Institut Universitaire Technologique de La Rochelle

Département Informatique

Stage de fin d'études

Émilien BARBAUD

Parcours informatique embarquée

Analyste-programmeur

Stage du 09/04/2018 au 22/06/2018

Panga

1 Rue Alexander Fleming, 17 000 La Rochelle

Enseignant tuteur : M. Régis NOHE

Maître de stage : M. Vincent MANZONI

Panga

Rapport consultable

Année Universitaire 2017-2018

Institut Universitaire Technologique de La Rochelle

Département Informatique

Stage de fin d'études

Émilien BARBAUD

Parcours informatique embarquée

Analyste-programmeur

Stage du 09/04/2018 au 22/06/2018

Panga

1 Rue Alexander Fleming, 17 000 La Rochelle

Enseignant tuteur : M. Régis NOHE

Maître de stage : M. Vincent MANZONI

Rapport consultable

Année Universitaire 2017-2018

Sommaire

Introduction

La technologie évolue sans cesse et s'invite de plus en plus dans nos maisons. Longtemps cantonnée aux ordinateurs, l'informatique se démocratise et équipe maintenant les télévisions, les voitures et même notre électroménager. L'informatique embarquée est un vaste domaine, mélange d'électronique et d'informatique, le tout dans un souci de miniaturisation et d'optimisation. C'est un secteur de l'informatique qui m'intéresse beaucoup. Le projet tutoré que nous avons effectué cette année avec la start-up Panga m'a donné l'opportunité de faire mon stage dans cette entreprise, dont l'activité est axée dans ce domaine.

Panga est une start-up située à La Rochelle dans le quartier des Minimes, spécialisée dans la domotique. Elle a d'ailleurs développé sa propre solution intitulée NeHo pour les habitants et NeMa pour les gérants. Cette solution est destinée à équiper autant des résidences que des écoles ou des bureaux. Elle est adaptative. Leur architecture permet de suivre en temps réel l'état général des bâtiments et des équipements collectifs, par les gérants mais aussi par les habitants, à la clé : des économies d'énergies, la création de liens sociaux entre usagers et la formation d'une « carte d'identité » vivante des bâtiments. La société Panga a été fondée par Patrick SIMON et Jean-Jacques ORLEMANS. On y travaille dans l'esprit start-up avec un open-space et un effectif de moins de dix employés. Nous sommes six stagiaires, dont un travaillant en collaboration avec moi, bien que nous disposons chacun de tâches spécifiques.

Pour développer, nous utilisons nos ordinateurs personnels. Dans mon cas, cela m'arrange car j'ai déjà travaillé pour Panga pendant le projet d'entreprise, je peux donc le continuer sur son support d'origine. Nous avons à disposition une connexion internet WiFi et filaire, un GIT et une boîte mail interne à l'entreprise pour communiquer.

Pour ma part, mon environnement de travail est Linux (Debian). Je développe sous Qt5 avec QtCreator. Je me sert de plusieurs bibliothèques, soit libres de droit, soit directement développées par Panga. J'ai à ma disposition un Raspberry Pi 3, qui me sert à tester la solution que je développe, étant donné que le programme final s'exécutera sur une architecture semblable (type ARM). Je fais de la compilation croisée sur celui ci. Je me sers également d'un Broker MQTT Mosquitto et d'une base de données ElasticSearch[1] pour faire communiquer mon application.[2]

1 Système de base de données. Site web : info.elastic.co/
2 Pour plus de précisions sur les technologies utilisées, se reporter à la partie du rapport qui y est consacrée.

Actuellement, Panga ne dispose pas de solution viable pour s'assurer du bon fonctionnement de leur matériel installé chez leurs clients. Mon rôle est de développer une solution de monitoring à distance pour les équipements déployés, qui permettra de visualiser à distance toutes les informations nécessaires au diagnostic de pannes, et si besoin est, lancer une intervention de maintenance. Je m'occupe du cœur du programme, tout ce qui est côté serveur, ce qui sera déployé dans l'architecture, en somme : le côté invisible pour les utilisateurs mais qui permet un bon fonctionnement de la solution. Mon collègue stagiaire s'occupe lui de toute la partie graphique côté client. Il est chargé d'afficher et de mettre en forme les données que mon application récupère chez les clients. Panga dispose déjà de quelques logements « pilotes [3]» où est déployée leur architecture, cependant elle ne compte pas s'arrêter là, il faut convaincre tous les potentiels clients de l'utilité et de la fiabilité de la solution Panga, d'où l'importance de ce logiciel de monitoring.

Pendant mon stage, j'ai pu compter sur Vincent MANZONI, mon maître de stage et sur Jérôme DAVID, chef de projet, pour m'apporter une expertise supplémentaire et des conseils pertinents. Vincent est très pédagogue et ayant été élève à l'IUT, il a su au mieux m'orienter pour trouver une solution à mes problèmes.

Le développement de mon rapport commencera par une présentation de l'entreprise, présentera le projet en détail, puis approfondira les technologies utilisées. Le rapport explicitera ensuite les difficultés que j'ai rencontré et expliquera comment je les ai surmonté. Finalement le planning prévisionnel, suivi du planning réel sera présenté.

3 La résidence « Vibratô » à La Rochelle : http://www.eden-promotion.fr/g35-vibrato-la-rochelle.html

I. Présentation de l'entreprise

La société Panga a été fondée par Patrick SIMON et Jean-Jacques ORLEMANS, on y travaille dans l'esprit start-up avec un open-space et un effectif de moins de dix employés.

Cette start-up située à La Rochelle dans le quartier des Minimes, est spécialisée dans la domotique. Sa propre solution (NeHo pour les habitants et NeMa pour les gérants) a été conçue pour équiper des résidences, des écoles ou des bureaux : elle est adaptative.

1. Le personnel

S'agissant d'une start-up, l'effectif est relativement restreint. L'entreprise accueille moins de dix salariés. On y trouve principalement des métiers assez classiques du domaine de l'informatique. On peut noter la présence d'une graphiste se chargeant à la fois de la partie communication de l'entreprise, et aussi du design de toutes les interfaces graphiques.

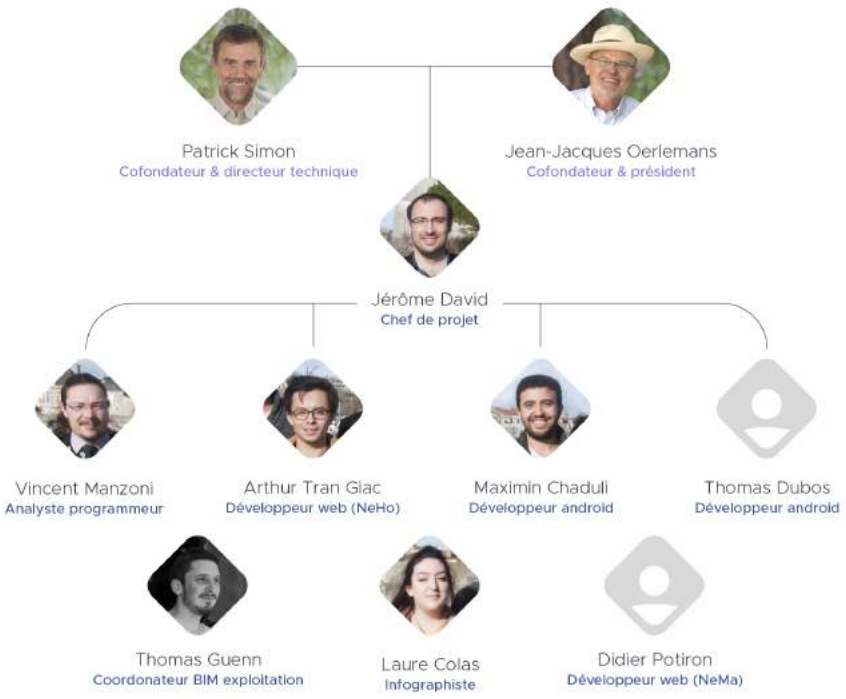


Illustration 1: Organigramme de l'entreprise

Notons aussi que l'on a eu l'occasion d'assister au départ de M Didier POTIRON, qui a eu une opportunité dans une entreprise d'une autre ville.

À la tête de l'entreprise on retrouve M Patrick SIMON, qui gère le développement de la start-up. Il rencontre les potentiels clients et investisseurs et présente la solution Panga lors des salons. On trouve aussi M Jean-Jacques ORLEMANS qui gère les aspects financiers et juridiques de l'entreprise.

