Projet SuPer

Supervision de Personnes

Laboratoire Souterrain à Bas Bruit de Rustrel

JAMIN Brice
SCHERER Nicolas

Dossier Technique
Revue n°2
Lundi 8 Avril 2013

BTS IRIS - Lycée Alphonse Benoit - L'Isle-sur-la-Sorgue
Académie d'Aix-Marseille - Session 2013

Projet SuPer – LSBB

Sommaire

pages :

Partie personnelle – Brice JAMIN **3**

- Suivi de l'avancement 4,5,6,7
- Conception interface Visualiser/Détecter lecteurs 8,9
- Conception détecter lecteurs, Encapsulation de QtcpServer 10,11
- Activation/Désactivation de l'écoute 12
- Modification de l'adresse et du port 13
- Connexion d'un nouveau client 14
- Fonctionnement MultiThread 15
- Forcer l'arrêt des communications 16
- Filtrage des clients 17
- Bilan 18

Partie personnelle – Nicolas SCHERER **19**

- Suivi de l'avancement 20,21
- Mise en place Gestionnaire de Version 22,23
- Modifications apportées, conception de Visualiser itération 1 24,25
- Codage initialisation IHM et visualisation lecteurs dynamiquement 26,27
- Choix de codages 28
- Séparation base de données (BDD) 28,29
- Scénarios d'erreurs – initialisation logiciel 30
- Plan et tests unitaires de Visualiser (IHM, Visualisation lecteurs) 31,32
- Spécification Interface Homme-Machine itération 2 33
- Organisation des données – itération 2, Spécification de la base de données 34,35
- Conception de Visualiser itération 2 (Visualiser Personnes) 36,37,38
- Début de codage Visualiser itération 2 39
- Bilan et Conclusion 40

Projet SuPer – LSBB

Partie personnelle

Brice JAMIN

Revue 2

Projet SuPer – LSBB

Suivi de l'Avancement

Compte-rendu d'activités

Suivi du Projet	
Date	*Objet*
22/02-27/02	Découverte et apprentissage du gestionnaire de version Git
27/02-28/02	Création d'un dossier de projet adapté au suivi de version
01/03	Découverte de QtCreator
02/03	Création classes TcpServer, TcpServerThread, ReaderDetector
03/03	Allumage / Extinction ReaderDetector Liaison TcpServer - ReaderDetector Lancement d'un nouveau thread
03/03-07/03	Documentation sur la gestion des threads
07/03	Récupération via requêtes SQL des champs d'un lecteur Identification d'un lecteur
08/03	Termine le thread lors d'une erreur Signalement détection d'un intrus avec transmission de son adresse ip
09/03	Création classe Reader
10/03	Signalement détection d'un reader avec transmission du reader
11/03	Découverte de l'event loop de Qt Classes Reader et ReaderClient deviennent respectivement ReaderData et Reader Création classe ClientTcp
12/03	ClientTcp hérite de QTcpSocket Création classe ClientTcpWindow Liaison classes ClientTcp et ClientTcpWindow Lancement de nouveaux ClientTcpWindow possibles via MainWindow Correction des clefs étrangères du script SQL de la BDD Problèmes : Les clients lancés en local ont tous la même adresse, 127.0.0.1 Ce qui rend la différenciation des readers impossibles en local
14/03	Problèmes : Plusieurs connexions SQL effectuées dans des threads différents impossible Fermeture de la connexion précédente lors de l'ouverture d'une nouvelle connexion

Projet SuPer – LSBB

Suivi de l'Avancement

Compte-rendu d'activités

Suivi du Projet	
Date	*Objet*
15/03	Problème : Erreur lors de la transmission d'objets en références constantes par signaux
17/03	Corrections : Des threads qui s'exécutaient encore lors de la fermeture du programme De la connexion BDD qui restait ouverte lors de la fermeture du programme Suppression de la classe Reader Renommage de la classe ReaderData en Reader
18/03	Renommage des signaux Rend possible la déconnexion d'un lecteur
18/03-19/03	Connexion d'un Reader met à jour la BDD
22/03	Correction : Ferme la socket avant la fin du run du thread
23/03	Découverte des différentes méthodes d'implémenter les threads avec Qt Création de la classe Thread, ré-implémentation de sa méthode run Création de la classe TcpClientManager Découverte de la nécessité de QTcpServer::incommingConnection() pour le multithread Ajout de l'attribut socketDescriptor Signalement des erreurs Tcp et Sql Emploi de moveToThread Récupération du bon QTcpSocket via setDescriptor
24/03	Découvertes : Avec les QObject, un parent détruit ses fils lors de sa destruction Un signal connecté à un slot en monothread revient à faire un appel direct du slot
25/03	Découverte du risque d'exécuter des slots dans le thread principal et non pas dans un nouveau
26/03	Connexions effectuées après un moveToThread permettent l'exécution dans le bon thread
27/03	Découverte du server non bloquant sans multithread Orientation vers un programme pouvant fonctionner en monothread ou en multithread

Projet SuPer – LSBB

Suivi de l'Avancement

Compte-rendu d'activités

Suivi du Projet	
Date	*Objet*
28/03	Rassemblement des tests en une seule application Créations de la classe Server, de ses signaux, slots, attributs, accesseurs Liaison IHM-Server effectuée depuis le main Connexion signaux IHM - slots Server Prise en compte par l'IHM de l'invalidité des ports et de l'adresse Sécurisation mise en écoute et accesseurs du Server IHM change sur démarrage/arrêt de l'écoute par le Server Création dans le Server des signaux port/address changés
29/03	Création enum Server::SwitchOnState pour gérer les erreurs d'écoute du Server
30/03	Surcharge operator <<() pour afficher les enums Server::SwitchOnState Volonté de protéger le contenu de Server hérité par QTcpServer Découverte de la possibilité d'accèder à des données protected depuis une petite-fille Héritage en protected de QTcpServer non réalisable Choix de l'encapsulation de QTcpServer au lieu de l'héritage Création classe TcpServer et implémentation de sa méthode incomming connection
30/03	Encapsulation de TcpServer dans Server Création dans Server du slot incommingConnection(int) Début réflexion sur la gestion de la libération des threads et objets liés aux threads
31/03	Libération des ressources en évitant delete this Destruction des threads via connexion dans le constructeur Création d'un signal déclenchant la libération des ressources Compte/décompte des threads lors de la création/destruction des threads Refus de la fermeture de la fenêtre et libération des ressources

Projet SuPer – LSBB

Suivi de l'Avancement

Compte-rendu d'activités

Suivi du Projet	
Date	*Objet*
01/04	Mémorisation de la demande de fermeture du programme et libération des ressources Modifications des attributs si différents après listen Tests sur les transmissions par signaux Création classe ClientManager, attributs, constructeur, destructeur Création slot stop et signal finished Création de la classe Thread et suivi des signaux de QThread Création bouton déclenchant la libération des ressources Création signal/slot pour stopper les ClientManager Implémentation de Server::incommingConnection et connexion avec TcpServer Découverte : Les attributs d'un QObject changeant de thread doivent l'avoir pour parent Création classe TcpSocket et suivi des signaux de QTcpServer Multithread fonctionnel Envoi du Reader vers l'IHM Transmission de ClientConnexion par référence constante
03/04	Filtrage des clients par vérification de l’existence de leur adresse dans la Bdd

Projet SuPer – LSBB

Partie personnelle

Conception interface Visualiser/Détecter lecteurs

La bibliothèque Qt offre un mécanisme de communication entre objets héritant de QObject reposant sur l'utilisation des signaux et slots.
Un slot est similaire à une méthode d'une classe et peut être connecté à des signaux, qui, une fois émis, déclencheront l'exécution du slot.

Il existe plusieurs type de connexions entre signaux et slots, dont les trois principales sont :

→ AutoConnection :

C'est le type de connexion par défaut.
Si les instances des objets connectés appartiennent au même thread, le type de connexion effectué est alors DirectConnexion.
Si elles n'appartiennent pas au même thread, le type de connexion effectué est alors QueuedConnection.

→ DirectConnection :

Lorsque le signal est émis, le slot est immédiatement exécuté.
Ce comportement est comparable à l'appel d'une fonction : le bloc de code appelant ne se poursuivra que lorsque l'exécution de la fonction sera terminée.
L'exécution du slot est alors synchrone.

