyer ces données à la suite, sans savoir si elles sont liées entre elles ou non.

Chaque publication contient donc une ou plusieurs données, et chacune de ses données contient elle-même une valeur, un type (booléen, chaine de caractère, nombre ?), un ensemble de tags et finalement une liste de relations autorisées à la consulter. Par exemple, un objet ne pourra pas consulter cette donnée si celle-ci requiert la relation « ami », alors que celui-ci ne possède pas cette relation avec l'objet concerné.

- **post_publication :** permet à un objet d'envoyer une publication sur son propre profil ou sur celui d'un autre objet.

- **get_publication :** permet à un objet de récupérer une publication particulière. Les données visibles dépendront de la visibilité de celles-ci (en fonction des relations entre deux objets notamment).

- **get_publications :** récupère plusieurs publications provenant du profil d'un autre objet ou du sien. D'autre part, plusieurs paramètres peuvent être utilisés pour trier ces publications. Comme pour *get_publication*, la visibilité des données est limitée.

- **remove_publication :** supprime une publication.

Propriétaires :

- **request_for_a_new_owner :** permet à un objet de demander à changer de propriétaire.

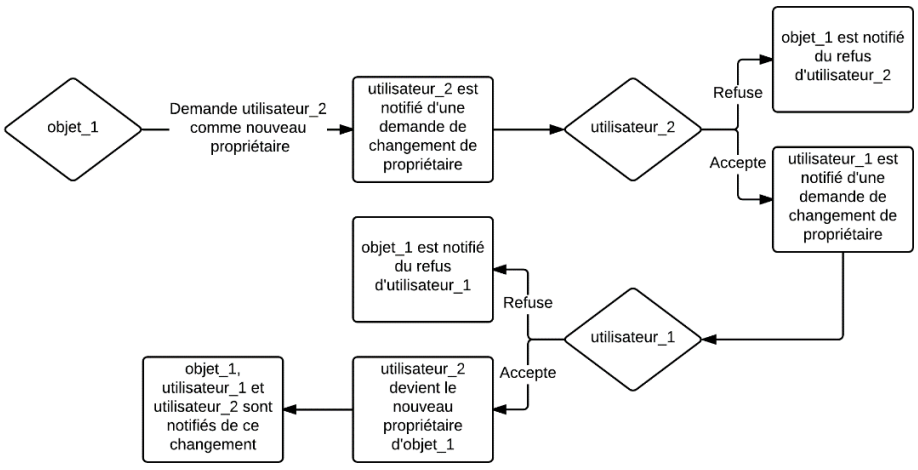

Figure 11 - Schéma montrant le processus de changement de propriétaire.

1. objet_1 a pour propriétaire actuel utilisateur_1, mais il souhaite changer pour utilisateur_2. Pour ce faire, il utilise la commande *request_for_a_new_owner*. Côté serveur, une notification concernant cette demande est émise à l'attention d'utilisateur_2.

2. utilisateur_2 consulte ses notifications (*get_notifications*), puis répond à la demande par un consentement ou un refus (*answer_notification*). Côté serveur, en cas d'acceptation, utilisateur_1 est notifié de la demande de changement de propriétaire.

3. utilisateur_1 consulte à son tour ses notifications, et accepte ou refuse de laisser objet_1 changer de propriétaire. S'il accepte, coté serveur, utilisateur_1 n'est désormais plus propriétaire d'objet_1 pour qui le nouveau propriétaire est utilisateur_2.

6.2.4 - Conclusion

L'API est vraiment le cœur et le point central de mon projet. Celle-ci permet aux objets connectés, aux utilisateurs et aux applications d'interagir entre eux, de publier des données, de créer des relations et bien plus encore. Bref, toutes les fonctionnalités d'un réseau social, accessibles non plus seulement aux êtres humains, mais aussi aux machines.

6.3 - Interface utilisateur

L'interface utilisateur réalisée en HTML, CSS, JavaScript et PHP, offre une interface bien plus intuitive pour des utilisateurs lambda, face à une API plutôt réservée aux objets ou aux développeurs.

Celle-ci permet à ses utilisateurs de consulter la liste des notifications qu'ils ont reçues (par exemple, la demande d'un objet qui souhaiterait faire de vous son nouveau propriétaire). D'autre part, elle permet aussi de consulter sa propre liste d'objets connectés et de visualiser leurs publications ainsi que les différentes relations qu'ils ont pu créer.