2. Les locaux

Panga est logée dans un bâtiment type « hôtel d'entreprise » nommé Créatio®ImagéTIC[4], cofinancé par la Communauté d'Agglomération de La Rochelle et par l'Europe. Le bâtiment dispose de nombreux locaux de taille variable, et de salles communes de travail, pour les réunions par exemple.

Dans le local réservé à Panga, on trouve une salle principale dédiée à l'open-space, et une petite salle secondaire où sont installés tous les équipements de Panga, comme les routeurs et les serveurs. M Patrick SIMON nous a gentiment mis à disposition une bouilloire, ainsi qu'un four micro-ondes et un réfrigérateur. Précisons aussi que la salle de travail est climatisée, confort non négligeable lorsque les beaux jours arrivent, dans une salle où des dizaines d'ordinateurs tournent à plein régime.

L'open-space constitue selon moi un avantage de taille, on peut facilement échanger avec ses collègues et cela facilite l'entraide. Ajoutons que l'ambiance y est plus détendue ainsi, et facilite le dialogue, ce qui, dans une boîte de développement informatique est primordial.

3. La solution Panga

Leur architecture permet de suivre en temps réel l'état général des bâtiments et des équipements collectifs, par les gérants mais aussi par les habitants. Cette solution fonctionne avec plusieurs composantes que voici :

a) NeHo

NeHo est l'interface utilisateur intégrée au sein des résidences ou des bureaux, permettant le contrôle de l'habitat, la gestion des consommations énergétiques et des espaces communs. Elle permet également d'utiliser un réseau collaboratif de proximité via une messagerie. NeHo prend la forme d'une tablette contenant une vue sur le logiciel lui-même.

4 Site web : https://eco.agglo-larochelle.fr/creatio-imagetic

NeHo permet la réappropriation des données du bâtiment par les usagers, afin de les sensibiliser. Les habitants peuvent donc échanger et réagir par rapport aux données que leur fournit le terminal présent dans leur habitat. (cf. Annexe 1)

b) NeMa

NeMa est une plateforme de service de gestion permettant aux gestionnaires (Bailleurs, Syndics) de superviser à distance les bâtiments, de gérer les espaces, de communiquer avec leurs occupants et de faciliter la maintenance et les réparations. Elle permet également aux résidents d'avoir accès à différents services via NeHo. NeMa prend la forme d'une plateforme SaaS[5] plus classique. (cf. Annexe 1)

c) Security Concentrator Module (SCM)

Ils permettent de faire communiquer les différents éléments. Ils sont définis comme des « passerelles de communication applicative ». Ils concentrent, collectent et réalisent les premiers traitements sur les données issues des capteurs et actionneurs d'un bâtiment et/ou d'un quartier.

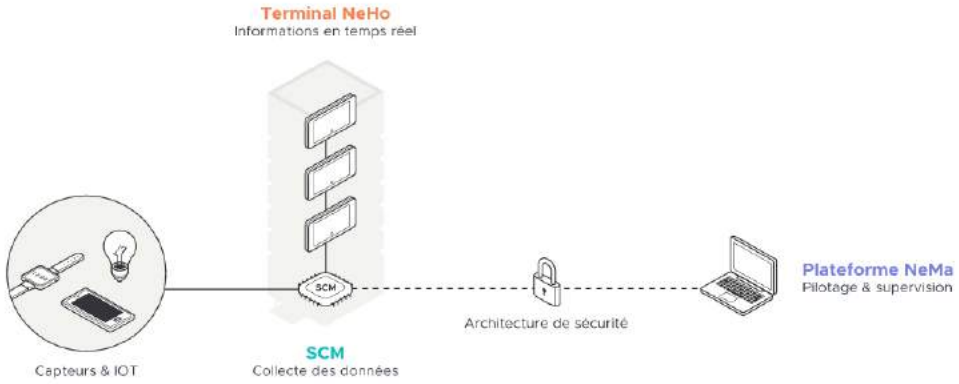


Illustration 2: Architecture principale

II. Présentation du projet

Pour installer son système chez les usagers, Panga utilise divers micro-ordinateurs sous Linux qui sont implantés dans les logements. Ce sont en grande partie les SCMs présentés dans la partie précédente. Cependant, en cas de ralentissement ou de panne, Panga ne dispose d'aucun moyen viable

5 Software as a Service : le logiciel est déporté sur un serveur et on y accède via un navigateur internet.

pour diagnostiquer le problème : tout ce qu'ils peuvent faire est de se connecter à distance en SSH[6], et chercher manuellement la cause de l'erreur.

Mon rôle est alors de développer un outil de supervision (monitoring) afin que Panga puisse faire le diagnostic de leurs appareils à distance. Cet outil doit être capable de collecter toutes les informations nécessaires pour juger du bon fonctionnement du système déployé par Panga. Il est important de comprendre que ce projet ne concerne que la partie supervision du système et non la collecte des données provenant des différents capteurs présents dans l'habitation. Un tel projet étant assez conséquent, je vais m'occuper uniquement de la couche basse. Certains de mes collègues stagiaires (notamment Loïc PONCHAUX) se chargeront des couches plus hautes, comme les IHM[7] et l'archivage de données. Mes tâches sont donc de récupérer les données utiles à la supervision sur les micro-ordinateurs, de les sérialiser, et enfin de les envoyer sur un Broker déjà présent dans l'architecture Panga. Travaillant sur de l'informatique embarquée, il y a des exigences toutes particulières sur l'optimisation du code, afin d'utiliser le moins de ressources possibles, précision d'autant plus importante qu'il s'agit d'un outil de supervision.

III. Technologies utilisées et implications dans le projet

Durant mon stage, je suis amené à utiliser diverses technologies dont je dresse ici la liste exhaustive, tout en vous expliquant en quoi elles m'ont été utiles :

1. Compilation croisée

La compilation croisée (ou cross-compilation) est au cœur de mon projet. Étant donné que je développe une application destinée à un appareil externe, je ne peux pas faire le développement sur celui-ci. La solution est donc programmer sur un ordinateur, pour ensuite compiler le projet pour une autre architecture.

Je développe alors mon programme sur mon PC, et je l'exporte vers une architecture ARM pour pouvoir tester mon programme sur un RPi3[8]. Qt Creator a grandement facilité les choses car il permet d'ajouter un périphérique distant au projet. Je n'ai alors qu'à compiler mon programme et il est

6 Secure Shell : c'est à la fois un programme informatique et un protocole de communication sécurisé.
7 IHM : Interfaces Homme Machine, désigne l'ensemble visuel qui permet d'interagir avec le programme.
8 Raspberry Pi 3 : nano-ordinateur monocarte à processeur ARM

automatiquement envoyé sur le RPi3 une fois la compilation terminée. J'ai le retour visuel du programme directement sur Qt Creator via SSH[9].

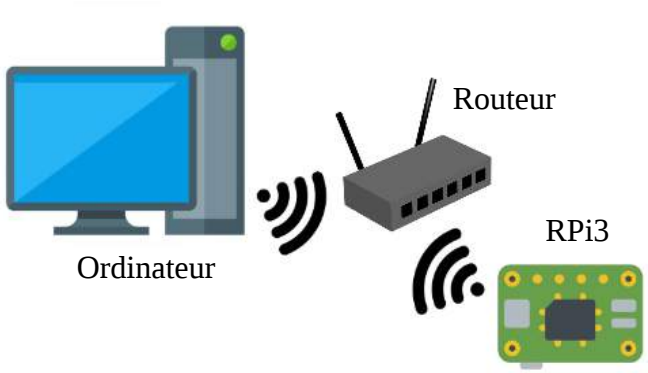


Illustration 3: Topologie de mon environnement de développement

La compilation croisée est cependant compliquée à mettre en place et ne facilite pas l'utilisation de bibliothèques. En effet, pour chaque bibliothèque utilisée, il faut la compiler pour une architecture ARM (le RPi3) depuis mon PC (architecture x86). Ceci peut être assez chronophage à faire car les créateurs ne documentent pas souvent la compilation croisée, bien que la méthode soit relativement semblable et universelle, il y a toujours quelques spécificités en fonction de notre architecture et des bibliothèques en question.

Le compilateur pour les versions ARM se nomme Gnueabihf. Le « hf » signifiant « hard float » car le RPi3 gère les nombres à virgule directement de façon matérielle ce qui n'est pas le cas de tous les processeurs.

2. SNMP

Le Simple Network Management Protocol (protocole simple de gestion de réseau) est un protocole de communication permettant de gérer les équipements d'un réseau, et d'en diagnostiquer les problèmes.

Ce protocole fonctionne avec un « manager » qui interroge des « agents ». Par exemple, le manager peut être un ordinateur et l'agent une imprimante. L'ordinateur va alors interroger l'imprimante pour connaître son niveau d'encre.