→ QueuedConnection :

Lorsque le signal est émis, il est mis dans une liste FIFO de l'event loop, la boucle d'évènements.
C'est une boucle infinie qui exécute successivement les événements de sa liste, et les y ôte.
Lors de l'exécution du programme les événements sont ajoutés dans une liste d'attente et le bloc de code contenant l'émission continue son exécution.
Une fois l'exécution terminée la boucle d'événements traite la file d'attente des signaux et déclenche l'exécution des slots auxquels ils sont connectés.
L'exécution des slots est alors asynchrone.

Projet SuPer – LSBB

Partie personnelle

Conception interface Visualiser/Détecter lecteurs

Ces types de connexions permettent ainsi la communications entre objets, même s'ils sont dans des threads différents.

Il est donc aisé de permettre à un objet d'en observer un autre lorsqu'ils héritent de QObject.

La partie "Visualiser" sera une IHM basée sur une application graphique Qt, composée d'une instance de la classe QMainWindow, qui délivre l'event loop du thread principal.
Héritant de QObject, elle peut utiliser les signaux et les slots.
Son fonctionnement repose en partie sur les informations détenues par la partie "Détecter", un serveur pouvant fonctionner en multithreads.
La communication entre threads différents étant grandement facilitée par les signaux, la partie "Détecter" devra employer des objets héritant de QObject et communiquer avec "Visualiser" par leur intermédiaire.

"Visualiser" doit observer "Détecter" et être mise à jour pour refléter l'évolution de "Détecter".
"Détecter" devra donc mettre à la disposition de "Visualiser" des signaux destinés à la renseigner de tous les événements dont il a connaissance.
"Visualiser" peut être composé de plusieurs objets observant des événements différents de "Détecter".

C'est pourquoi, afin de simplifier la gestion de la communication, "Visualiser" aura la responsabilité de connecter ses propres objets à "Détecter", et ce dernier ignorera qui l'observe.

Ainsi, la partie "Détecter" fonctionnera indépendamment de "Visualiser", et "Visualiser" reposera uniquement sur les noms des objets de "Détecter" et de leurs signaux et slots, ce qui limite les dépendances entre les deux : seules les connexions devront être changées depuis "Visualiser" en cas de changement.

Projet SuPer – LSBB

Partie personnelle

Conception Détecter lecteurs
Encapsulation de QtcpServer

Il existe une classe QTcpServer qui permet d'écouter sur une adresse et un port, et de signaler l'arrivée des clients Tcp.

Elle possède pour cela une méthode protected nommée incommingConnexion, qui transmet en paramètre le descripteur de la socket du client.

Une fois en possession de ce descripteur, il est alors possible de créer un QTcpSocket, d'en modifier le descripteur avec setDescripteur, et il est alors possible de communiquer avec le nouveau client.

Cette méthode est appelée automatiquement par la classe QTcpServeur, et pour recevoir son descripteur il est obligatoire de dériver de QTcpServer et de ré-implémenter sa méthode incommingConnexion(int).

Je peux donc créer une classe Server pour cela et effectuer un héritage public de QTcpServer. Mais cette classe servira d'interface à l'IHM, et je ne souhaite pas que l'IHM puisse accéder à certaines méthodes publiques héritées de QTcpServer. Il me faut protéger ma classe Server des accès non désirés.

Je pourrai hériter de manière protected, mais cela rendrait également protected les méthodes de la classe mère de QtcpServer, QObject, ce qui interdirait toute communication via signaux et slot sur ma classe Server, ce qui n'est pas envisageable, la communication avec l'IHM devant s'effectuer justement via signaux et slots.

Pour résoudre ce problème j'ai utilisé une classe intermédiaire, nommée TcpServer, dérivée publique de la classe QTcpServer, à laquelle j'ajoute un signal sig_incommingConnexion(int).
J'ai ensuite ré-implémenté la méthode incommingConnexion(int), récupéré le descripteur transmis, et l'ai émis vers l'extérieur à l'aide du nouveau signal sig_incommingConnexion(int).

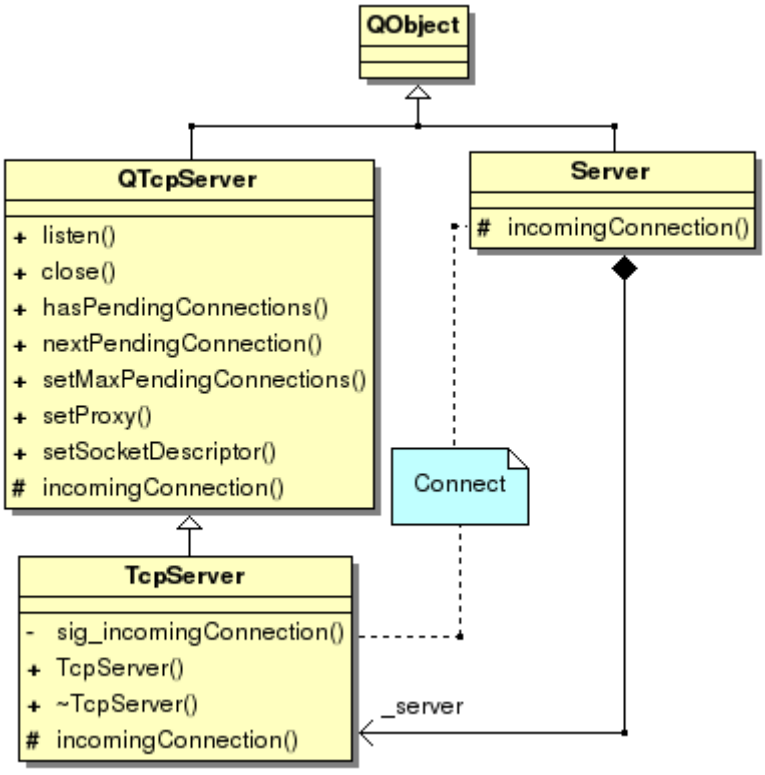
Projet SuPer – LSBB

Partie personnelle

Conception Détecter lecteurs
Encapsulation de QtcpServer

Ma classe Server n'a plus qu'à encapsuler une instance de cette classe TcpServer et connecter le signal TcpServer::incommingConnection() à un slot, nommé incommingConnexion().

La classe Server a ainsi une méthode semblable à celle de QTcpServer, sans pour autant en hériter, ce qui assure que l'extérieur n'y a pas accès et est obligé de passer par Server pour agir dessus.



Projet SuPer – LSBB

Partie personnelle

Activation désactivation de l'écoute

La classe Server possède deux slots publics switchOn() et switchOff(), qui ont pour rôle d'activer/désactiver l'écoute du serveur. Elle repose sur l'utilisation des méthodes listen() et close() de la classe QTcpServer.
Ce sont des slots pour permettre à une IHM de connecter par exemple le signal clicked() d'un de ses QPuhButton.
Elle a ainsi le choix : utiliser les signaux ou faire appel directement au slot, ce qui est possible car les slots sont des méthodes.
La différence est que si elle utilise les signaux elle ne pourra pas accéder au retour de la méthode switchOn, qui est un énuméré nommé SwitchOnState contenant :

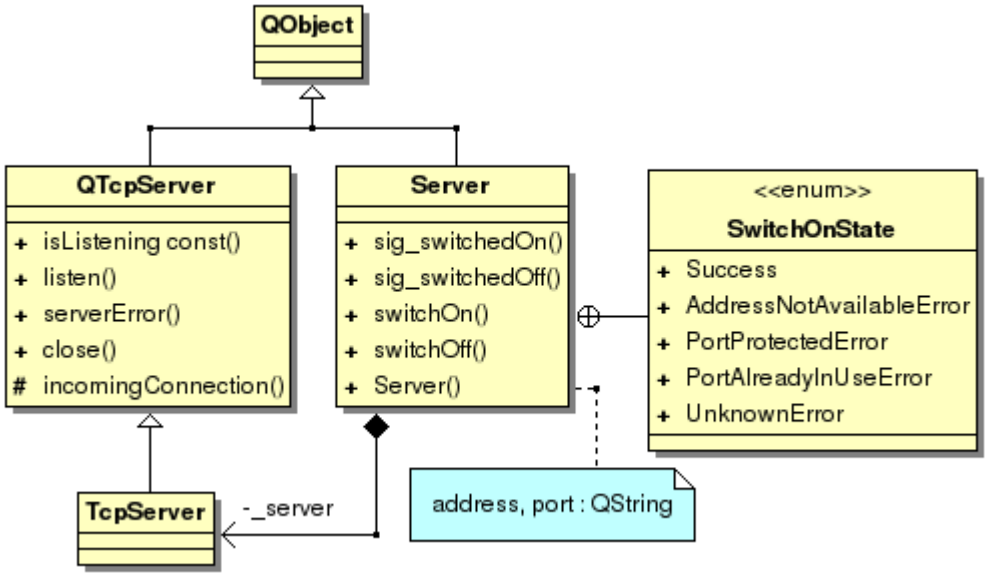
→ Success : L'appel a réussi, le serveur est en écoute et le signal switchedOn() a été émis.