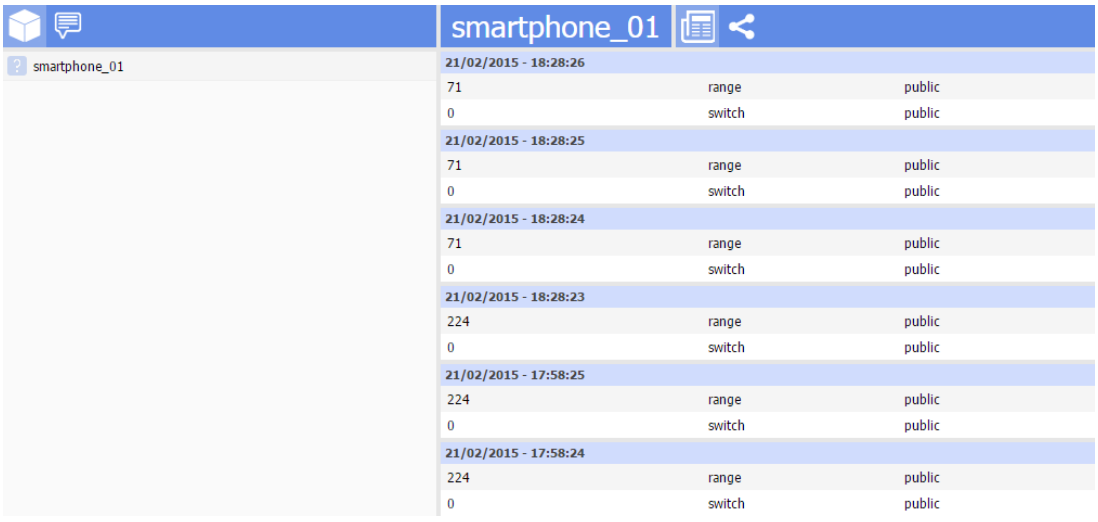

Figure 12 - Liste des dernières publications de l'objet « smartphone_01 ».

À chaque clic de l'utilisateur, cette interface effectue un appel AJAX, c'est-à-dire, une requête HTTP en JavaScript, vers l'API de ThingBook. C'est donc une application qui n'a pas besoin d'être située sur le même serveur que cette dernière, car elle ne fait pas directement appel aux bases de données ou aux fonctions PHP utilisées par l'API. Elle permet ainsi de prouver en partie le bon fonctionnement de l'API.

Cependant, cette interface n'a pas du tout le même rôle que les applications décrites dans le framework. En effet il s'agit ici d'une application qui simule les actions d'un utilisateur, fournissant un contrôle direct sur le compte de l'utilisateur, mais aussi de ses objets connectés. L'utilisateur doit donc lui fournir ses identifiants, ce qui pose d'importants problèmes de sécurité s'il s'agissait d'une application tierce. Les applications vues dans le framework sont de véritables acteurs publiant des données que différents objets peuvent ensuite consulter. Ils pourront alors réagir en conséquence, mais en aucun cas elles n'ont de contrôle direct sur le compte de ces objets.

7 - Évaluation et Résultats

Pour évaluer le prototype, plusieurs objets connectés ont été utilisés :

- Deux smartphones. Ce sont des objets désormais très répandus et avec lesquels on peut facilement interagir. L'un servira d'interface d'interaction entre l'utilisateur et les autres objets connectés, et le second réagira en fonction des données publiées par le premier.
- Une carte arduino Yun. Celle-ci embarque un microcontrôleur capable de gérer d'autres modules électroniques, et un processeur exécutant une version allégée de Linux, permettant d'établir une connexion internet. Cette carte a servi à la conception d'une prise électrique connectée.

Pour chacun de ces objets, une application spécifique a été développée. Celles-ci utilisent l'API de ThingBook pour créer différentes relations, publier des données ou encore les consulter.

La première a pour rôle de proposer une interface utilisateur basique, composée de boutons permettant d'effectuer différentes tâches les unes après les autres. Ces taches sont automatisables, mais ceci n'est pas souhaité dans le cas d'évaluation de ce prototype (nous ne verrions alors plus directement le couple action/réaction produit à chaque appel de l'API). Cette interface contient aussi un sélecteur de nombre (de 0 à 255) et un bouton on/off, qui constitueront à eux deux les sources de données publiées sur le réseau social par le premier smartphone.

La deuxième a pour simple rôle de changer de couleur : du noir au blanc. Cette couleur est déterminée par la valeur du sélecteur de nombre publiée par le premier smartphone.

La prise connectée, quant à elle, réagit en fonction de la position du bouton on/off.

Ces deux derniers objets sont autonomes : ils s'enregistrent seuls sur le réseau social, se lient automatiquement entre eux grâce à une relation « friend », et consultent les données du smartphone numéro un, pour changer leur état.

Dans ce schéma d'évaluation, il est important de souligner que les différents objets ne communiquent pas directement entre eux. L'ensemble des données produites par le premier smartphone est publié sur ThingBook, puis consulté au besoin par les autres appareils. En outre, au début de l'opération d'évaluation, aucun des objets n'est déjà préenregistré sur le réseau social ou ne possède de relations avec d'autres. Finalement, leur identifiant est généré aléatoirement au lancement de leur application, permettant ainsi de reproduire les tests sur n'importe quel nombre d'objets.

