GéoLoc Indoor

Projet RICM4 - Polytech' Grenoble

Imad ARRADA - Damien CRASTES
Quentin FAURE - Diana STOIAN

2015-2016

Sommaire

Présentation du projet

De nos jours, la géolocalisation est devenue une technologie très répandue. Elle est utilisée afin de guider un utilisateur d'un point A à un point B, de se retrouver en ville ou même afin de repérer les cafés ouverts dans les alentours. De fait, la géolocalisation est très répandue en extérieur, et c'est dans ce cas qu'elle est le plus efficace.

Mais, qu'en est-il en intérieur ? A cause de l'utilisation de satellites et du réseau GSM pour positionner un objet, la géolocalisation est moins performante en intérieur. Les bâtiments ralentissent, voir même dans certains cas, bloquent ces signaux et rendent l'utilisation de cette technologie très difficile. C'est là que le projet GeoLoc Indoor intervient !

Le but de ce projet est de permettre la localisation d'objet en intérieur. Par le biais de technologies plus adaptées à ce milieu, il est maintenant possible d'obtenir une localisation précise d'un objet !

En utilisant la technologie Bluetooth Low Energy, des objets sont repérés en intérieur grâce à des micro contrôleurs placés dans un bâtiment. Ces derniers font remonter des informations concernant les objets qu'ils voient passer à un serveur, qui va les traiter pour les placer sur une carte OpenStreetMap.

Matériel et librairies utilisés

Le matériel utilisé tout au long du projet est le suivant :

- Micro contrôleur STM32F401 avec un Shield Bluetooth Low Energy (connecté à un ordinateur pour joindre le serveur)
- TI SensorTag et iTag
- Serveur Amazon Cloud
- Terminal Android

Au niveau des librairies (et logiciels) :

- OSMDroid et AndroidStudio
- Spring MVC
- BLE API et X_NUCLEO_IDB0Xa1 sur MBed (pour les STM32)
- Paho (librairies MQTT)

Solution proposée

La solution proposée par le projet GéoLoc Indoor se base sur des échanges entre des micro contrôleurs (modèle STM32F401) et des balises (iTag, TI SensorTag) via la technologie Bluetooth 4.0, aussi dite Bluetooth Low Energy. Les micro contrôleurs, placés à des endroits prédéterminés, vont inspecter les alentours à la recherche de paquets Bluetooth. Ces informations

vont être remontées à un serveur distant afin d'être traitées, et la position de la balise sera calculée. Ensuite, la position sera stockée dans une base de donnée.

Par la suite, le rendu de la position d'une balise se fait via OpenStreetMap sur un client Android. Lors du lancement du client sur Android, l'utilisateur va choisir le flux de quelle balise recevoir. Les données de localisation d'une balise seront donc envoyées et placées sur une carte avec un rendu de l'intérieur des bâtiments.

Repérage d'une balise

Bluetooth Low Energy

Cette technologie est donc utilisée pour déterminer la distance entre un balise et un micro contrôleur. Deux modes sont utilisés dans le projet.

Central

Un central peut fonctionner de deux façons possibles :

- Soit il scanne lui meme les périphériques en émettant une requête de paquet puis les traitent.
- Soit il attend des paquets des périphériques et réagit à ceux ci.

Lors d'une demande de connexion, le scanne passe en second plan (il est souvent désactivé pour économiser l'énergie).

Périphérique

Un périphérique Bluetooth Low Energy n'émet que très peu de paquets bluetooth tant qu'il n'est pas connecté à un central. Il émet des paquets de type "advertising" qui contiennent son adresse MAC, quelques informations (comme son nom), et son type. Le type d'un périphérique repose sur 2 principes : si il accepte les connexions et si, le cas échéant, celles ci doivent venir d'un central particulier. Un périphérique qui se connecte à un central réduit énormément sa fréquence d'émission d'advertising, il peut même arrêter d'émettre pour des raisons d'économie d'énergie.