→ AddressNotAvailableError : L'adresse d'écoute n'est pas libre, il faut alors la libérer (arrêter le programme qui l'utilise) ou en choisir une autre.

→ PortProtectedError : Le port demandé pour la mise en écoute est réservé pour d'autres applications. C'est par exemple le cas du port 80, utilisé pour la communication HTTP.

→ PortAlreadyInUseError : Le port est déjà utilisé, il faut le libérer (fermer l'application qui l'utilise) ou en choisir un autre.

→ UnknowError : Une erreur inconnue est survenue. Normalement cela ne devrait pas arriver, mais si jamais cela arrive un jour, l'utilisateur saura qu'une erreur est survenue, même inconnue, ce qui est mieux que de l'informer que tout va bien alors que tout va mal.



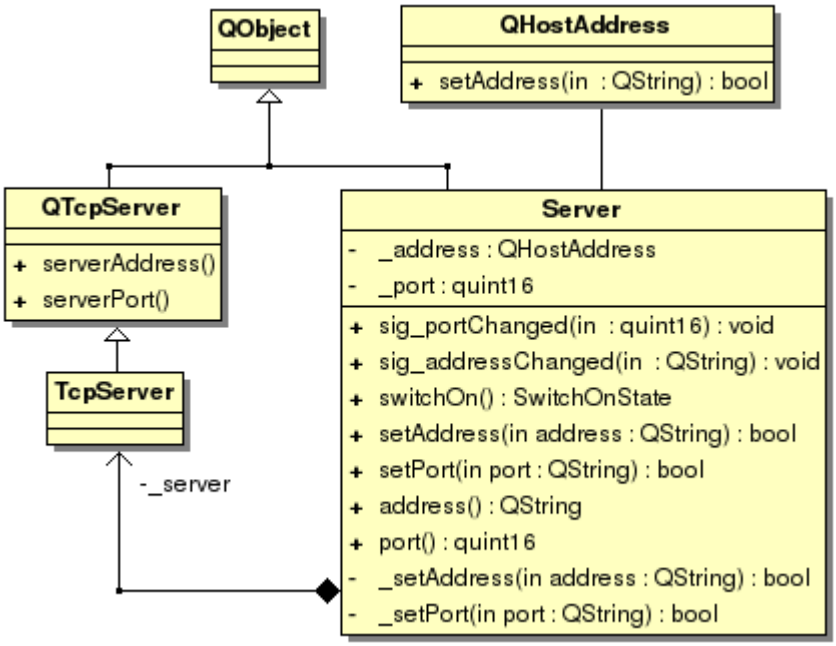
Projet SuPer – LSBB

Partie personnelle

Modifications de l'adresse et du port

La classe Server a pour attributs une adresse d'écoute et un port.
Il faut les lui renseigner avant de la switchOn() sans quoi la demande sera ignorée en retournant une erreur.
Ils sont modifiables depuis :
→ Les paramètres du constructeur
→ Les accesseurs en écriture setAddress(QString) et setPort(QString). Ils sont tous deux des slots.
Ils prennent en paramètre des Qtring, classes courantes et faciles d'utilisation pour l'IHM. Elle peut ainsi, par exemple, connecter les signaux textChanged(QString) de ses QLineEdit directement sur ces slots.

Quelque soit l'accès utilisé, en cas de valeur correcte, les signaux correspondants sig_portChanged(quint16) et sig_addressChanged(QString) seront émis, et en cas de valeur incorrecte les attributs resteront inchangés.
Les accesseurs en écriture quant à eux retournent en plus un booléen, à true si la valeur est correcte, sinon à false.
Lors de la transmission de valeurs par les paramètres du constructeur, seul le prochain appel à listen permettra de savoir s'ils étaient incorrects.

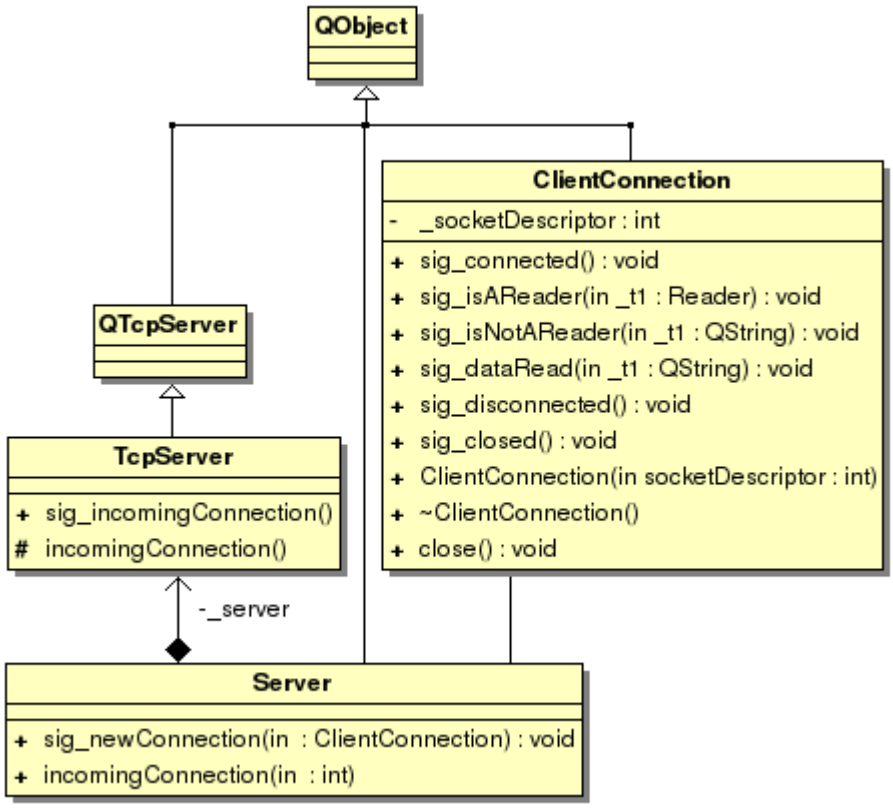


Projet SuPer – LSBB

Partie personnelle

Connexion d'un nouveau client

A la connexion d'un nouveau client, une instance de la classe ClientConnection est créée dynamiquement.
Son rôle est de gérer tout ce qui est en rapport avec la communication avec le client.
Elle se crée en fournissant en paramètre de son constructeur le descripteur de socket du nouveau client, qu'elle mémorise dans un attribut.
Elle offre plusieurs signaux sur lesquels l'IHM peut vouloir se connecter, c'est pourquoi elle est envoyée vers l'extérieur via l'émission du signal sig_newConnection(const ClientConnection&).
Si l'IHM connecte l'un de ses slots à ce dernier signal, elle peut alors se connecter aux signaux du ClientConnection reçu :
→ sig_isAReader() : Le nouveau client est un lecteur
→ sig_isNotAReader() : Le nouveau client n'est pas un lecteur.
→ sig_dataRead() : Le client a envoyé des données
→ sig_disconnected() : Le client s'est déconnecté
→ sig_closed() : La communication avec le client à été fermée sur l'appel de la méthode close().