7.1 - Tâches

7.1.1 - Tâche 1 – Enregistrement sur le réseau social

1 - Le premier smartphone s'enregistre sur le réseau social sous le nom de « smartphone_01 ».
2 - smartphone_01 demande à l'utilisateur « user_01 » s'il veut devenir son propriétaire.
3 - Depuis l'interface utilisateur, user_01 consulte ses notifications et accepte alors de devenir propriétaire de smartphone_01.
4 - user_01 reçoit alors confirmation qu'il est bien le possesseur de cet objet, puis consulte sa liste d'objets pour vérifier que smartphone_01 s'y trouve bien.

7.1.2 - Tâche 2 – Création de relations

Entre smartphone_01 et smartphone_02 :

1 - Le deuxième smartphone s'enregistre sur le réseau social sous le nom de « smartphone_02 ».
2 - smartphone_02 demande alors à créer une relation « friend » avec smartphone_01.
3 - smartphone_01 consulte ses notifications et accepte cette relation.

Entre smartphone_01 et smartphone_02 :

4 - La prise connectée crée à son tour son compte sur le réseau social.
5 - Elle demande ensuite une relation « friend » avec smartphone_01.
6 - Comme dans le premier cas, smartphone_01 consulte ses notifications et accepte cette relation.

7 - Depuis l'interface utilisateur, user_01 consulte le profil de smartphone_01 et vérifie que smartphone_01 a bien deux nouvelles relations.

7.1.3 - Tâche 3 – Publication/Consultation de données

1 - smartphone_01 publie périodiquement sur son propre profil une valeur comprise entre 0 et 255, ainsi qu'un état « on » ou « off ». Ces deux valeurs sont choisies par l'utilisateur grâce à un sélecteur et à un bouton depuis l'interface du premier smartphone.
2 - smartphone_02 consulte ces données, puis change de couleur en fonction de la dernière valeur publiée par smartphone_01.
3 - La prise connectée consulte aussi le profil de smartphone_01 et s'allume ou s'éteint en réaction.

7.2 - Résultats

7.2.1 - Tâche 1 – Enregistrement sur le réseau social

En cliquant sur le bouton « s'enregistrer », le premier smartphone lance une requête *register_object* et se crée un compte.

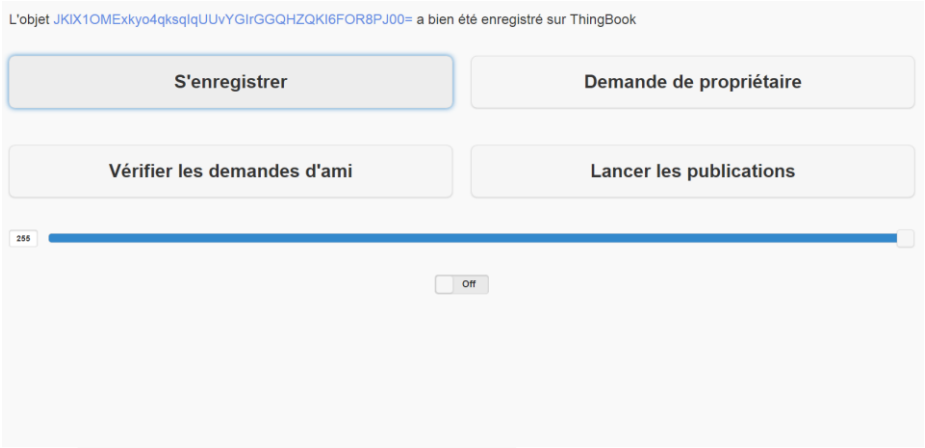

Figure 13 - Enregistrement de « smartphone_01 ».

Puis, en appuyant sur le bouton « demande de propriétaire », le smartphone lance une requête *request_for_a_new_owner*. Depuis l'interface utilisateur, on peut alors accepter cette demande et devenir ainsi le propriétaire de smartphone_01.

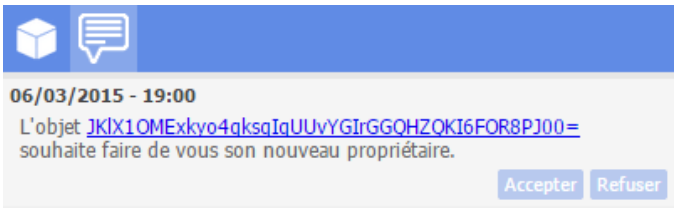

Figure 14 - Notification de demande de propriétaire.