Utilisation du Bluetooth Low Energy

Nos périphériques sont de tous type (téléphone, micro contrôleurs, SensorTag TI, iTag, ou autre périphérique pouvant émettre en BLE). Comme nous voulons faire de la Géolocalisation en temps réel, il est nécéssaire de garder une fréquence de détection la plus élevée possible. Pour cela, il faut donc scanner uniquement les paquets d'advertising émis par nos périphériques. Nos centraux étant des STM32, il est préférable de réduire la consommation d'énergie, la recherche active n'est donc pas utilisée.

Détection et positionnement

Le bluetooth étant un système reposant sur des ondes à courte portée, il est nécéssaire d'avoir plusieurs STM32 servant de central au sein d'une même pièce. Les cartes fournissant la puissance du signal bluetooth reçu, il n'est pas nécéssaire de surcharger le nombre de balise : 3 suffisent. Lors de l'émission d'un paquet d'advertising par un périphérique, les cartes STM32 réagissent et transmettent alors au serveur (au travers d'un port série connecté à un ordinateur), leur identifiant, l'identifiant du périphérique détecté et la puissance du signal.

Le serveur connaissant la position des centraux, il est alors possible de calculer la position du périphérique grâce à un algorithme de centre de masse (ici la masse est égale à la force du signal). La puissance du signal ne dépendant que du périphérique émetteur, cela permet de conserver le même algorithme pour tous les périphériques et de conserver un calcul correct.

Information complémentaires

Il est possible d'utiliser tout type de central et non des STM32 tant qu'ils respectent la transformation de la puissance du signal et qu'ils émettent les données de manière formatée. Actuellement il est impossible d'utiliser des Arduino 101 car leur bibliothèque bluetooth ne prend pas encore en charge le mode central.

Enfin, comme seul le serveur connait la position des centraux, il est possible de les déplacer sans avoir à modifier le code du serveur ou du central, il suffit de modifier la position enregistrée dans la base de donnée pour que tout reste fonctionnel. Il est donc possible de rajouter des centraux pour augmenter la précision ou encore agrandir facilement le périmètre de recherche.

Traitement des informations

L'application qui représente le serveur du projet est faite en Java, en utilisant le framework Spring MVC. Son but principal est de calculer la position du dispositif, de la sauvegarder dans une base de données et de les envoyer après aux clients Android.

Le traitement des informations est réalisé par deux applications : une application « locale » qui va faire la connexion entre les cartes, et une autre application qui va être déployée dans le Cloud, sur une machine virtuelle Amazon.

Application dite locale

Pour faire la connexion entre les cartes et l'application, un objet de type SerialPort est utilisé (configuré avec des paramètres comme baud_rate, data_bits), ainsi qu'un EventListener, qui va écouter sur le port auquel chaque carte va être connectée. Les données sont récupérées dans des objets qui appartiennent à une classe Beacon, qui contient le nom du dispositif, l'id de la carte, la puissance du signal. Ces objets vont être envoyés après en utilisant les services Rest à une base de données qui réside sur la machine virtuelle Amazon.

Les données reçues, sous la forme d'une chaîne de caractères, doivent tout d'abord être parsées en délimitant les champs pour pouvoir créer l'objet Beacon. L'envoi est réalisé à l'aide d'un objet RestTemplate qui va faire un Post pour un objet de type Beacon vers l'url de la base de données,en spécifiant aussi la classe de l'objet envoyé. En conclusion, le but de cette application locale est juste de récupérer les données des cartes et de les transmettre via http à une autre application qui existe en ligne.

Application dans le Cloud

La deuxième application est celle qui va faire le calcul et qui va établir une connexion avec les clients Android, afin de pouvoir visualiser sur le téléphone la position d'un dispositif qu'on veut identifier.

Premièrement, les objets Beacon vont être récupérés depuis la base de donnée, et ces informations sont utilisées pour calculer la position. Pour ce faire, un endpoint dans lequel on spécifie l'id du dispositif visé est récupéré. La requête HTTP est une requête GET et renvoie au

format JSON le calcul final de la position(couple longitude/latitude) et d'autres informations, comme la date de la requête ou l'étage. La position va être sauvegardée après dans une autre table, qui va être utilisée pour maintenir un historique des positions du dispositif.