Projet SuPer – LSBB

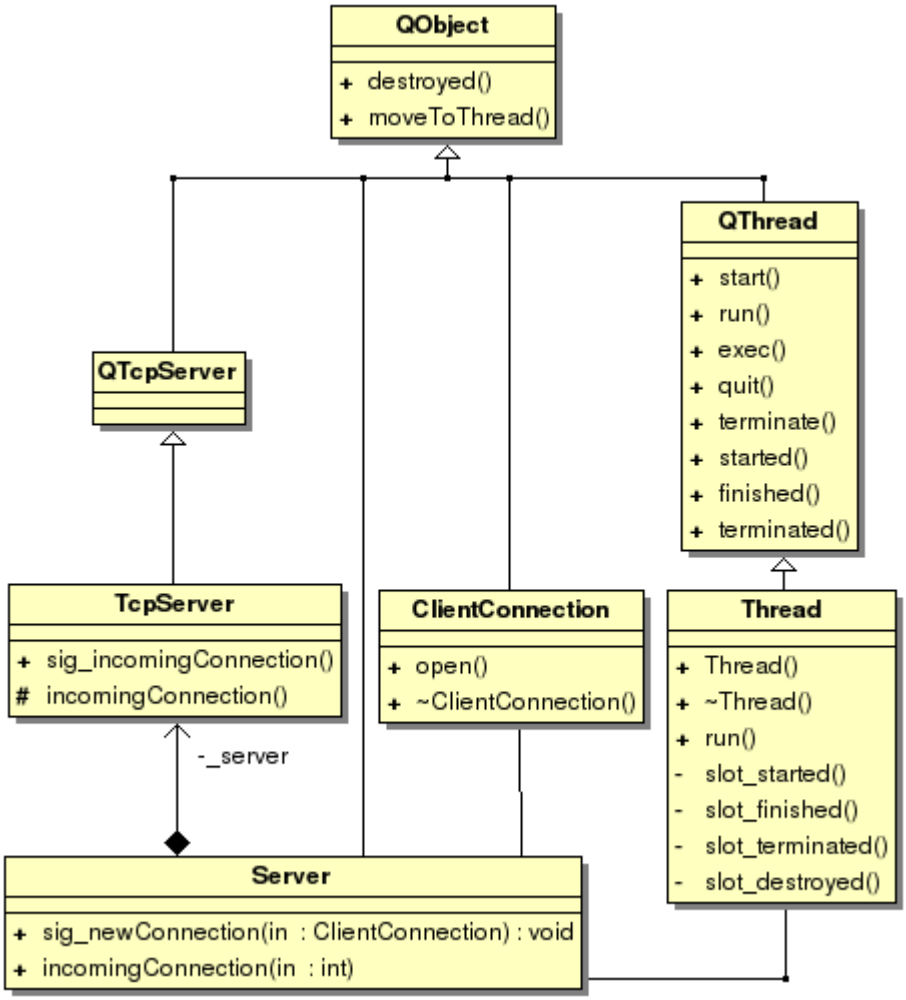
Partie personnelle

Fonctionnement MultiThread

A la connexion d'un nouveau client un nouveau QThread est créé dynamiquement.
Il lui est rattaché une instance de l'objet devant être exécuté dans un thread différent, ce qui est le cas de l'instance de la classe ClientConnection créée également à chaque nouvelle connexion d'un client.

Se déroule ensuite les connexions nécessaires au bon fonctionnement du ClientConnection et du Qthread :
le lancement du thread déclenchera l'initialisation de la connexion.
La déconnexion du ClientConnection entraînera sa mort, car une fois déconnecté la connexion est perdue définitivement, et si le même client se reconnecte, une nouvelle connexion sera ouverte : il faut donc libérer la mémoire.
La mort du ClientConnection entraînera l'arrêt du thread, puis sa mort.
Idem, inutile de conserver un thread sans aucune exécution à effectuer.

L'exécution du thread est alors déclenchée via l'appel de sa méthode start().

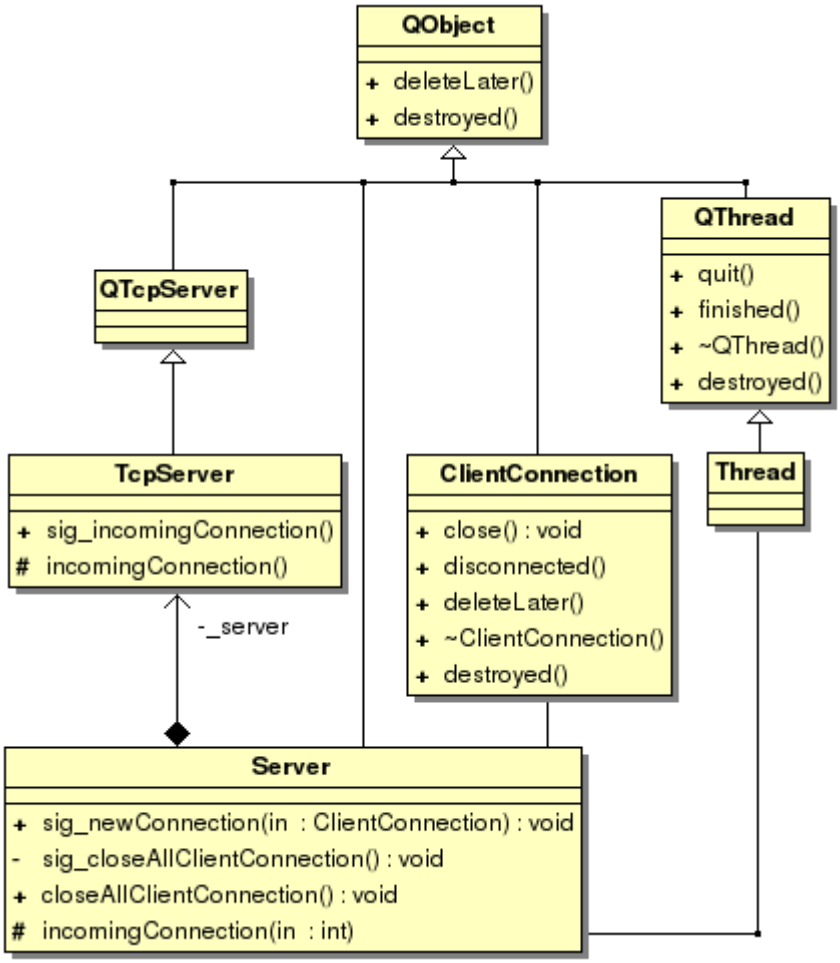


Projet SuPer – LSBB

Partie personnelle

Forcer l'arrêt des communications

Il est possible d'arrêter la communication avec un client en utilisant la méthode ClientConnection::close().
Cela a pour conséquence non seulement de stopper toute communication, mais aussi de libérer les ressources de cette connexion (toute déconnexion d'un client entraîne la mort de l'instance ClientConnection ainsi que de son thread hôte).
Forcer l'arrêt des communications avec tous les clients, et par conséquent libérer chaque connexion, peut être intéressant par exemple lorsqu'on souhaite quitter le programme.
Il est rendu possible via un signal de la classe Server, closeAllClientConnection().
A chaque instanciation d'un nouveau ClientConnection, le signal est connecté sur son slot close(), si bien que ce signal est perpétuellement connecté à tous les slots close() de toutes les instances de ClientConnection de l'application.
Une seule émission de ce signal suffit donc à forcer l'arrêt des communications et à libérer leurs ressources.
Ce signal est émis lors de l'exécution de son slot closeAllClientConnection().
L'IHM peut donc s'en servir pour libérer la mémoire avant de se fermer.