9 Secure Shell : c'est à la fois un programme informatique et un protocole de communication sécurisé.

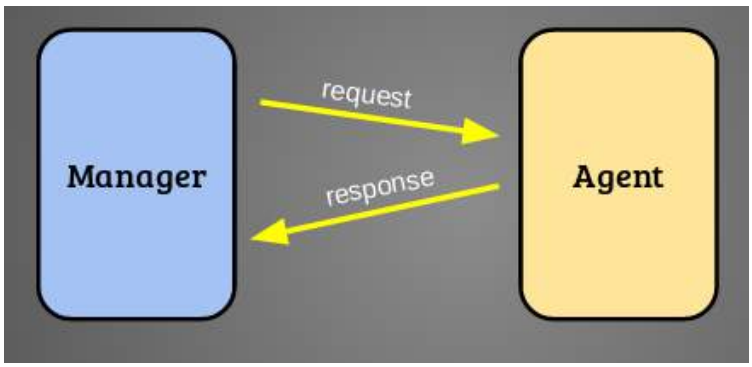


Illustration 4: Requête SNMP simple

Le SNMP se base sur des « Management Information Base » ou MIBs. Une MIB est un ensemble d'informations structurées sur une entité réseau sous forme d'un arbre. On s'y repère alors grâce au numéro que porte chaque nœud, on a donc une suite de chiffres constituant l'adresse de l'objet, appellée OID (Object IDentifier) ou identificateur d'objet.

En voici un exemple : .1.3.6.1.2.1

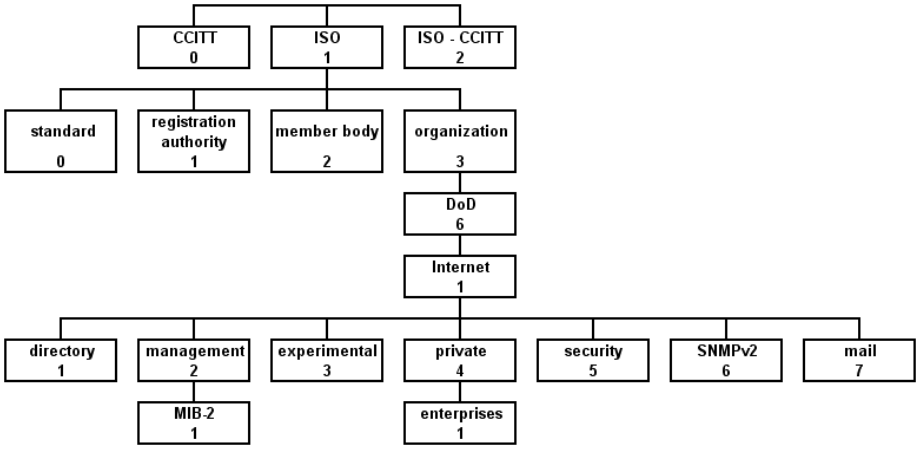


Illustration 5: Structure d'une MIB

Les MIBs présentent un gros avantage : elles sont standards et s'adaptent à tout type de matériel. Autrement dit, si l'on reprend l'exemple de l'imprimante et du niveau d'encre : peu importe l'imprimante, l'information sur le niveau d'encre devrait être situé au même endroit dans la MIB. C'est pour cette raison, le programme que je développe utilise cette technologie : peu importe le micro-ordinateur utilisé par Panga, les OIDs devraient être les mêmes pour chaque information que je récupère.

Pour mon projet, on se retrouve alors avec une topologie un peu plus atypique. Nous cherchons toujours à faire du monitoring sur du matériel, cependant il faut que se soit ce même matériel qui

récupère ses propres informations. Il faut alors envoyer des requêtes SNMP en local, pour que l'appareil « s'auto-requête ». L'appareil à surveiller jouera alors à la fois le rôle du manager et de l'agent.

Illustration 6: Topologie SNMP du projet

Comme vous l'aurez compris, les requêtes SNMP se font avec des OIDs. Pour trouver ses OIDs dans les MIBs, je me sers de la requête SNMP Walk. Elle permet de parcourir toutes les branches filles d'une adresse jusqu'aux feuilles. Une MIB peut contenir plusieurs milliers d'OIDs, c'est pourquoi la recherche d'une information précise peut être délicate.

On peut distinguer deux types de requêtes dont on aura besoin :

a) Requêtes SNMP simples

Ce sont les requêtes SNMP classiques, on interroge une adresse feuille, qui nous renvoie une valeur. Ces requêtes sont utilisées par exemple pour obtenir la température du CPU. On se retrouve alors avec une requête et une réponse, par exemple 48°C.

b) Requêtes SNMP multiples

Parfois il est nécessaire qu'une requête nous renvoie plusieurs valeurs. On peut prendre comme exemple la liste des processus en cours. Chaque processus va générer un OID différent, qu'on ne peut pas connaître à l'avance. Il faut alors lancer une requête SNMP Walk pour parcourir l'arbre afin de

trouver toutes les feuilles correspondantes. Ici la liste de tous les processus. La méthode correspondante dans la bibliothèque se nomme « get_bulk() ».

3. SNMP++

Afin d'utiliser le SNMP dans mon programme, je dois recourir à une bibliothèque. Il en existe deux qui dominent le marché : NetSNMP et SNMP++. La bibliothèque NetSNMP étant programmée en C et SNMP++ en C++, pour des raisons de simplicité nous avons décidé d'utiliser cette dernière.

SNMP++ est une bibliothèque développée par HP[10] dans les années 2000. Bien qu'elle ne soit pas de toute dernière génération, elle a eu quelques correctifs et mises à jour, et fonctionne correctement. La documentation n'est pas très fournie, et se limite aux commentaires Doxygen[11].

Un exemple d'une application simple est inclus, visant à faire des requêtes SNMP en fonction de champs textes que l'utilisateur peut renseigner avec des OIDs. Du fait du manque de documentation, cet exemple m'a été très utile, bien que certaines méthodes présentes étaient dépréciées depuis sa création. Il a alors fallu chercher des substitutions.

Cette bibliothèque a été conçue avec un système pour rendre les requêtes asynchrones. Il a fallu l'adapter pour qu'elle fonctionne correctement avec le système d'évènements de Qt : les signaux et les slots, dont je décris le fonctionnement plus bas.

Les performances de la bibliothèque sont plutôt bonnes, on peut faire des centaines de requêtes asynchrones en quelques millisecondes avec une consommation CPU minime. En somme, l'outil parfait pour créer un logiciel de monitoring.

4. Protobuf

Une fois les données récupérées sur le micro-ordinateur, il faut les envoyer. Or envoyer des données brutes et non structurées ne présente que peu d'intérêt. Je dois donc trouver un moyen de les sérialiser.

Le choix de l'outil de sérialisation s'est porté sur Protobuf, développé par Google, et ce pour plusieurs raisons :

10 Site web de Hewlett Packard : https://www8.hp.com/fr/fr/home.html
11 Outil de génération de documentation, site web : www.doxygen.org/

- Protobuf est multi-plateforme, cela veut dire que mes collègues travaillant sur la réception des données n'auront pas de contraintes au niveau du langage de développement.

- Protobuf compresse les messages. Les variables déclarées dans le fichier « .proto » sont fortement typées, et Protobuf effectue une sérialisation binaire, il compresse donc les messages lors de la sérialisation. On peut le comparer au format de données JSON, lui aussi très utilisé pour transporter des données structurées : avec Protobuf le message est plus petit (-35 % en moyenne) mais on perd le côté formaté et facilement lisible par l'homme du JSON. Dans notre cas, les données ne sont pas prévues pour être lues par un humain, donc Protobuf est avantageux. Cependant, étant donné que la plupart des données seront des chaînes de caractères, la compression ne sera pas optimale. En effet il est difficile de compresser une chaîne de caractère étant donné que n'importe quelle lettre peut suivre une autre.

- Protobuf est simple d'utilisation, il suffit de créer un fichier « .proto » pour renseigner la structure des informations à envoyer, et Protobuf se charge de nous créer le code source (un « .hpp » et un « .cpp ») avec toutes les classes et les méthodes nécessaires à la sérialisation. (cf. Annexe 6)

- Protobuf est documenté. Google fournit une documentation assez exhaustive pour cet outil, ce qui représente un gain de temps conséquent pour la prise en main.

- Protobuf n'est pas utile que pour la sérialisation, mais gère aussi toute la structure des données dans le code. Ainsi, pas besoin de créer de classes supplémentaires pour gérer le stockage dans la mémoire des données récupérées, étant donné que l'on peut avoir un objet Protobuf qui les contient.