7.2.2 - Tâche 2 – Création de relations

La demande de création d'une relation « friend » est accomplie de façon autonome par la prise connectée et le deuxième smartphone, grâce à la commande *request_for_a_new_relationship*. Il suffit alors de cliquer sur le bouton « Vérifier les demandes d'ami », depuis smartphone_01, pour que celui-ci accepte la relation. Ce dernier consulte alors ses notifications (via *get_notifications*), recherche d'éventuelles demandes d'ami et répond favorablement (*answer_notification*).

Le retour d'information se fait depuis l'interface utilisateur. On peut alors observer que smartphone_01 a créé des relations « friend » avec deux objets, qui sont le deuxième smartphone et la prise.

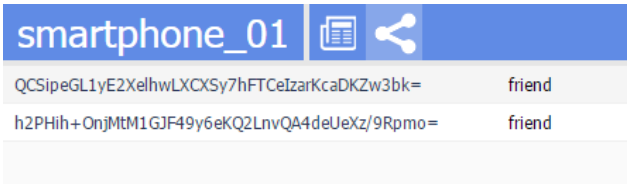

Figure 15 - Consultation des relations d'un objet.

7.2.3 - Tâche 3 – Publication/Consultation de données

En appuyant sur le bouton « Lancer les publications », smartphone_01 va publier périodiquement sur son propre profil la valeur de sa barre de sélection et l'état du bouton on/off.

Le deuxième smartphone va lire de son côté régulièrement le profil de smartphone_01. À partir de la dernière valeur publiée de la barre de sélection, il va alors changer de couleur : noir pour 0, en passant par gris, jusqu'à blanc avec 255.

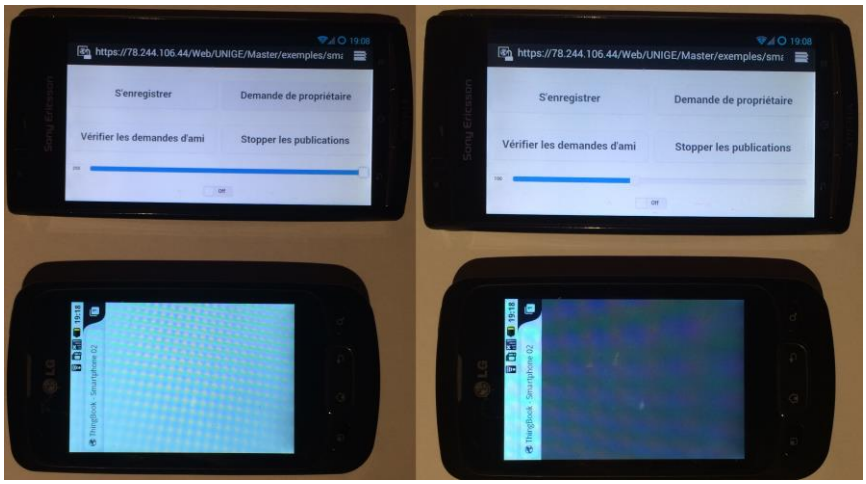

Figure 16 - smartphone_01 au-dessus, smartphone_02 en dessous - Changement de couleur de smartphone_02, en fonction des valeurs publiées par smartphone_01.

La prise connectée, elle aussi, réagit aux données postées par le premier smartphone et s'allume ou s'éteint en conséquence.

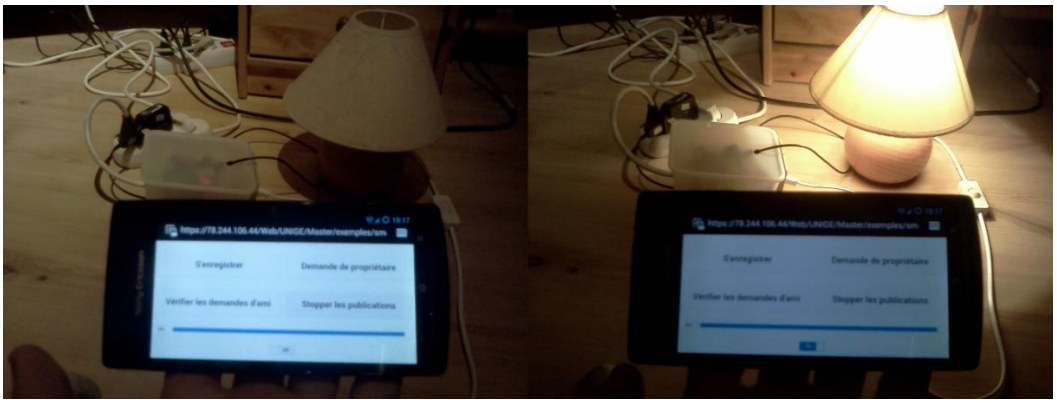

Figure 17 - La lampe reliée à la prise s'allume et s'éteint en fonction de l'état du bouton de smartphone_01.