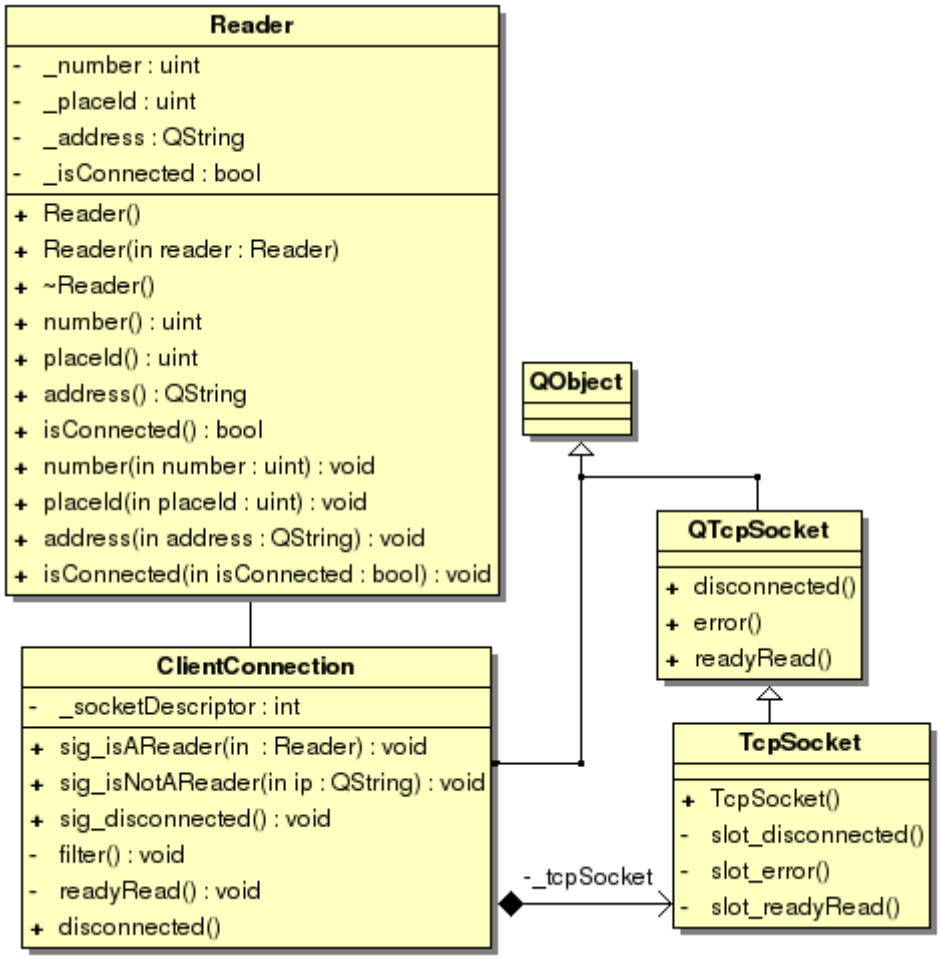


Projet SuPer – LSBB

Partie personnelle

Filtrage des clients

Le filtrage des clients est la première tâche effectuée lors de la connexion d'un nouveau client.
Elle consiste à obtenir l'ip de ce client et à consulter la base de données pour savoir si ce client est référencé comme étant un lecteur.
S'il ne l'est pas, le signal isNotAReader(QString) est transmis avec son adresse ip, et la connexion avec le client est aussitôt coupée.
S'il l'est, alors toutes les informations du lecteur sont récupérées depuis la base de données, puis sont stockées dans une classe Reader et sont enfin transmises via l'émission du signal isAReader(Reader).
Ce n'est qu'à partir de ce moment là que les données envoyées par le client sont lues et traitées.



Projet SuPer – LSBB

Partie personnelle

Bilan

Bilan :

La notion de threads était toute nouvelle pour moi, nous avions à peine commencé à les aborder en cours, il est donc normal que j'ai passé un temps conséquent dessus.
Cela a été ardu mais très enrichissant.
Je pense que cette tâche est la plus complexe parmi celles qui m'ont été attribuées.
Maintenant que le plus difficile est fait, le reste devrait aller beaucoup plus vite.
C'est la raison pour laquelle je suis optimiste quant à la charge de travail qu'il me reste à réaliser.

Projet SuPer – LSBB

Partie personnelle

Nicolas SCHERER

Revue 2

Projet SuPer – LSBB

Suivi de l'Avancement

Compte-rendu d'activités

Suivi du Projet		
Semaine	*Intervenant*	*Objet*
Du 04/03 au 08/03	Commun	→ mise en place gestionnaire de version (documentation)
	SCHERER Nicolas	→ accès base de données (BDD) → ajout dynamiquement du nombre d'onglets par rapport à la BDD (problème : sauvegarder des pointeurs Widgets des onglets) → validation « onglet dynamique » (onglets + légendes + images)
Du 11/03 au 15/03	SCHERER Nicolas	→ itération initialisation IHM dynamiquement OK → itération ajouter lecteur dynamiquement → conception des images → amélioration des fenêtres (ergonomie, boutons, vues)
Du 18/03 au 22/03	SCHERER Nicolas	→ superposition de Qlabel vs utilisation de scènes (documentation) → documentation sur Qt-IHM (beaucoup de lecture - apprentissage) → ajout d'un lecteur OK →BDD séparée de l'IHM

Projet SuPer – LSBB

Suivi de l'Avancement

Compte-rendu d'activités

Suivi du Projet		
Semaine	*Intervenant*	*Objet*
Du 25/03 au 29/03	SCHERER Nicolas	→ tests unitaires visualisation lecteur + affichage IHM dynamique → conception visualisation revue 2
Du 01/04 au 05/04	SCHERER Nicolas	→ conception visualisation revue 2 → avancement compte-rendu → codage traitement trame (début) → décodage trame → test « badge existe » → nouvelle personne (calcul de la position) (EN COURS)
Lundi 8 Avril	Revue n° 2	

Ce tableau récapitulatif ne représente que le travail effectué en heures de projet. Il est bien évident que beaucoup de travail est aussi fourni en dehors de ces heures là (notamment chez soi).

Projet SuPer – LSBB

Gestionnaire de Version

Mise en place

Nous avons utilisé Git. C'est un logiciel multi-plateforme (Windows, Linux...) de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, le créateur du noyau Linux.

La gestion globale du projet est amplement simplifiée, ainsi nous pouvons développer des nouveaux modules, appelés « branches » sans affecter nos autres modules. Nous pouvons aussi suivre l'avancement de notre projet, revenir en arrière si besoin, travailler chez nous...etc
Git va surtout nous simplifier le travail lors de l'intégration des modules développés chacun de notre côté ou en commun.

Nous utilisons aussi Github qui est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git.
Github propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets ouverts.

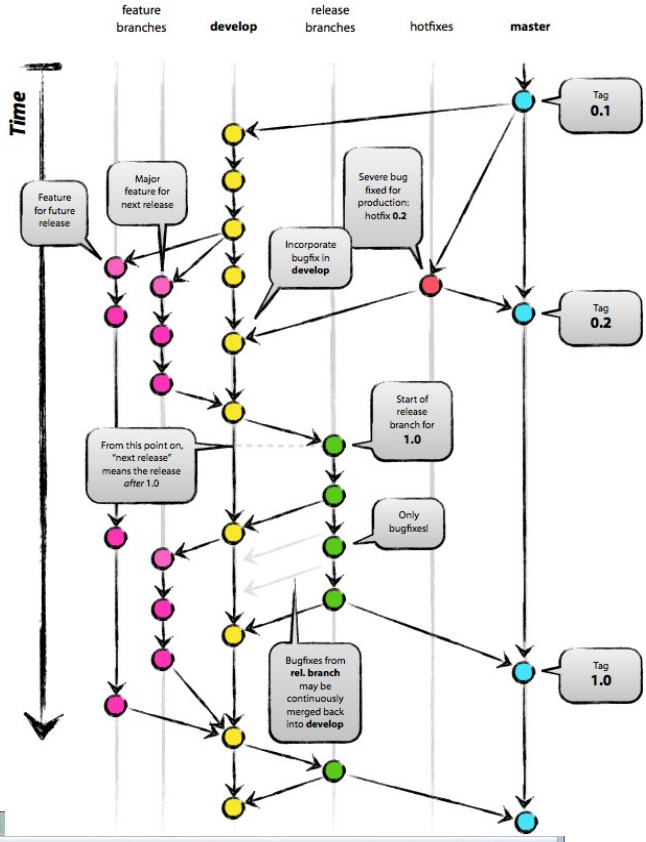
Grâce à ce système nous avons trois dépôts sur Github, un commun pour l'intégration finale, les deux autres pour chaque membre de l'équipe du projet. Ensuite les dépôts nous concernant peuvent être rapatriés sur plusieurs postes (au lycée, chez nous...) et nous pouvons travailler dessus, créer des nouvelles branches pour les différents modules...etc