- Protobuf est gratuit, même pour une utilisation commerciale et des projets professionnels, ce qui est assez rare pour le souligner, et est un avantage non négligeable financièrement pour une start-up.

```
message SearchRequest {
  required string query = 1;
  optional int32 page_number = 2;
  optional int32 result_per_page = 3;
}
```

Illustration 7: Exemple de fichier .proto pour Protobuf

Le fichier « .proto » est ensuite transformé par un outil nommé « protoc » qui se charge de créer le code source. Cet outil est aussi développé par Google et est inclus dans Protobuf. (cf. Annexe 5)

Au niveau de la sécurité il est important de noter que Protobuf ne chiffre pas les messages en les sérialisant. D'autant plus que nous sérialisons quasiment que des chaînes de caractères, les messages sont en majorité lisibles par un humain qui intercepte la communication. Cependant, ces messages ne constituant pas des données sensibles mais bien des données très techniques, et les messages étant envoyés sur le réseau local, un chiffrement des données ne s'impose pas à ce niveau. (cf. Annexe 7)

5. MQTT

MQTT (MQ Telemetry Transport) est un protocole majeur dans l'internet des objets. Il permet d'envoyer des informations sur un sujet donné à un serveur qui fonctionne comme un broker de message. Le broker pousse les informations sur le réseau, vers les clients qui se sont précédemment abonnés au sujet.

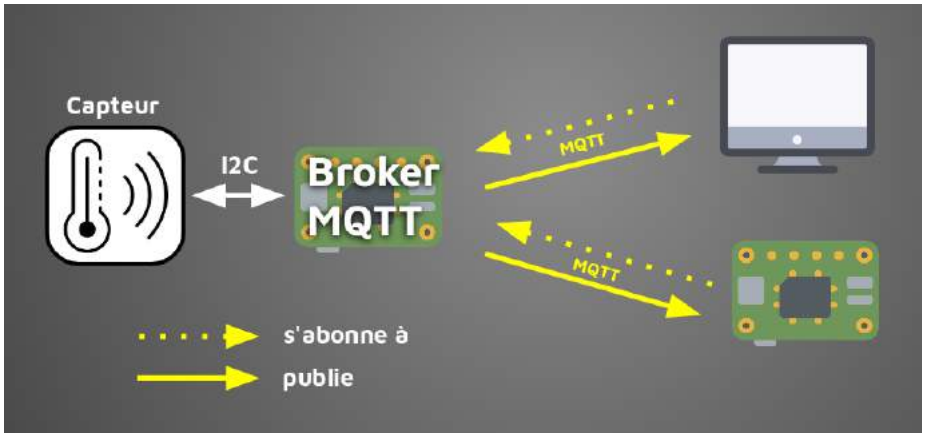


Illustration 8: Topologie et fonctionnement d'un réseau MQTT basique

Ce protocole de communication est déjà présent dans toutes les installations de Panga : c'est le cœur de leur architecture réseau. Il paraît donc évident de l'utiliser pour transmettre mes données. Étant donné que mon programme est écrit en C++, je dois utiliser une bibliothèque pour communiquer en MQTT.

6. Hasimov

Panga a développé pour cela sa propre bibliothèque C++ nommée Hasimov. Bien qu'elle soit peu documentée, elle est assez simple à comprendre et reprend toutes les fonctions utiles du MQTT. Elle permet de se connecter à un broker et d'y publier des messages. Elle fonctionne aussi dans l'autre

sens pour s'abonner et recevoir des messages. La partie qui m'intéresse est évidemment la première : il faut que j'envoie les données collectées. La bibliothèque Hasimov fonctionne avec les signaux et les slots de Qt, dont j'expliciterai le fonctionnement plus bas.

7. Mosquitto

Pour utiliser le protocole MQTT, il nous faut un système de messagerie qui l'implémente : Mosquitto. C'est un broker de message open source léger et facile d'utilisation. Il est très utilisé pour faire fonctionner l'internet des objets. Mosquitto est un projet développé par Eclipse Foundation[12].

8. Qt

Je développe l'entièreté de mon logiciel sous Qt 5.7, sur la demande de l'entreprise. Cela me permet d'utiliser toutes les bibliothèques et classes incluses dans Qt, qui sont souvent bien utiles.

a) Qt Creator

C'est mon IDE[13], qui me facilite grandement la vie pour la compilation croisée. En effet, il permet d'ajouter un périphérique distant, ce que j'ai fait avec mon RPi3 de développement. Il suffit de lui indiquer l'adresse IP du périphérique et les identifiants de connexion en SSH, et le tour est joué. Qt Creator se charge alors d'envoyer le programme compilé sur le RPi3, et me donne un retour visuel de l'exécution du programme sur la console en SSH.

Avoir un IDE me permet aussi de déboguer mon programme et de manipuler les fichiers plus facilement. En somme un gros gain de temps et de productivité.

12 Site web d'Eclipse Foundation : https://www.eclipse.org/
13 Integrated Development Evironment (Environnement de développement) : interface graphique pour coder.

b) QDebug

QDebug fournit un flux de sortie spécialisé pour les informations de débogage. C'est le remplaçant du « std::cout » traditionnel. Je m'en sert alors pour faire sortir tous les journaux (logs) de mon application. Panga a son propre système qui se charge d'enregistrer et d'archiver les journaux, afin de les visualiser en cas de problème.

c) QProcess

QProcess sert à exécuter des programmes externes et à communiquer avec eux. Au début du projet, j'utilisais cette méthode pour faire les requêtes SNMP. Je lançais alors des commandes Linux depuis mon programme avec QProcess.

Bien que cette méthode soit fonctionnelle, elle a un gros désavantage : sa lenteur. Chaque requête SNMP prenait 60 ms pour s'exécuter, valeur correcte, sauf lorsqu'on prend en compte que des dizaines de requêtes sont à faire par seconde. Nous sommes alors limités à 16 requêtes par secondes, le tout avec une consommation du temps CPU déraisonnable. Cette méthode n'est donc pas viable pour ce projet, mais elle m'a permit de commencer sur une base plutôt simple, pour ensuite améliorer mon programme avec une bibliothèque dédiée : SNMP++, présentée ci dessus.

9. Les signaux et les slots

Qt permet une gestion asynchrone des évènements. On a d'un côté les signaux, émis par certains objets (par exemple lors d'un clic sur un bouton), et les slots qui sont reçus par d'autres objets pour exécuter une méthode particulière.

On relie alors les objets émettant des signaux aux objets les recevant avec une méthode « connect », qui lie les deux dans la gestion de l'évènementiel.

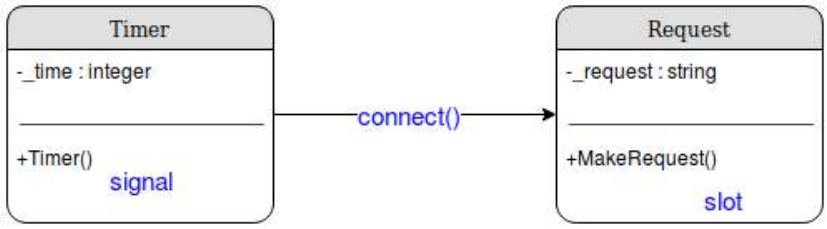


Illustration 9: Schéma d'un système de signaux/slots simplifié

10. Les fichiers INI

Les fichiers INI sont des fichiers textes utilisés pour enregistrer la configuration d'un logiciel, ils sont sauvegardés sous une extension de fichier « .ini ». Ils fonctionnent de manière assez simple et lisible : un homme doit être capable de modifier le fichier. Historiquement, c'est un format inventé par Microsoft[14] en 1985. De nos jours, il est d'usage d'utiliser des fichiers INI pour la configuration des logiciels.

Je stocke donc les paramètres de mon logiciel sur un fichier INI qui sera analysé à l'exécution. C'est dans des fichiers INI que l'on inscrira tous les OIDs nécessaires pour faire les requêtes SNMP, ainsi que tous les paramètres comme les adresses, les ports, les délais, etc.

Le choix de technologie pour cette tâche aurait pu se porter sur un fichier JSON, cependant ce format présente plusieurs inconvénients pour cette utilisation : aucun système de commentaire n'est officiellement supporté. L'indentation avec tous les crochets rend le fichier moins lisible par l'Homme. De plus, Qt fournit un analyseur de fichier INI : « QSettings » simple d'utilisation. Ces raisons justifient l'utilisation de fichiers INI.

11. La documentation

a) HTML - Bootstrap

La documentation pour les fichiers de configuration assez complexes a été faite directement dans un fichier joint en HTML. Pour une rapide mise en forme elle utilise Bootstrap[15]. Cette solution me permet d'expliquer au mieux le fonctionnement de mes fichiers de part la liberté du format.