Bien que simples et très basiques, ces applications prouvent le bon fonctionnement du service. Nous pourrions alors imaginer des programmes bien plus performants et évolués pour d'autres objets connectés, utilisant pleinement les possibilités qu'offre l'API. Par exemple, en se servant d'un réseau de relations entre objets, il serait possible d'en contrôler plusieurs à la fois et de les coordonner pour effectuer des actions complexes (ex. : réseau routier autogéré avec véhicules autonomes). Nous pourrions aussi voir apparaitre des objets indépendants capables d'agréger les données provenant d'autres objets proches pour en tirer des conclusions (ex. : des mini stations météo calculant individuellement une estimation du temps grâce à des capteurs de température, d'humidité ou encore de vent, disséminés un peu partout autour, dans un rayon de plusieurs kilomètres).

7.3 - Évaluations des performances

En plus d'évaluer le fonctionnement de ThingBook, nous avons décidé de tester ses performances pour étudier sa réaction en cas de forte affluence de requêtes. Nous avons utilisé le service Amazon EC2 qui propose des serveurs virtuels privés sous forme de cloud computing. Il est ainsi très facile de personnaliser le serveur et de changer ses capacités à la volée. Pour les tests suivants, nous avons utilisé une instance t2.micro, c'est-à-dire : un CPU cadencé à 2.5Ghz, 1Go de Ram, 16Go de SSD, une connexion internet 1Gb/s et comme OS Ubuntu server. L'installation de ThingBook est très simple, puisqu'il suffit d'installer apache, PHP 5, MySQL, puis de cloner le répertoire git contenant le code source [L17][L19]. Il ne reste plus qu'à modifier le fichier *config.php* pour rentrer vos propres identifiants.

Un script spécial, disponible sous le répertoire *exemples/scalability/*, teste la réponse du service face à 100 machines simulées en parallèle depuis un navigateur internet. Chacune de ces machines s'enregistre sur le réseau social, crée 3 relations, et publie une donnée. Ce sont donc en tout 100 inscriptions, 100 publications et 300 créations de relations qui sont demandées à l'API. Le temps total est soigneusement calculé pour chaque requête, puis groupé par type.

```
createAccounts took 9357ms
postPublications took 10398ms
createRelationships took 33493ms
```

Figure 18 - Temps totaux pour différents types de requêtes.

En étudiant ces données, on se rend compte qu'il faut environ 94ms pour s'enregistrer sur le réseau social, 104ms pour publier une donnée, et 111ms pour créer une relation (rappelons que pour créer une relation, il faut d'abord faire une demande à un objet, que celui-ci consulte ses notifications, puis qu'il l'accepte). La plupart de ces requêtes prennent donc environ 100ms. Si ce chiffre semble être élevé si l'on compte effectuer des centaines de milliers de requêtes à la seconde, n'oublions pas qu'une importante partie de ce temps est due à la connexion navigateur-serveur (obtenu par exemple via la commande *ping*), et surtout que l'API fait appel à des services externes (DynamoDB et GrapheneDB) pour le stockage des données et des relations. Bref dans ces circonstances, les services de cloud computing délocalisés offrent de plutôt bonnes performances en termes de rapidité, tout en supportant d'importantes montées en échelle.

Pour un prototype, les performances semblent tout à fait acceptables, et pourraient supporter environ 1000 connexions simultanées. Toutefois, ces résultats ne permettraient pas une mise en production réelle et à grande échelle. Les services de cloud computing externes offrent de nombreux avantages, mais ont aussi leurs limites. Ils ne seraient donc, par exemple, pas viables pour un service aussi important que Facebook (avec des milliards d'utilisateurs et certainement des millions de requêtes par secondes). De plus, ni l'infrastructure ni le langage de programmation ne permettent à l'heure actuelle une si forte montée en échelle.

7.4 - Analyse

En prenant un peu de recul, nous pouvons dire que le prototype est fonctionnel et se comporte comme un véritable réseau social pour les objets connectés. En fournissant aux développeurs une API permettant d'effectuer de nombreuses actions sur ThingBook, le service permet de créer plusieurs surcouches applicatives, à la fois pour des machines et pour des utilisateurs. De plus, le prototype est relativement simple à installer, est fourni avec une documentation du fonctionnement de l'API, et utilise des services externes pour la gestion des relations et des données, garantissant en théorie de pouvoir supporter une forte montée en échelle.