Pour accéder à la position, un autre endpoint, similaire au celui d'avant est disponible. Pour celui-ci, il est possible de choisir un nombre de résultats, au lieu de se contenter des 20 premiers résultats par défaut.

Calcul de la position

Le calcul concret de la position a été réalisé grâce à la formule suivante :

$$X_{obj} = \frac{\sum_{i=1}^{n} (signalStrength_i * X_{Stmi})}{\sum_{i=1}^{n} signalStrength_i} \qquad Y_{obj} = \frac{\sum_{i=1}^{n} (signalStrength_i * Y_{Stmi})}{\sum_{i=1}^{n} signalStrength_i}$$

Ici, signalStrength représente la puissance du signal, n le nombre de cartes STM utilisée pour le calcul, et Xstm et Ystm sont les positions des cartes qui ont identifié l'objet.

Pour trouver l'étage de l'objet, le même principe de calcul est appliqué. Afin de réussir à calculer la position, 3 cartes sont nécessaires, et des données considérées comme encore récentes[1]. Pour pouvoir modifier plus facilement la position des cartes STM, leur position est sauvegardée dans une table, et est récupérée au moment du calcul. L'ajout d'une position d'une carte est fait avec une requête Post, contenant un objet de type LocationSTM en format JSON à un certain endpoint.

En bref, pour identifier un objet, il faut :

- Récupérer les données de cartes qui vont passer par une application locale qui va les envoyer dans une application stockée sur une machine virtuelle Amazon
- Sauvegarder temporairement les données dans une base de données, qui se trouve aussi dans la machine virtuelle. Ces données vont soit être prises dans le calcul de la position et puis supprimées, soit elles vont être supprimées, en étant considérées comme pas assez récentes pour qu'on puisse calculer une position.
- Quand l'appel au service du calcul de la position est fait, la position est sauvegardée automatiquement dans une table de la meme base de données, où l'historique est sauvegardé.

Structure des applications

En ce qui concerne la structure des projets, l'architecture utilisée est Model View Controller. Ici le Model est représenté par les tables de la base de données (au total 3 tables : LocationHistory, Beacons, utilisée pour les données temporelles, et LocationSTM, la position des cartes STM). La partie View est la représentation JSON, au moment où une demande de position est faite, à un endpoint. Le Controller est celui qui fait appel aux services et celui dans lequel les endpoints sont

[1] Ceci est dû à la gestion des données avec deux bases de données : une récente, et une pour fournir un historique

créés, qui vont répondre avec certaines informations en format JSON(en ayant un Controller @RestController).

L'application a été construite avec des layers pour mieux séparer son fonctionnement :

- Le Web Layer, qui est représenté par le Controller et dans lequel est injectée une instance de LocationService, pour faire appel aux divers services.
- Le Service Layer dans lequel sont injectées des instances des Repositories pour pouvoir sauvegarder, supprimer ou chercher dans la base de données.
- Le Repository Layer, où se trouvent les interfaces qui étendent CrudRepositoryInterface afin qu'on puisse mettre en place des opérations Create Read Update Delete sur la base de données.

Technologies spécifiques au serveur

L'utilisation de Dependency Injection comme patron de conception permet un couplage plus faible entre les classes. L'injection d'objets dans les framework se fait avec l'annotation @Autowired, sans utiliser le mot clé « new ».

Spring Boot est aussi utilisé pour pouvoir configurer plus facilement une application Java Spring, surtout à leur création. Avec Spring Boot, il n'est plus nécessaire d'écrire manuellement les requêtes SQL, le framework créé automatiquement ces requêtes.

Quant à la sauvegarde et à la manipulation des données, PostgreSQL est utilisé. Pour la connexion avec les bases de données, c'est JPA et Hibernate qui interviennent. La configuration de la connexion est dans le fichier Properties, où est spécifiée l'url de la base de données à laquelle se connecter, le mot de passe pour y accéder et ainsi que d'autres informations.