Je détaille dans cette documentation le paramétrage des fichiers de configuration : de l'utilité de certains champs au fonctionnement global du traitement du fichier. (cf. Annexe 3)

b) Doxygen

Pour la documentation du code source, j'utilise Doxygen qui permet une génération automatique depuis les commentaires présents dans le code. Il permet de générer en partie des diagrammes de classes simples. (cf. Annexe 4)

14 Multinationale spécialisée dans l'informatique. Site web : https://www.microsoft.com/fr-fr
15 Collection d'outils utile à la création du design de sites web. Site web : https://getbootstrap.com/

c) Draw.io

Afin d'avoir un diagramme de classe englobant tout mon projet j'ai utilisé le service web Draw.io qui permet de créer son propre diagramme de classe personnalisé. Ce diagramme global ne pouvait pas être généré automatiquement par Doxygen de part sa complexité. Il est néanmoins nécessaire pour la compréhension du projet. (cf. Annexe 2)

IV. Les difficultés à surmonter

Pendant mon stage en entreprise, j'ai rencontré des problèmes et des difficultés auxquels je n'avais jamais été confronté. Nous allons voir ici comment j'ai géré ces situations, et si j'ai réussi ou non à résoudre mes problèmes.

1. Les compilations de bibliothèques

Les compilations de bibliothèques sous Linux étaient une nouveauté pour moi. Ça n'a pas été facile de comprendre tous les mécanismes de ce procédé au début du stage, d'autant que j'ai plusieurs bibliothèques à utiliser pour mon projet.

Force est de constater que la compilation croisée est compliquée pour beaucoup de personnes : généralement, on tombe rapidement sur des dizaines de personnes qui ont le même problème que nous sur les forums. Ce qui peut être un avantage en soi, car de fait, la solution est souvent à portée de clics.

On peut noter des procédures récurrentes pour la compilation :

1- « ./configure » qui est présent pour nous permettre de donner des options à notre compilation. C'est ici qu'on peut mettre des paramètres pour la compilation croisée.

2- « make » qui effectue la compilation. Grâce à une astuce de Vincent MANZONI, je sais qu'on peut mettre « -j5 » en paramètre afin d'utiliser quatre cœurs du processeur simultanément, réduisant ainsi le temps de compilation, grâce aux tâches réalisées parallèlement.

3- « make install » qui nécessite souvent une élévation de privilèges (en fonction du dossier d'installation). Cela permet d'installer sur la machine ce que nous venons de compiler.

Pour faire de la compilation croisée, il faut compiler pour l'architecture du périphérique voulu, mais depuis un autre PC. La difficulté est de trouver les marqueurs spécifiques pour chaque bibliothèque à indiquer dans le « ./configure » pour effectuer la compilation adéquate pour le

périphérique. Cependant, je travaille avec un RPi3, qui possède une architecture ARM. C'est une architecture très répandue, et la solution se trouve soit directement sur le site de la bibliothèque, soit dans les forums la plupart du temps.

2. L'utilisation de SNMP++

a) Synchrone / asynchrone

La difficulté résulte dans l'exécution de la requête : s'il est simple d'en faire une, en lancer des dizaines peut mettre beaucoup de temps. En utilisant la méthode « get() » de la bibliothèque, j'envoie une instruction bloquante qui envoie une requête SNMP, et attends la réponse, avant de poursuivre l'exécution du programme. Cependant, entre l'envoi et le retour de la requête, il peut s'écouler quelques précieuses millisecondes. En admettant que chaque requête mette dix millisecondes à revenir, cela implique un délai de seconde complète pour faire cent requêtes… seconde où le programme est bloqué et ne peut rien faire d'autre que patienter. Il faut donc une solution pour optimiser cela.

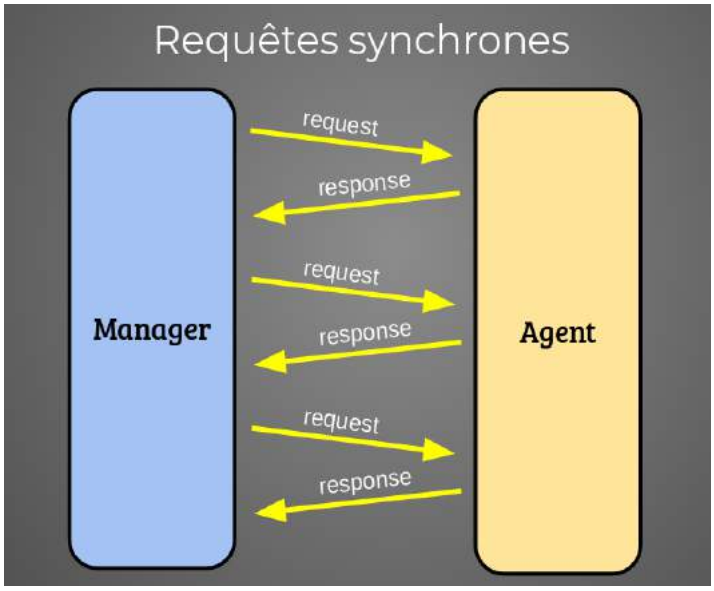


Illustration 10: Timeline des requêtes SNMP effectuées de manière synchrone

La solution réside donc dans l'envoi de requêtes asynchrones. De cette façon, toutes les requêtes sont envoyées quasiment en même temps, et le programme est libéré jusqu'au retour des

requêtes. Cela change fondamentalement le fonctionnement et l'efficacité du programme : il ne reste pas coincé sur une ligne de code à cause d'une instruction bloquante.

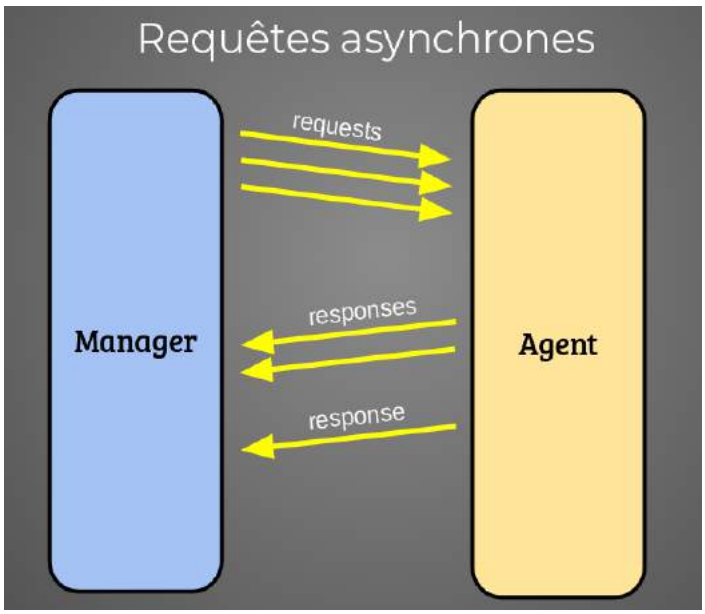


Illustration 11: Timeline des requêtes SNMP effectuées de manière asynchrone

L'implémentation de cette solution a été assez chronophage et laborieux, étant donnée la faible documentation de la bibliothèque et le seul exemple de code que je possédais. Exemple qui, de plus, datait de plusieurs années et contenait des méthodes dépréciées et d'anciennes pratiques de code en C++ qui ne se font plus actuellement.

Une fois les requêtes SNMP rendues asynchrones, les performances du programme sont au rendez-vous, on peut lancer une centaine de requêtes qui sont traitées en quelques millisecondes, le tout pour une utilisation CPU dérisoire.

b) Requêtes multiples

Les requêtes multiples sont normalement gérées avec une commande SNMP Walk, qui parcours tous les nœuds et les feuilles fils dans l'arborescence. L'implémentation de cette commande dans la bibliothèque SNMP++ se nommant « get_bulk() » fonctionne différemment : elle scanne un nombre donné d'OIDs sans pour autant s'arrêter aux nœuds fils, mais en continuant dans l'arborescence.

Cette différence d'implémentation est dérangeante pour l'optimisation du programme. Même si on peut paramétrer un nombre suffisamment grand pour être certain d'avoir tous les OIDs fils, cela implique de faire le tri une fois toutes les réponses reçues. Autrement dit : le programme sera forcément amené à faire des requêtes inutiles.

Prenons comme exemple le nombre de processus en cours. Si je spécifie un nombre maximum de deux cents processus, et qu'à un moment donné, seulement cinquante processus sont lancés, le programme fera cent cinquante requêtes inutiles.