Projet SuPer – LSBB

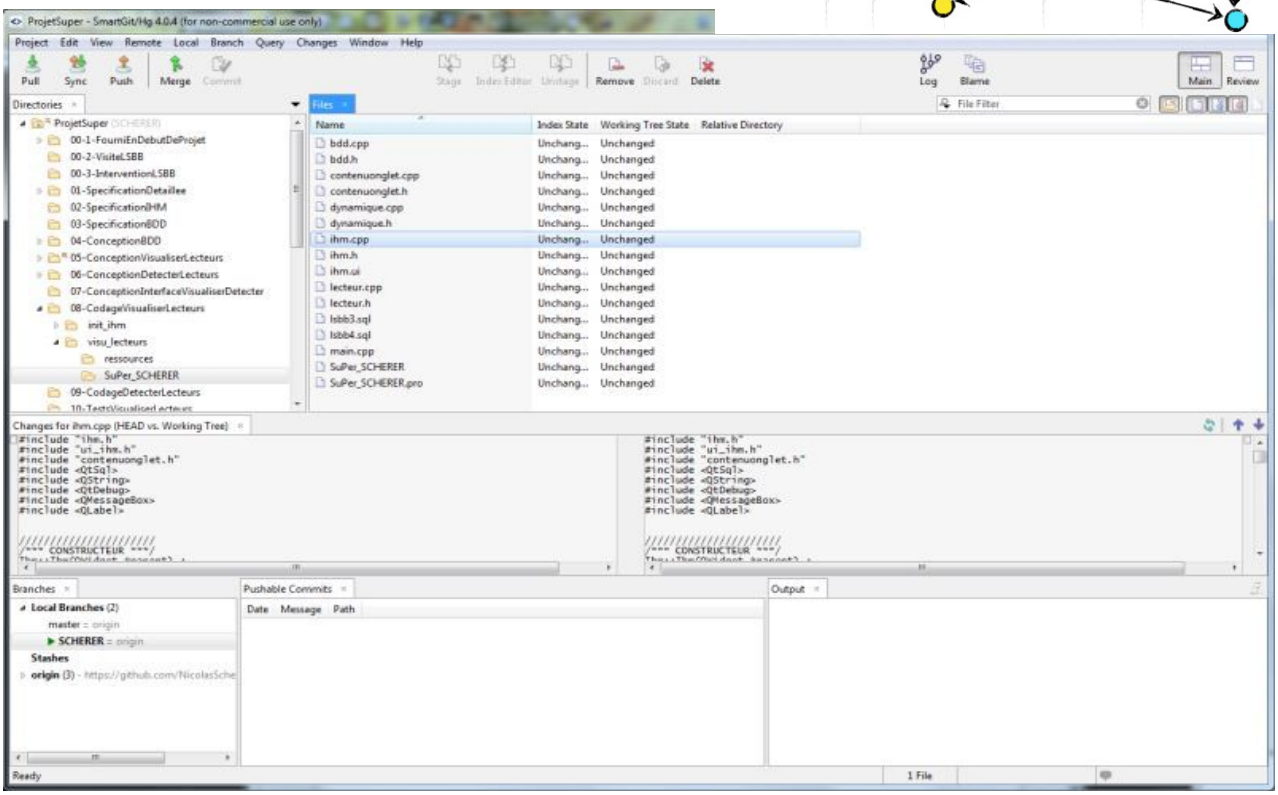
Gestionnaire de Version

Mise en place

Vous pouvez voir l'image ci-contre, qui illustre l'utilisation des différentes branches.



Brice JAMIN utilise Git directement dans un terminal en ligne de commande, et pour ma part j'utilise un logiciel « SmartGit » qui utilise les mêmes fonctions qu'en ligne de commande, mais je le trouve plus facile d'utilisation, plus rapide et plus agréable.



Projet SuPer – LSBB

Partie personnelle

Modifications apportées,
conception de visualiser itération 1

De nombreux changements ont eu lieu après la première conception que j'avais réalisée et présentée lors de la première revue, car de nouvelles idées sont venues et l'utilisation de Qt m'a montré parfois des solutions plus appropriées.

→ Premièrement, pour répondre à une demande du client, il a fallu revoir la façon d'ouvrir le logiciel. En effet le client souhaite que le logiciel ne soit pas une contrainte, c'est-à-dire une obligation de repasser par les développeurs dès qu'il y a un changement, par exemple lorsque l'on ajoute des lecteurs dans le tunnel. Nous avons donc mis en place un système étroitement lié à la base de données. Le client modifie uniquement la base de données et le logiciel au démarrage se met à jour en fonction. Tout est donc géré dynamiquement : le nombre de vues, d'images, de lieux, de lecteurs...

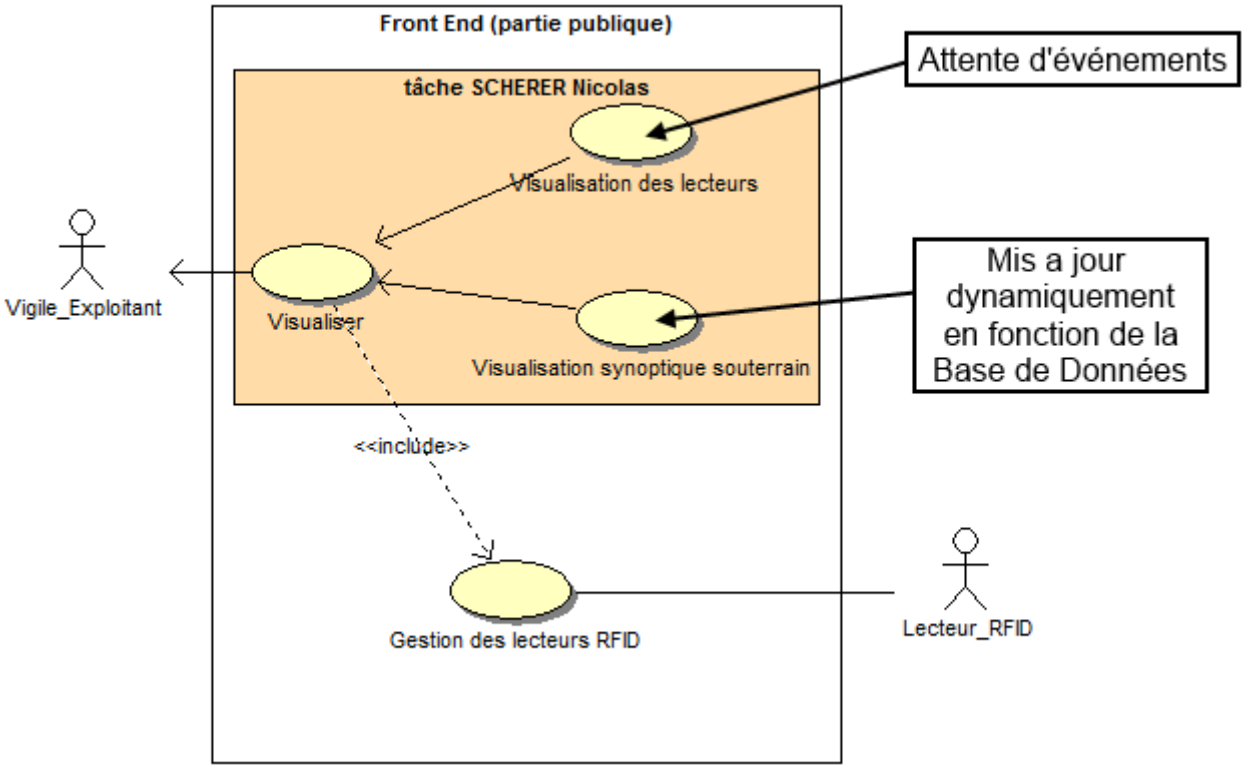


Diagramme des cas d'utilisation

→ Deuxièmement, l'affichage du lecteur s'effectue lorsque un lecteur se connecte. Mais lorsque un lecteur se déconnecte, il est plus simple de l'effacer et d'afficher dans une zone textuelle « le lecteur n°xx vient de se déconnecter ».