Comme outil pour la gestion et l'automatisation de production des projets logiciels, Maven s'est révélé très pratique, en ajoutant dans le fichier pom.xml toutes les dépendances nécessaires pour le projet. Maven a permis de faire un build du projet et de créer les jars (destinés aux serveurs).

Envoi des informations

Principe de MQTT

Le mode d'envoi d'information choisi est le format MQTT. MQTT est un protocole de messagerie de type Publish/Subscribe. MQTT, signifiant littéralement Message Queuing Telemetry Transport, a été créé en 1999 dans le but de communications satellite.

Ses principales caractéristiques sont d'être un protocole très léger et de n'utiliser que très peu de bande passante. Ces qualités le rendent alors très pratique dans le cadre de communication M2M (machine to machine) ou bien IoT (Internet of things).

Modèle Publish/Subscribe

Le modèle Publish/Subscribe, communément appelé pub-sub, est le cœur de MQTT, en contraste avec HTTP dont l'architecture repose sur le paradigme request/response. Pub-Sub se découle selon des évènements et autorise les publications de messages vers des clients. Le tout se déroule autour d'un broker de message qui est chargé de délivrer tous les messages entre l'expéditeur et les bons destinataires.

Chaque client publie un message au broker, incluant un "topic" au message. Le topic représente l'information de routage pour le broker. Il faut savoir que chaque client souhaitant recevoir un message doit s'abonner (subscribe) à un certain topic. Le broker délivrera alors les messages à tous les clients abonnés à un topic donné. Ainsi les clients n'ont pas besoin de se connaître entre eux car ils peuvent communiquer à travers un topic.

On peut donc observer une très grande possibilité de mise à l'échelle des solutions permettant l'indépendance entre les producteurs de données et les consommateurs.

La différence avec HTTP est que le client n'a pas à envoyer de requête pour récupérer de l'information car le broker se charge de l'envoyer directement au client. De ce fait, chaque client MQTT dispose d'une connexion TCP ouverte permanente. Si la connexion est interrompue par quelconque moyen, les messages du buffer peuvent être stockés pour qu'il puisse les envoyer au client quand il se reconnectera.

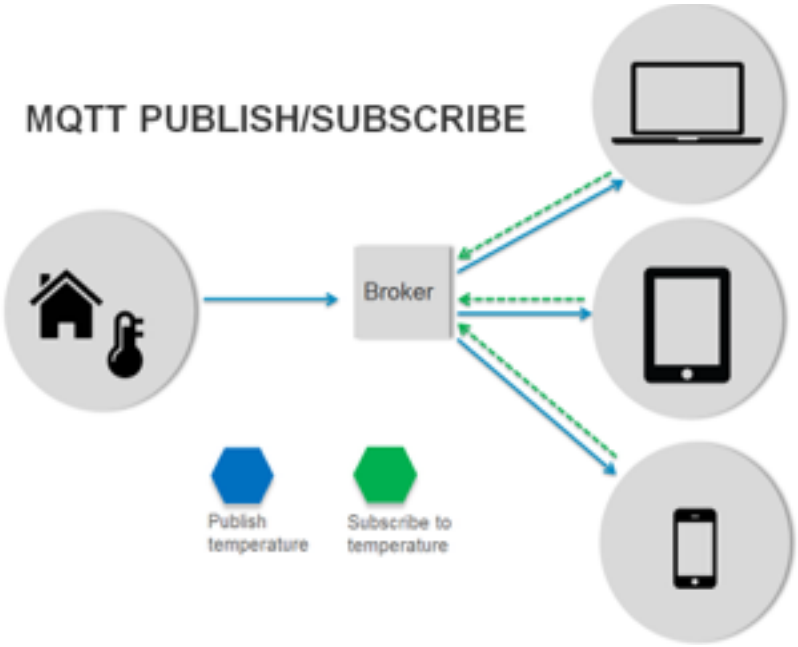

SCHÉMA DE FONCTIONNEMENT DE MQTT AVEC UN THERMOMÈTRE

Les topics ont une particularité très utile. Un topic est une chaîne de caractères pouvant avoir différents niveaux hiérarchiques grâce au caractère '/'. Il y a plusieurs avantages à cette caractéristique. Le premier est qu'un client peut s'abonner à exactement un topic, ou bien à une palette de topics. Le caractère '+' dans un topic désigne tous les sous-topics arbitrairement dans un même niveau hiérarchique. Pour désigner tout un sous-topic sur plusieurs niveaux (sous arbre jusqu'à la feuille), on utilise le caractère '#'.