Une potentielle solution existe : faire une suite de requêtes « get_next() ». Ces requêtes vont chercher le prochain OID dans l'arborescence. Je pourrais donc vérifier à chaque réponse que l'OID correspondant est bien une adresse fille de l'originale. Cependant en appliquant ce principe, on perd tous les avantages de l'asynchronicité mise en place précédemment. Cette solution n'est donc pas viable pour un grand nombre de requêtes.

Je travaille actuellement pour trouver une solution à ce problème.

3. Le stockage des données

Comme dit précédemment, les données doivent être stockées dans un fichier INI. En premier lieu, il faut s'assurer de lire le fichier INI une seule fois au démarrage de l'application. En effet, analyser un fichier texte demande du temps et des ressources, ce serait un gâchis de lire le fichier à chaque fois qu'on a besoin des valeurs. Toutes les informations du fichier seront alors mises dans la mémoire vive, afin d'en minimiser le temps d'accès.

Une autre particularité est la manière d'inscrire les OIDs dans le fichier INI. En effet un tel fichier repose sur une structure « clé = valeur », non adéquate pour enregistrer des listes. On a alors une première clé qui sert à définir la liste des OIDs. Chaque OID est ensuite défini dans sa propre section.

```
OIDs=".1.2.3.4.5",".9.8.7.6.5"

;SNMP Get request example
[.1.2.3.4.5]
name=uptime
desc=Time since last start

;SNMP Walk request example
[.9.8.7.6.5]
name=CPU cores
desc=All CPU cores usage
walk=true
max=16
group=4
```

Illustration 12: Structure d'un fichier INI permettant de définir les OIDs

4. Optimisation et performance

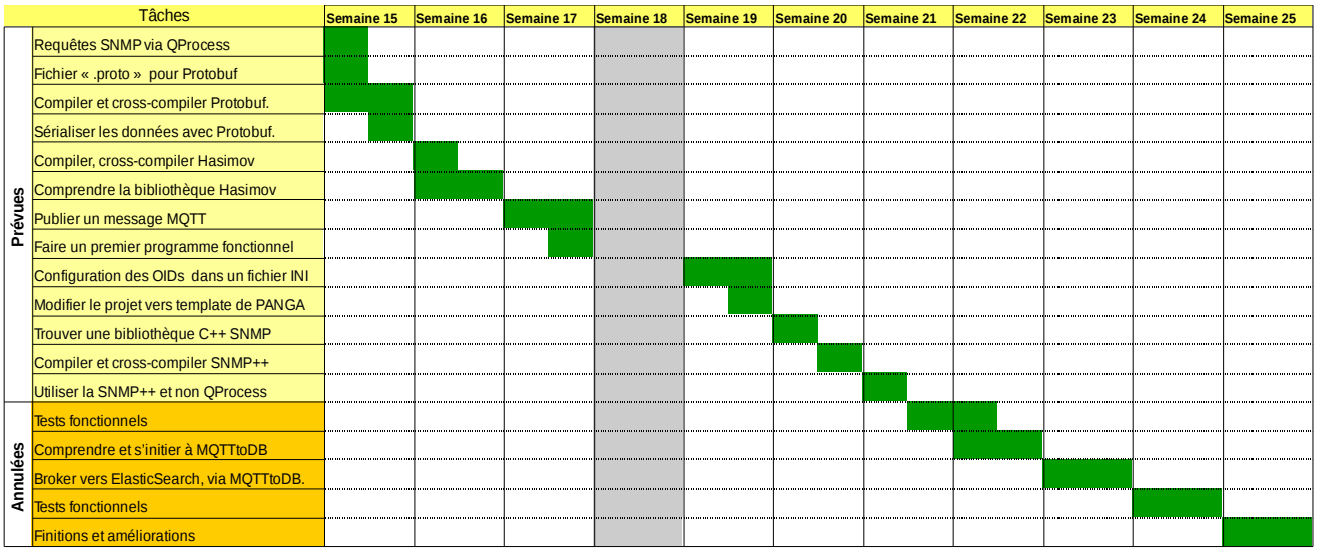
Tout au long du projet, les performance du programme et l'optimisation du code ont été une priorité. Il ne faut pas oublier que cet outil est destiné à un micro-ordinateur aux ressources très limitées. Chaque détail compte, de l'utilisation de la mémoire vive à l'utilisation du processeur.

Je développe un outil de supervision, il doit alors être le plus transparent possible lors de son exécution. Il ne s'agit pas de faire un logiciel de monitoring qui consomme toutes les ressources de la machine. L'utilisation du CPU doit par exemple se limiter à quelques petits pourcents, car le logiciel sera lancé dès que la machine sera allumée, d'autant que pour des petits processeurs comme ceux d'un RPi3, l'utilisation peut monter très vite au moindre algorithme lancé.

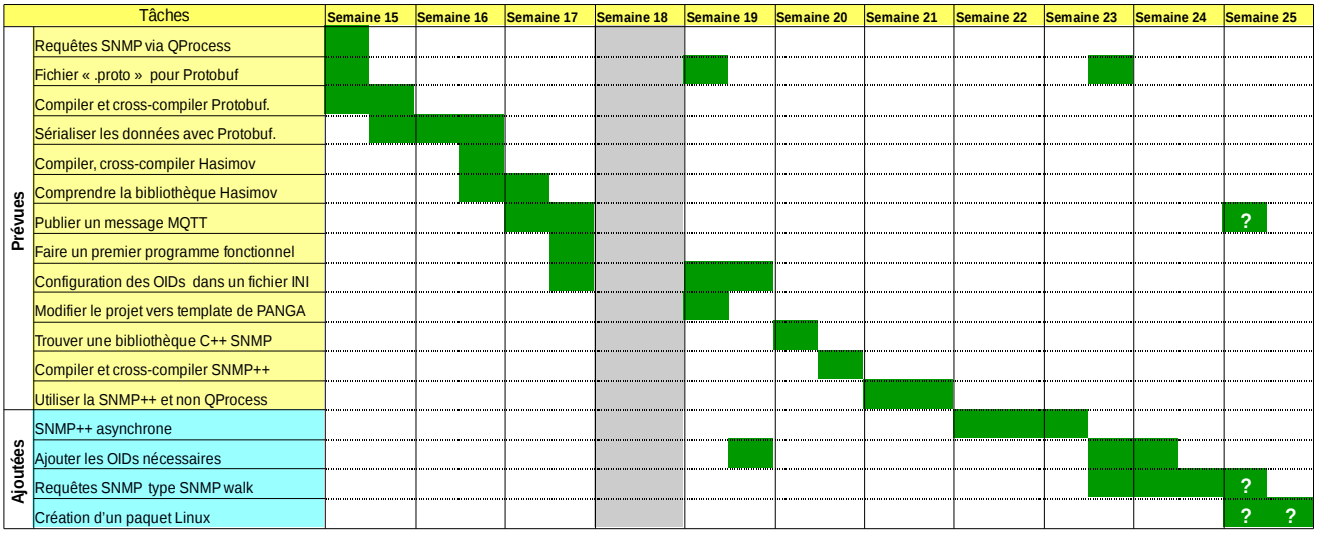
V. Planning

Vincent MANZONI et Jérôme DAVID ont réalisé un planning que j'ai tenté de suivre le long de mon stage. Il y a eu cependant un changement de planning lors de l'arrivée d'un collègue stagiaire : Loïc PONCHAUX. Certaines tâches lui ont été déléguées, et de nouvelles m'ont été assignées.

1. Prévisionnel



2. Réel



Conclusion

Ce stage est ma première expérience professionnelle dans le domaine informatique. Il m'a permis de mieux me représenter le monde du travail et de mettre mes connaissances au service d'une entreprise et m'a fait prendre conscience des enjeux du monde du travail.

J'ai beaucoup appris durant ce stage, notamment dans dans la compilation de bibliothèques sous Linux, dans le développement sous Qt et sur la compilation croisée. Cette expérience n'a pas été enrichissante uniquement sur le plan des connaissances, mais m'a également fait prendre de l'autonomie et de la rigueur dans mon travail : presque la totalité des technologies sur lesquelles j'ai travaillé m'étaient inconnues à mon arrivée dans l'entreprise.

Le programme que j'ai développé sera poursuivi par l'entreprise afin de l'intégrer au mieux dans son système. Ce projet est essentiel pour Panga et était sur le chemin critique. La tâche qu'ils m'ont confié était donc importante.

Bien que les connaissances acquises pendant le stages soient importantes, le plus utile aura été d'apprendre à chercher et à trouver les solutions par moi même. C'est fondamental en informatique, on développera toujours quelque chose de nouveau, qui n'existait pas avant. Il est alors important de ne pas trop s'appuyer sur nos connaissances mais bien d'aller plus loin, c'est ce qui apporte de la valeur à notre code.