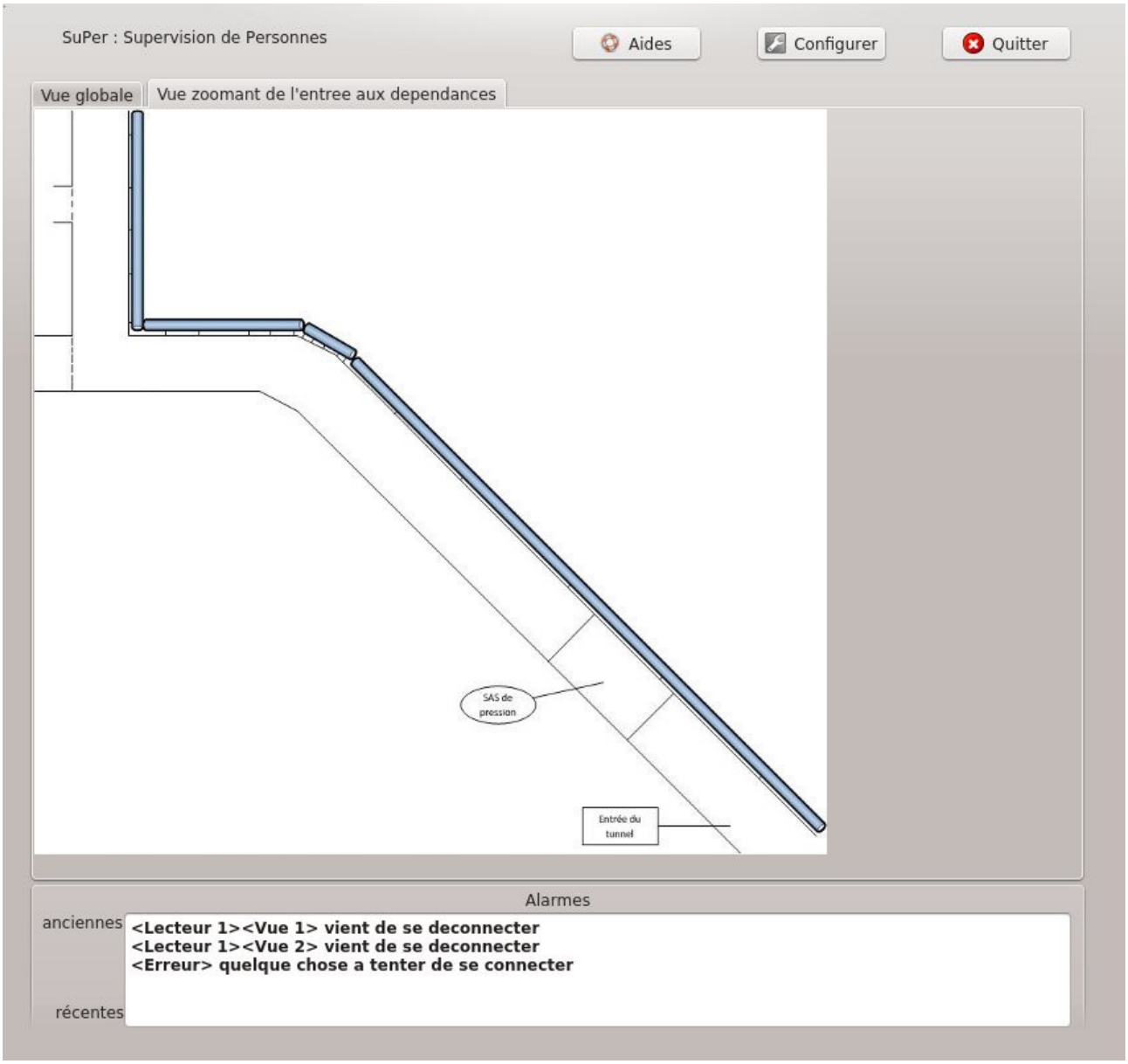
Projet SuPer – LSBB

Partie personnelle

Modifications apportées,
conception de visualiser itération 1

→ Troisièmement, l'affichage par défaut des vues de l'interface n'affiche pas la position des lecteurs. Il était prévu lors de ma première conception de placer les lecteurs en gris. Mais comme tout est fait dynamiquement en lien avec la base de données, cette position a été revue.

→ Enfin, j'ai ajouté une zone textuelle appelée « Alarmes » pour notifier à l'utilisateur des changements importants (intrus, lecteurs déconnectés, personnes inactives...).



Projet SuPer – LSBB

Partie personnelle

Codage initialisation IHM et visualisation lecteurs dynamiquement

J'ai codé en deux temps, premièrement avec l'accès à la base de données dans le code de l'IHM (interface homme-machine), c'est-à-dire dans le code principal, pour axer mes réflexions sur l'affichage dynamique : créer des onglets, des vues, des lecteurs dynamiquement.
Ensuite, une fois que cette partie était fonctionnelle, j'ai séparé le code de la base de données dans une classe séparée.

Voici la méthode d'ajout d'onglet appelée dès la création de l'IHM, s'il y a au préalable des vues (donc des onglets) entrées dans la base de données par le client.

```
/*** méthode AJOUT ONGLET ***/
    //nouveau onglet dynamique avec légende
    ContenuOnglet *pContenuOnglet = new ContenuOnglet(0, image);
    ui->tabWidget->insertTab(num_vue, pContenuOnglet, legende);

    //sauvegarde du pointeur onglet
    pDynamique->onglet[num_vue] = pContenuOnglet;
    qDebug() << "valeur dans la classe" << pDynamique->onglet[num_vue] << endl;
```

ContenuOnglet est une classe qui permet d'afficher dynamiquement le contenu d'un onglet, je me suis inspiré des exemples dans l'aide de Qt.
Il faut sauvegarder le pointeur (Widget) sur l'onglet pour l'ajout d'éléments sur la vue par la suite (ajout lecteurs, personnes...etc). Il est ajouté à un attribut de la classe Dynamique.

Après, je me suis penché sur l'affichage des lecteurs :

```
/*** méthode AJOUT LECTEUR ***/
    //se placer sur le bon onglet
    QWidget *onglet;
    onglet = pDynamique->onglet[num_vue];
    //test valeur
    qDebug() << "valeur pointeur onglet" << onglet << endl;

    //nouveau label dynamique pour mettre l'image correspondant
    QLabel *labelL = new QLabel(onglet);
    //image utilisé
    labelL->setPixmap(QPixmap("../ressources/lecteur_actif.jpg"));
    labelL->setGeometry(x, y, 15, 42); // largeur hauteur à définir

    //sauvegarde du pointeur du label du lecteur
  pDynamique->labelL[num_vue][numLecteur] = labelL;
```

Projet SuPer – LSBB

Partie personnelle

Codage initialisation IHM et visualisation lecteurs dynamiquement

Il faut toujours sauvegarder le pointeur du label (QWidget) de lecteur, car sachant que tout est fait dynamiquement, il faut absolument une trace pour y avoir accès si l'on souhaite supprimer ce lecteur par exemple.

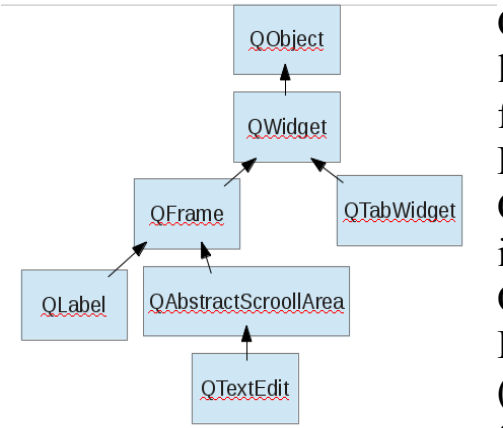
Toutes ces méthodes sont appelées par des slots, issus d'événements pour respecter le cahier des charges.

La complexité relative de l'utilisation de Qt pour l'affichage dynamique m'a demandé beaucoup d'apprentissage et de documentation sur les différentes solutions.

De base nous utilisons « Qt Designer » pour faire nos fenêtres en fixe, et jusqu'à maintenant nous ne voyions pas tout le code que cela entraîne par derrière.
J'ai donc dû regarder beaucoup d'exemples sur la documentation officielle pour simplement comprendre comment se crée certain Widget.

(documentation Qt)
La classe QWidget est la classe de base de tous les objets d'interface utilisateur.

Le widget est l'atome de l'interface utilisateur : il reçoit des événements de souris, de clavier et d'autres depuis le système de fenêtrage, et peint une représentation de lui-même à l'écran. Tous les widgets sont rectangulaires et sont triés dans un z-ordre. Un widget est coupé par son parent et par les widgets se trouvant devant lui. Un widget qui n'est pas embarqué dans un widget parent est appelé une fenêtre.



Ce diagramme des classes de Qt, représente l'arborescence (ici l'UML n'est pas respecté, les flèches ne devrait pas être pleines).
Par exemple pour ajouter une image, j'utilise QLabel qui hérite de QFrame puis QWidget, cette image qui va s'ajouter sur un onglet : QTabWidget qui hérite de QWidget.
Enfin la gestion des événements (connect/slot) (non représentée sur ce diagramme) hérite de Qobjet.

Projet SuPer – LSBB

Partie personnelle

Choix de codages – Séparation BDD

Lors du développement d'un logiciel important, pratiquement quel que soit son domaine d'application, le paradigme Modèle/Vue/Contrôleur s'impose de lui-même pour réaliser une modélisation maîtrisable du problème à traiter. Depuis la version 4, Qt propose des solutions pour sa mise en œuvre avec l'environnement Graphics View.

Malgré que cet environnement soit intéressant à mettre en place et donne une autre dimension, je ne m'y suis pas attardé.
J'avais beaucoup de retard sur le plan prévisionnel, et j'ai donc décidé de « simplement » utiliser une superposition de plusieurs QLabel, pour former ce dont j'avais besoin. C'est-à-dire une image de fond de la vue, puis des lecteurs fixes et enfin des personnes qui bougent (fait à l'itération deux).