Toutefois, il a aussi quelques défauts. Comme nous l'avons vu juste au-dessus, les performances actuelles ne permettraient pas de gérer des millions d'objet et d'utilisateurs, notamment à cause du temps que mettent les requêtes à s'effectuer. C'est un souci que nous pourrions peut-être régler en testant les performances du prototype avec des bases de données NoSQL locales plutôt que décentralisées. De plus, ThingBook est peu accessible pour des utilisateurs lambda. En effet, même si une interface utilisateur a été développée, elle est loin de répondre à tous les besoins qu'ils pourraient avoir, surtout en ce qui concerne la gestion des autorisations. Le service est donc très tourné vers les développeurs, au détriment des utilisateurs non experts qui recherchent la simplicité avant tout avec un produit fini, esthétique et ergonomique. En outre, le prototype n'est pas suffisamment sécurisé vis-à-vis des données sensibles qu'il reçoit (mots de passes, données très personnelles, etc.). C'est donc une partie très importante, qu'il faudrait développer dans une perspective de travaux futurs.

Finalement, arrêtons-nous quelques instants sur le graphe de désirabilité, faisabilité et viabilité proposé par IDEO :

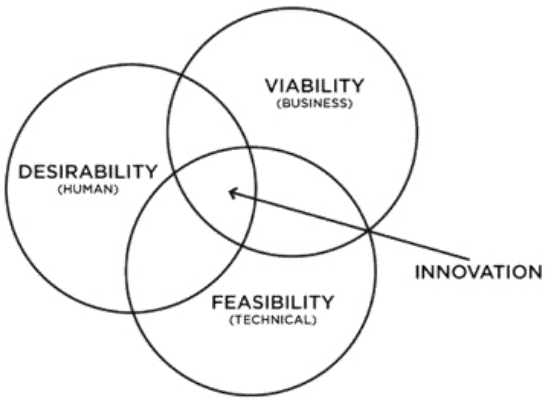

Figure 19 - IDEO desirability feasibility viability graph.

Dans ce mémoire, nous nous sommes principalement intéressés à la partie technologique, c'est-à-dire à la faisabilité d'un réseau social d'objet. En créant un prototype fonctionnel, nous avons prouvé que techniquement le concept est réalisable. À n'en pas douter, il y a certainement un modèle économique viable pour ce service et qui reste à être travaillé. Par exemple, en vendant ce service aux grandes entreprises, ou en proposant des applications capables de gérer plusieurs centaines d'objets et de les coordonner. Finalement, pour la partie désirabilité, nous pouvons dire que ThingBook est désirable et utile, car il uniformise la communication et les échanges entre objets connectés et êtres humains. Toutefois, comme vu précédemment, l'API est faite pour les développeurs et l'interface graphique ne couvre pas tous les besoins des utilisateurs lambda. De ce fait, pour ces derniers, nous pouvons dire que le prototype n'est pas encore désirable. C'est donc un point à travailler pour rendre le service vraiment attractif et utilisé massivement.

8 - Conclusion et Travaux futurs

8.1 - Résumé du travail accompli

À travers ce document, nous nous sommes demandés s'il était possible de concevoir un réseau social non pas pour les êtres humains, comme on en trouve beaucoup, mais pour les objets connectés. Un tel service simplifierait le développement de certaines applications, notamment en ce qui concerne la partie stockage des données, et l'enregistrement des objets. De plus, il ouvrirait de nombreuses perspectives d'avenir, comme nous l'avons vu dans la vidéo « Brad The Toaster », où les objets connectés sont autonomes, partagent leurs sentiments, et interagissent à la fois entre eux et avec des personnes.

Tout d'abord, nous avons fait un état des technologies déjà présentes, gravitant autour de la notion d'objets connectés et de réseau social. En constatant leurs limites, nous avons posé la problématique du réseau social d'objets connectés, n'existant tout simplement pas à l'heure actuelle.

Pour définir l'utilité de ce service, plusieurs scénarios distincts ont été imaginés, démontrant les avantages qu'il offre dans des perspectives d'avenir proches comme éloignées. Cela a permis d'identifier les différents utilisateurs (humains, machines ou programmes), leurs besoins et leurs rôles.

Nous nous sommes ensuite penchés sur la conception de son architecture. ThingBook s'adresse avant tout à des machines, de ce fait, les objets connectés se serviront d'une API pour interagir avec le réseau social. Cette dernière est une porte d'entrée pour effectuer des actions du côté du service, grâce à des requêtes au format JSON. D'un autre côté, nous avons vu que le nombre de ces objets va croître très fortement et de façon exponentielle, l'architecture doit donc supporter une puissante montée en échelle. Pour répondre à cette problématique, nous avons utilisé différents services de cloud computing pour les bases de données NoSQL, qui sont DynamoDB et GrapheneDB. Ces derniers furent d'ailleurs une expérience très enrichissante, puisqu'ils proposent des fonctionnalités tout à fait inédites par rapport à des bases de données traditionnelles.