Résumé

Après avoir commencé un projet d'entreprise avec la start-up Panga, j'ai eu l'opportunité de le poursuivre durant mon stage de fin d'études. Panga est une entreprise spécialisée dans la domotique, elle a développé son propre système permettant aux habitants contrôler et de superviser leur logement via une tablette - interface NeHo - et de gérer les résidences pour les propriétaires, techniciens et fournisseurs - interface NeMa - via un service en ligne type SaaS.

Panga ne dispose pas de système de supervision à distance viable pour leur architecture. Lors de mon stage, mon rôle a été de développer un outil de supervision pour la solution Panga. L'objectif final étant d'intégrer cet outil à l'interface NeMa pour que les techniciens puissent faire un diagnostic à distance et intervenir en cas de panne dans un logement. J'ai travaillé en collaboration avec un collègue stagiaire : je m'occupais des couches basses de l'application, et lui des couches hautes.

J'ai développé une application en C++ sous Qt, qui récolte des informations périodiquement sur les différents micro-ordinateurs présents dans les logements à l'aide du SNMP. L'application sérialise ensuite les données avec Protobuf, pour les envoyer sur le réseau des résidences déployé par Panga, constitué de Brokers MQTT. Les diverses technologies sont détaillées dans ce rapport.

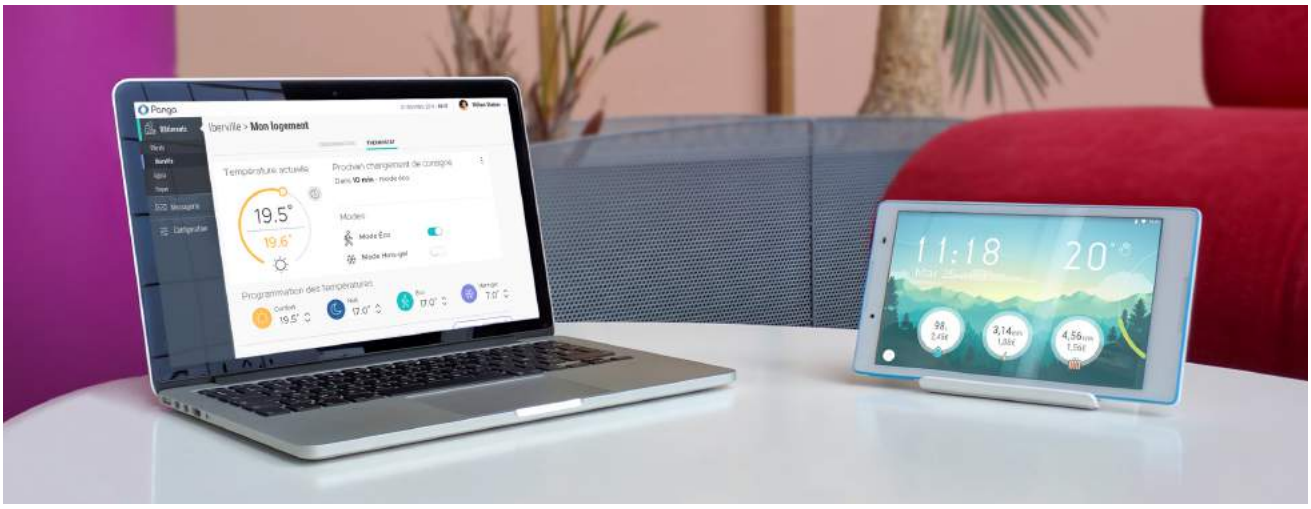
Abstract

After starting a business project with the Panga start-up, I had the opportunity to complete it during my final internship. Panga is a company specialized in home automation, it has developed its own system allowing residents to control and supervise their housing via a tablet - the NeHo interface - and to manage residences for owners, technicians and suppliers - the NeMa interface - via an online service.

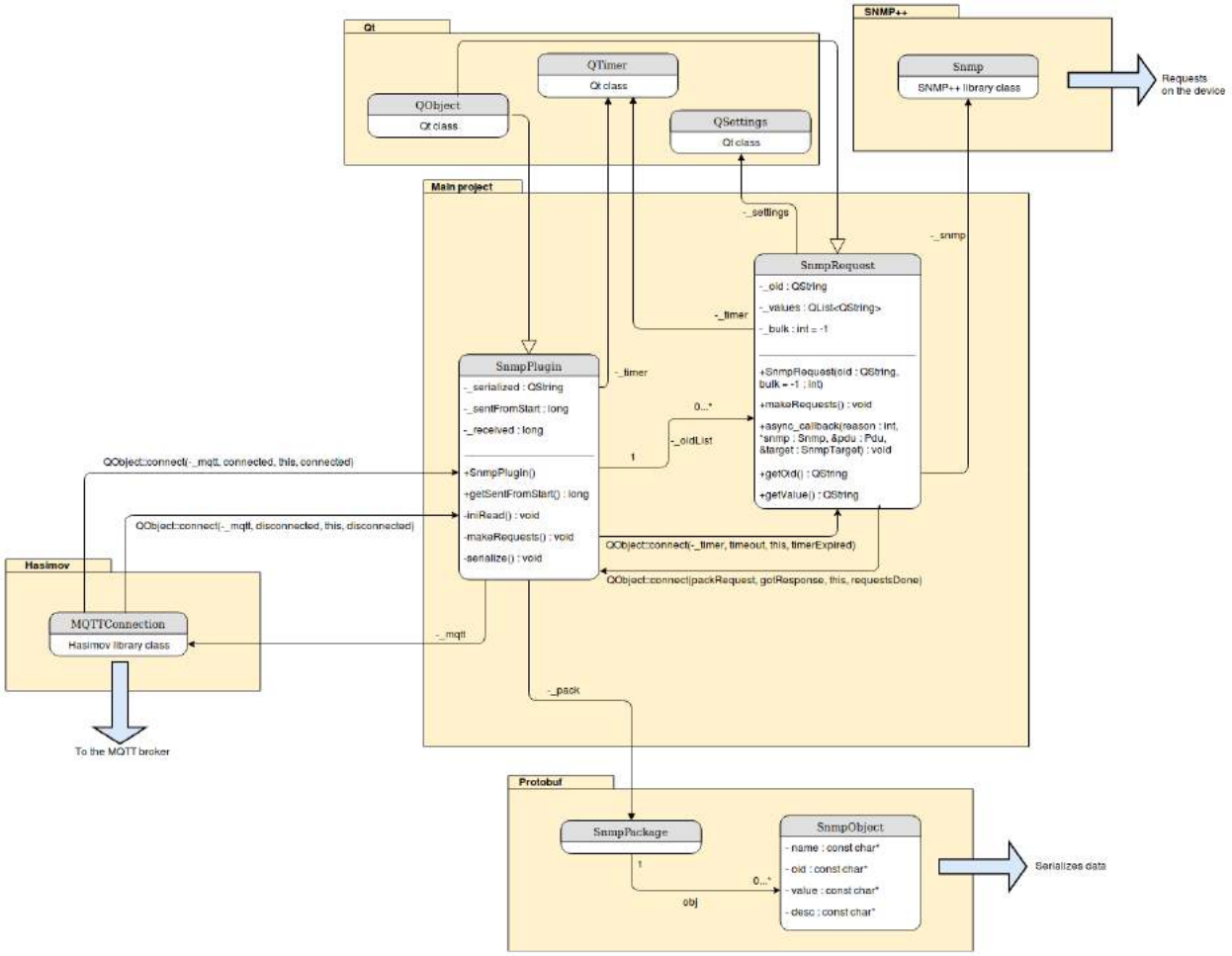
Panga does not have a sustainable remote monitoring system for their architecture. During my internship, my goal was to develop a supervision tool for the Panga solution. The final objective is to integrate this tool into the NeMa interface in order technicians to perform a remote diagnosis and act in case of a breakdown in a housing. I worked in collaboration with a trainee colleague: I was in charge of the lower layers of the application, and he was in charge of the upper layers.

I developed an application in C++ under Qt, which collects information periodically on the various microcomputers present in the housing using the SNMP. The application then serializes data with Protobuf, to send it to the network deployed by Panga, made up of MQTT Brokers. The various technologies are detailed in this report.

Annexes



Annexe 1 : *NeMa à gauche et terminal NeHo à droite*



Annexe 2 : *Diagramme de classes du projet*

OIDs file structure

File path : "config/config.ini" by default.
The name of this file is defined in **"settings.ini"**, in "snmp" section.