Utilisation dans le projet Geoloc Indoor

Dans notre projet, ce mécanisme est repris par deux entités : le serveur amazon et les clients Android. Dans notre configuration, le serveur amazon publie à intervalle de temps régulier la position des balises. La position publiée comporte différents champs, comme l'identifiant de la balise, la date, la latitude *etc*… Les clients Android souhaitant connaître la dernière position mesurée d'une balise s'abonnent au topic associé à cette dernière. Pour cela, ces clients utilisent la fonctionnalité «subscribe» du protocole MQTT. Tous les clients abonnés recevront l'emplacement de la balise qu'ils suivent.

Par un souci d'unicité de chaque topic, nous avons décidé de donner l'adresse MAC de chaque balise comme nom de topic. Les adresses étant au format IPv6, il n'y aura pas de problème de multiplicité d'un flux.

Rendu

Une fois traitées, les données sont susceptibles d'être récupérées par un client Android. Pour ce faire, l'utilisateur dispose d'un écran de sélection afin de choisir quelle balise visualiser. Une fois sélectionnées, l'application s'abonne aux flux de chacune pour récupérer ses informations (subscribe MQTT). Si une balise est désélectionnée, l'abonnement au flux est résilié, et ses informations ne seront plus affichées sur la carte.

Le rendu sur une carte d'intérieur n'est pas encore fonctionnel. Des tuiles sont prêtes, réalisées en intégrant les plans du bâtiment de Polytech à des cartes OpenStreetMap, en se basant sur des coordonnées GPS récupérées avec des vues satellites. La dernière étape serait donc d'envoyer la carte à partir du serveur, au moment où l'application Android demande la liste des balises disponibles. Ensuite, chaque salle serait identifiée par une coordonnées GPS, qui permettrait donc de placer une balise dans une salle.

Note sur les cartes « indoor »

Les cartes indoors sont une face d'OpenStreetMap qui est encore en développement. Plusieurs solutions « non officielles » sont proposées, mais pour la plupart, elle ne sont pas maintenu ou alors non utilisable.

Certaines de ces API sont pourtant assez prometteuses, comme openlevelup.net, qui propose certains endroits assez complets. Cependant, la version web est assez lente, et les données ne sont pas exportables.

Une solution envisageable à l'heure actuelle serait, comme mentionné ci-dessus, de créer des tuiles personnalisées, qui serait stockées sur le serveur et envoyées au client. Les différents niveaux devraient être implémentés sous forme de couche, que l'on peut changer depuis l'interface du client selon ses besoins. De cette manière, chaque salle serait identifiée par ses coordonnées GPS, et une balise serait placée dans une salle avec ses coordonnées.

Un premier rendu d'une telle solution est disponible, et a été réalisé avec la version gratuite de MapTiler. Seulement, un outil comme celui-ci a ses limites : pour placer le plan d'intérieur, la solution la plus simple est de se référer à une carte ou des images satellites. Pour la première, les bâtiment de Polytech sont dessinés très grossièrement. Pour la seconde, les images satellites dans l'application ne permettent pas de placer correctement les points.

Afin de placer correctement le plan, nous avons donc récupéré les coordonnées GPS de certains points sur GoogleMaps, et les avons reportés sur le plan. Le rendu est déformé, dû au manque de points (une vingtaine n'a pas suffit pour la première moitié du deuxième étage). Or, les images satellites ne permettent pas de positionner plus de points précisément, car elles manquent de repères dû à la faible qualité des images et au toit du bâtiment.