L'architecture pensée, elle fût ensuite réalisée au travers d'un prototype, développé en PHP, un des langages les plus courants pour la conception d'applications web. En effectuant des requêtes HTTP, contenant des paramètres spécifiques, vers l'API de ThingBook, les objets connectés et les utilisateurs sont en mesure d'accomplir les mêmes actions que l'on retrouve sur un réseau social traditionnel. En plus de cette API documentée, une interface utilisateur est disponible, garantissant plus de simplicité pour ces derniers.

Finalement, nous avons imaginé plusieurs tâches pour mettre à l'épreuve le prototype. Pour ce faire, nous avons réalisé deux applications pour smartphones, ainsi qu'une destinée à une carte Arduino. Ces tâches exécutées sans peine, démontrent ainsi le bon le fonctionnement du service, et même si celles-ci sont rudimentaires, l'état actuel du prototype permet de réaliser des applications bien plus perfectionnées pour tous types d'objets connectés, ouvrant de très nombreuses perspectives d'avenir.

8.2 - Travaux futurs

Il reste encore quelques points à améliorer en ce qui concerne le prototype, et qui pourraient être développés dans un autre mémoire ou lors d'une mise à jour postérieure :

- **Ajouter plus de sécurité :**

  C'est un aspect sur lequel nous ne nous sommes pratiquement pas penchés, mais qui mériterait de s'y intéresser de très près pour une mise en service réelle. En effet, un tel service collecte des données très personnelles et parfois sensibles. Ces données devraient donc être stockées de façon encryptée pour ne pas être lisibles en cas de vol.

  Actuellement, seuls les mots de passe utilisateurs sont hachés. Il faudrait réfléchir à un dispositif similaire, mais déchiffrable pour les données brutes. De plus, même si l'échange des requêtes se fait par HTTPS, les mots de passe et identifiants sont transmis en clair. C'est un point très important à corriger en intégrant à l'API un mécanisme d'identification sans que les mots de passe aient à être transmis directement.

- **Créer un lien avec un réseau social pour les êtres humains :**

  Puisque plusieurs réseaux sociaux existent déjà, sont omniprésents et que Facebook est le plus utilisé, nous pourrions imaginer un service faisant le lien entre ThingBook et Facebook. À travers l'API Facebook, nous pourrions peut-être intégrer le réseau social humain au réseau social d'objets connectés et/ou inversement. Par exemple, nous pourrions nous poser les questions suivantes :

  - Qu'est-ce que cela apporterait ? Certainement une meilleure acceptation de ThingBook, ainsi qu'un échange inédit entre êtres humains et machines.

- Quels comportements pourrions-nous voir émerger ? Des êtres humains et machines communiquant directement entre eux, ou encore des objets qui s'adapteraient aux envies des utilisateurs en fonction des « like ».

Autant de questions qui pourraient être étudiées en profondeur dans de futurs travaux.

- **Intégrer véritablement les applications :**

Les applications sont prêtes à être insérées, mais ne sont pas activées dans le code source. En fait, comme celles-ci ont besoin des mêmes fonctionnalités que les objets connectés, elles peuvent pour l'instant s'enregistrer en tant qu'objet. Toutefois, leur nature diffère ainsi que leurs rôles, il serait donc judicieux de trouver une fonctionnalité qui les distingue vraiment pour les implémenter à part entière. C'est une mise à jour mineure, mais qui a tout de même son importance.

- **Joindre un langage de requêtes :**

Cette idée n'est pas tout à fait concrète, mais mérite que l'on s'y intéresse : dans l'état actuel des choses, il est possible d'effectuer des requêtes plutôt ciblées concernant les publications (ex. : par rapport à une date). Ces paramètres sont transmis en tant qu'attributs dans les requêtes JSON.

Pour perfectionner encore plus ce processus, il serait peut-être avantageux de penser à un langage de requêtes, par exemple très proche de SQL, capable d'effectuer des actions encore plus précises concernant les données publiées (ex. : recherche par tag). Ces requêtes pourraient alors aussi bien fonctionner pour un SELECT, un UPDATE, un INSERT ou un DELETE. Nous pourrions même imaginer un système qui marche pareillement pour les recherches de relations et capable de construire des graphes.

Références

Articles

[A01] - IBM Systems Journal, « The origins of ubiquitous computing research at PARC in the late 1980s », rédigé par Mark Weiser, publié en 1999.

[A02] - Scientific American, « The computer for the 21st century », rédigé par Mark Weiser, publié en 1991.

[A03] - RFID Journal « That 'Internet of Things' Thing », rédigé par Kévin Ashton, publié en 2009.

[A04] - « L'Internet des objets au service de la gestion énergétique des bâtiments», rédigé par Yann Bocchi, publié en 2013.

[A05] - BearingPoint, « Smart-Home : retour vers le futur ? », rédigé par Emmanuel Autier, publié en 2012.

[A06] - IEE Review, « The Connected Car », rédigé par Chris Evans-Pughe, publié en Janvier 2005.