Description

This file is used to set all the SNMP OIDs to retrieve from the device. First you have to make a list of the OIDs you wants, then you can define their properties by creating sections for each OID.

First key

The file should start with a key called "OIDs". This contains the list of all the OIDs sections.

Key	Type	Description	Required/Optional	Default
OIDs	string list	A list with all the OIDs to retrieve. Uses a comma (,) as separator, each string is surrounded by quotes ("). For further explanations, see the example below.	Required	--

OIDs sections

Key	Type	Description	Required/Optional	Default
name	string	A significant name of the OID value. This name will be the same in Protobuf messages.	Required	--
desc	string	Brief description of the OID value. This description will be the same in Protobuf messages.	Optional	"No description"
walk	boolean	Indicates whether or not the program have to make an SNMPWalk request intead of a SNMPGet request. For multiple requests protobuf will send **only one string** with a list inside. The separator is a semicolon (;).	Optional	false
max	integer	**Only for SMPWalk requests.** Maximum number of OIDs returned by the SNMPWalk. Larger values will be more CPU and RAM consuming.	Optional	20
group	integer	**Only for SMPWalk requests.** The main group of requests will be splitted in smaller groups. This tag controls the size of the small groups created. **Should be at least twice smaller than "max" tag to take effect,** larger values are useless. Small values are time consuming and large values will result in more CPU usage, balance it according to your needs. **Not implemented yet !**	Optional	10

Annexe 3 : *Extrait de la documentation réalisée : fichier de configuration pour les OIDs*

Member Function Documentation

◆ async_callback()

```
void SnmpRequest::async_callback ( int                  reason,
                                   Snmp_pp::Snmp *      snmp,
                                   Snmp_pp::Pdu &       pdu,
                                   Snmp_pp::SnmpTarget & target
                                 )
```

Should not be called by yourself, except you know exactly what you are doing.

Parameters

reason The reason for this callback

snmp The Snmp object from SNMP++ lib.

pdu The Pdu object from SNMP++ lib.

target The target you created to define SNMP.

◆ getOid()

```
QString SnmpRequest::getOid ( )
```

Getter for the OID.

Returns

Returns the OID

◆ getValue()

```
QString SnmpRequest::getValue ( )
```

Getter for the value.

Is empty when instanciating the object, will take the right value after a **makeRequest()**.

Returns

Returns the first OID value on the list (for SNMP Get)

Annexe 4 : *Extrait de la documentation réalisée : documentation Doxygen*

```
syntax = "proto2";

package proto.monitoring;

message SnmpObject {
     required string oid = 1;
     required string value = 2;
     required string name = 3;
     optional string description = 4;
     optional sint32 walk = 5;
}

message SnmpPackage {
     repeated SnmpObject obj = 1;
}
```

Annexe 5 : *Fichier de structure Protobuf pour la sérialisation des données*

```
void SnmpPlugin::serialize(){
    //Protobuf
    QString serialized =
                QString::fromStdString(_pack.SerializeAsString());

    qDebug() << "Message serialized :";
    qDebug() << serialized;

    _serialized = serialized;
}
```

Annexe 6 : *Code source "SnmpPlugin.cpp". Sérialisation avec Protobuf, une fois la structure mise en place. On voit la facilité de l'outil.*

```
\nJ\n\u0012.1.3.6.1.2.1.1.3.0\u0012\u00132 days, 19:31:25.84\u001A\u0006uptime\"\u0015Time since last
start(\u0001\nG\n\u0017.1.3.6.1.4.1.2021.4.5.0\u0012\u0006949580\u001A\tTotal RAM\"\u0017Total of dynamic
memory(\u0001\nB\n\u0018.1.3.6.1.4.1.2021.4.11.0\u0012\u0006862784\u001A\bRAM free\"\u0012Amount of free
RAM(\u0001\nA\n\u0017.1.3.6.1.4.1.2021.4.6.0\u0012\u0006760388\u001A\bRAM used\"\u0012Amount of used
RAM(\u0001\nQ\n\u001A.1.3.6.1.4.1.2021.10.1.3.1\u0012\u00041.10\u001A\rCPU load 1min\"\u001CCPU load for the last
minute(\u0001\nT\n\u001A.1.3.6.1.4.1.2021.10.1.3.2\u0012\u00041.00\u001A\rCPU load 5min\"\u001FCPU load for the 5 last
minutes(\u0001\nV\n\u001A.1.3.6.1.4.1.2021.10.1.3.3\u0012\u00040.65\u001A\u000ECPU load 15min\" CPU load for the 15 last
minutes(\u0001\nX\n\u0019.1.3.6.1.4.1.2021.11.50.0\u0012\u000590324\u001A\bCPU User\"(Sum of CPU time used by the current
user(\u0001\nW\n\u0019.1.3.6.1.4.1.2021.11.52.0\u0012\u000538606\u001A\bCPU time\"'Sum of CPU time used since last
restart(\u0001\n^\n\u0019.1.3.6.1.4.1.2021.11.53.0\u0012\b97008373\u001A\bCPU Idle\"+Sum of CPU time not used since last
restart(\u0001\nT\n\u0019.1.3.6.1.4.1.2021.9.1.7.1\u0012\b12263116\u001A\tdisk free\" Amount of free space on the
disk(\u0001\nS\n\u0019.1.3.6.1.4.1.2021.9.1.8.1\u0012\u00072327848\u001A\tdisk used\" Amount of space used on the
disk(\u0001\nV\n\u0019.1.3.6.1.4.1.2021.9.1.6.1\u0012\b15242572\u001A\ndisk total\"!Total amount of space on the
disk(\u0001\n=\n\u0019.1.3.6.1.4.1.2021.9.1.2.1\u0012\u0001/\u001A\ndisk mount\"\u000FDisk mount
path(\u0001\nE\n\u0019.1.3.6.1.4.1.2021.9.1.9.1\u0012\u000215\u001A\tdisk rate\"\u0017Percentage of disk
used(\u0001\nJ\n\u0017.1.3.6.1.4.1.2021.4.3.0\u0012\u0006102396\u001A\tSwap size\"\u001ASize of the swap
partition(\u0001\nO\n\u0017.1.3.6.1.4.1.2021.4.4.0\u0012\u0006102396\u001A\tSwap free\"\u001FAmount of swap memory
available(\u0001\n\u0001\n\u0012.1.3.6.1.2.1.1.1.0\u0012ILinux raspberrypi 4.9.79-v7+ #1091 SMP Tue Feb 6 13:18:45 GMT 2018
armv7l\u001A\nOS version\"+Operating system name and installation
date(\u0001\nD\n\u0012.1.3.6.1.2.1.1.5.0\u0012\u000Braspberrypi\u001A\u000BSystem name\"\u0012Name of the
system(\u0001\nW\n\u0017.1.3.6.1.2.1.25.3.3.1.2\u0012\t1;1;100;1\u001A\nCore usage\"#Core usage rate for the last minute(
```

Annexe 7 : *Aperçu en ASCII d'un message Protobuf sérialisé*

```
void SnmpRequest::async_callback(int reason, Snmp_pp::Snmp * /*snmp*/, Snmp_pp::Pdu &pdu,
Snmp_pp::SnmpTarget &target)
{
    // Check the reason of callback
    if (reason == SNMP_CLASS_ASYNC_RESPONSE)
    {
        qDebug() << "Response !";
    }
    else if (reason == SNMP_CLASS_TIMEOUT)
    {
        qWarning() << "Timeout  !";
    }
    else
    {
        qWarning() << "Did not receive any async response/trap.";
    }

    // The Pdu must contain at least one Vb
    if (pdu.get_vb_count() == 0)
    {
        qWarning() << "Error : Pdu is empty";
    }else{

        QString valueStr;

        for(int i=0; i<pdu.get_vb_count(); i++){
            Snmp_pp::Vb vbb;
            pdu.get_vb(vbb,i);

     //We have to check if this is relly a child of OID (for SNMP Walk requests)
            if(QString::fromStdString(vbb.get_printable_oid()).prepend(".").contains(_oid))
{

                valueStr = vbb.get_printable_value();
                if(!valueStr.isEmpty()){
                    if(i!=0){
                        _value += SETTINGS::SNMP::SEPARATOR;
                    }else{
                        //Remove the default value before concatenate
                        _value = "";
                    }
                    _value += valueStr;
                }
            }
        }

        // Display OID and value
        qDebug() << "For this OID :" << _oid;
        qDebug() << "Found this value :" << _value;

        emit gotResponse();
    }
}
```

Annexe 8 : *Code source fichier "SnmpRequest.cpp". Callback pour la réception de la valeur SNMP*