[A07] - « Addicted Products », rédigé par Simone Rebaudengo, publié en 2012.

[A08] - Journal of Computer-Mediated Communication, « Social Network Sites: Definition, History, and Scholarship », rédigé par Danah M. boyd et Nicole B. Ellison, publié en 2008.

Liens

[L01] - « According to New Gartner Report, Manufacturers Will Have to Reinvent their Business Models to Profit from the Internet of Things », rédigé par Mathieu Baissac, publié le 4 février 2014, consulté le 13 août 2014, disponible sur http://blogs.flexerasoftware.com/ecm/2014/04/according-to-new-gartner-report-manufacturers-will-have-to-reinvent-their-business-models-to-profit-from-the-internet-of-thi.html.

[L02] - Gartner, « Gartner Says the Internet of Things Will Transform the Data Center », rédigé par Janessa Rivera et Rob Van der Meulen, publié le 19 mars 2014, consulté le 13 août 2014, disponible sur http://www.gartner.com/newsroom/id/2684616

[L03] - Le Parisien - « Las Vegas : la robe en fibre optique change de couleur à la demande », publié le publié le 9 janvier 2014, consulté le 2 août 2014, disponible sur http://www.leparisien.fr/high-tech/las-vegas-la-robe-en-fibre-optique-change-de-couleur-a-la-demande-09-01-2014-3477403.php.

[L04] - Science publique, « Pourquoi l'espérance de vie en bonne santé diminue-t-elle ? » (Émission télé), présentée par Michel Alberganti, diffusée le 31 mai 2013, consulté le 23 juillet 2014, disponible sur http://www.franceculture.fr/emission-science-publique-pourquoi-l%E2%80%99esperance-de-vie-en-bonne-sante-diminue-t-elle-2013-05-31.

[L05] - « Rupture de tendance : l'espérance de vie en bonne santé diminue pour la première fois. Dans quel état vivons-nous la grande vieillesse ? », rédigé par Christophe de Jaeger, publié le 23 juin 2013, consulté le 23 juillet 2014, disponible sur http://www.atlantico.fr/decryptage/rupture-tendance-esperance-vie-en-bonne-sante-diminue-pour-premiere-fois-dans-quel-etat-vivons-grande-vieillesse-frederic-jaeger-764665.html.

[L06] - « Les techniques interconnectées de Bosch au service de la voiture de demain », rédigé par Laurent Meillaud, publié le 29 avril 2013, consulté le 2 août 2014, disponible sur http://voitureduf futur.blogspot.ch/2013/04/les-techniques-interconnectees-de-bosch.html.

[L07] - « Addicted products: The story of Brad the Toaster», crée par Simone Rebaudengo, publié en 2012, consulté le 16 octobre 2014, disponible sur http://vimeo.com/41363473.

[L08] - « Gartner: How Will 26B IoT Units Affect Data Centers in 2020 ? », rédigé par James Sullivan, publié le 21 mars 2014, consulté le 8 août 2014, disponible sur http://www.tomsitpro.com/articles/internet-of-things-gartner-data-center-2020,1-1795.html.

[L09] - « The Internet of Things will force companies to rethink data-center operations », rédigé par Michael Kassner, publié le 16 avril 2014, consulté le 8 août 2014, disponible sur http://www.techrepublic.com/article/the-internet-of-things-will-force-companies-to-rethink-data-center-operations/.

[L10] - « How NoSQL will power the Internet of Things », rédigé par Adrian Bridgwater, publié le 9 janvier 2014, consulté le 25 septembre 2014, disponible sur http://www.computerweekly.com/blogs/open-source-insider/2014/01/how-nosql-will-power-internet-of-things.html.

[L11] - ThingSpeak, plateforme open data pour l'IoT, consulté le 24 octobre 2014, disponible sur https://thingspeak.com/.

[L12] - OpenIoT, middleware qui récupère automatiquement les données provenant de senseurs, consulté le 24 octobre 2014, disponible sur http://openiot.eu/.

[L13] - « OpenIoT - Phenonet, sensorised farming », disponible sur https://www.youtube.com/watch?v=9mbjVll79rM.

[L14] - « OrbiWise Poubelles Intelligentes», disponible sur http://vimeo.com/106790750.

[L15] - GrapheneDB, disponible sur http://www.graphenedb.com/.

[L16] - Amazon DynamoDB, disponible sur http://aws.amazon.com/fr/dynamodb/.

[L17] – Code source de ThingBook, disponible sur https://github.com/lifaon74/Web/.

[L18] – Documentation de l'api de ThingBook, disponible sur https://github.com/lifaon74/Web/blob/master/UNIGE/Master/docs/documentation.xml.

[L19] – Serveur EC2 hébergeant ThingBook, disponible sur http://52.16.38.94/Web/UNIGE/Master/.