Dès ces fonctionnalités mises en œuvre, j'ai pu séparer l'accès Base de données du reste du code, dans une classe appelée bdd.
Cette classe comporte toutes les méthodes permettant de faire des requêtes sur la base de données.

Enfin pour gérer le retour de certaines méthodes, j'ai mis en place des structures de données et des listes (QList).
Exemple :

```
/*** METHODE OBTENIR VUE et POSITION en FONCTION du n° LECTEUR ***/
bool Bdd::getVuePosFctLect(int numLecteur, QList<T_TupleLecteurE> *listeLecteur)
{

    //avec le numéro obtenu, obtenir la vue et la position (x,y)
    [...]
    //allocation pointeur
    this->pTupleLecteurE = new T_TupleLecteurE();

    //réponse requête
    while(query->next()){
         int num_vue = query->value(1).toInt();
         [...]
        //ajout sur liste
         this->pTupleLecteurE->num_vue = num_vue;
         [...]
         listeLecteur->append(*pTupleLecteurE);
}
```

Projet SuPer – LSBB

Partie personnelle

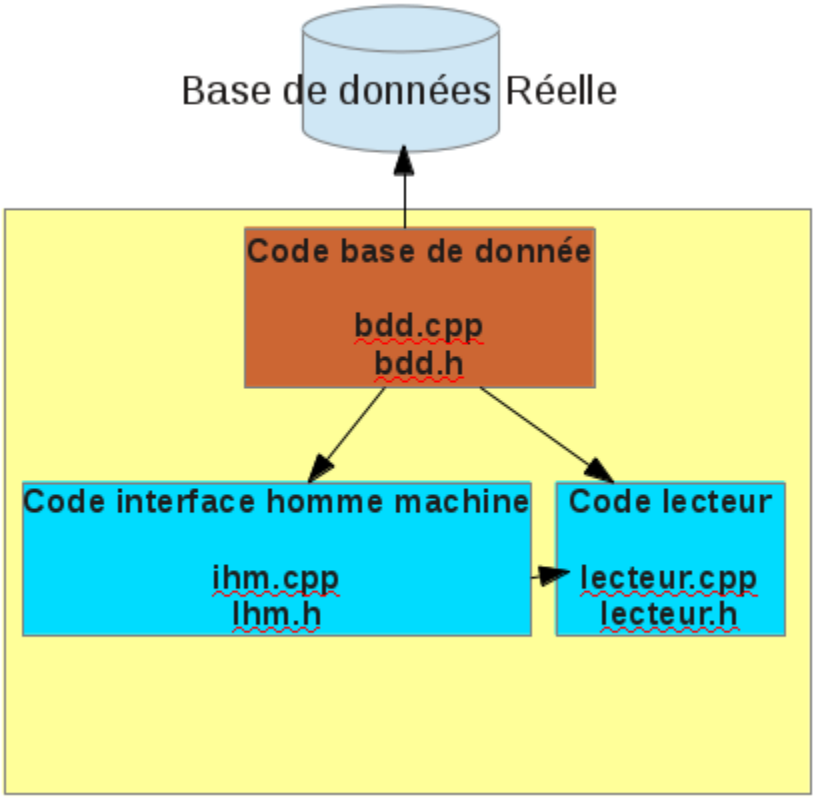
Séparation BDD

```
    delete this->pTupleLecteurE;

    return true;
}

//enregistrement d'un lecteur qui se connecte
typedef struct s_TupleLecteurE {
    int num_vue;
    int x;
    int y;
} T_TupleLecteurE;
```

Les déclarations des QList se font dans le code de l'IHM (code principal) pour éviter la recopie en mémoire. Il y a un échange d'adresses de QList.



Interaction entre les différentes zones de codes, seulement le code de la base de données à un accès réel avec la base de données.

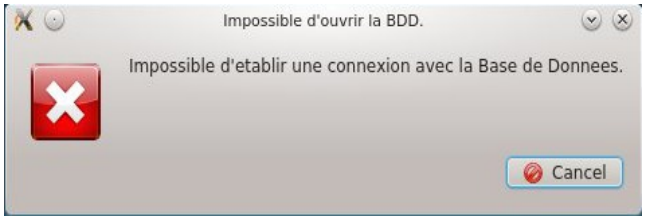
Projet SuPer – LSBB

Partie personnelle

Scénarios d'erreurs – initialisation logiciel

Au niveau du scénario d'erreur, s'il y a un problème avec la base de données : que cette dernière soit inaccessible et/ou qu'elle soit vide. L'IHM doit réagir en fonction :

→ une boite de dialogue s'ouvre au démarrage du logiciel pour indiquer l'impossibilité de se connecter



→ le logiciel se met dans une posture d'assistance (onglet avec explications)

Projet SuPer – LSBB

Partie personnelle

Plan et tests unitaires de Visualiser
(IHM, visualisation lecteurs)

Après la conception et le codage de la tâche Visualiser (IHM, visualisation lecteurs) (T103 et T104 ; se référer à la Liste des Tâches pages 18 du Dossier Technique Revue n°1), il est nécessaire de faire des plans et tests unitaires sur cette portion de code appelée aussi module.

Le test unitaire est un procédé permettant de s'assurer du fonctionnement correct du module.

Grâce à la spécification et à la conception, notamment lors de la mise en place des scénarios nominaux, alternatifs et d'erreurs, il est assez simple de définir un plan de test.

Résultats des Tests unitaires :

→ de la tâche Visualiser (visualisation lecteurs)

→ de son sous-ensemble (initialisation de l'interface dynamiquement)

Effectués durant la semaine du 18 au 22 Mars 2013.

Scénario	Résumé de l'état	Résultats attendus	Test Réussis
Scénario Nominal	Lancement du logiciel	Mise à jour dynamique des vues, images... en fonction de la base de données	Oui
Scénario Nominal	Quitter le logiciel	En cliquant sur le bouton « Quitter » le logiciel se ferme normalement	Oui
Scénario Nominal	Un lecteur se connecte	Affichage du lecteur en Vert sur toutes les vues	Oui

Projet SuPer – LSBB

Partie personnelle

Plan et tests unitaires de Visualiser
(IHM, visualisation lecteurs)

Effectués durant la semaine du 18 au 22 Mars 2013.

Scénario	Résumé de l'état	Résultats attendus	Test Réussis
Scénario Nominal	Un lecteur se déconnecte	Suppression du lecteur sur toutes les vues, affichage textuel d'Alarmes : (n° lecteur, n° vue vient de se déconnecter)	Oui
Scénario d'Erreur	Un intrus a essayé de se connecter	Affichage textuel Alarmes : quelque chose a tenté de se connecter	Oui
Scénario d'Erreur	Accès à la Base de données impossible	Fenêtre d'avertissement, ouverture logiciel avec explications du problème et solutions possibles	Oui
Scénario d'Erreur	Base de données vide (informations sur les vues notamment)	Ouverture logiciel avec explications du problème et solutions possibles	Oui

Tests unitaires validés le

Projet SuPer – LSBB

Partie personnelle

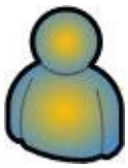
Spécification Interface Homme-Machine
itération 2

<u>But et objectifs :</u>

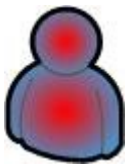
Le but de l'itération 2 est de visualiser à l'écran une personne, en mouvement ou non, son sens de passage, ainsi que son identité.
Ce qui entraîne pour l'IHM plusieurs objectifs :
→ quelle personne (identité)
→ personne en train de « monter » dans le tunnel (sens de passage)
→ personne en train de se diriger vers la sortie du tunnel (sens de passage)
→ personne en mouvement
→ personne en danger (sans mouvement)
→ personne invisible (non réception du badge)
→ localisation personne

Nous utilisons toujours l'identification par le biais d'images :

personne active



personne invisible



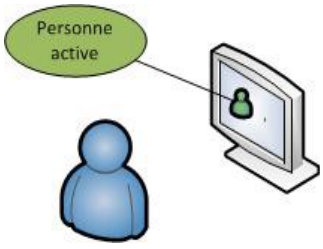
personne en danger

sens de passage (haut /bas)

La personne sera probablement identifiée dans une fenêtre contenant son identité lors du passage de la souris sur son image.

<u>Récapitulatif des codes de l'interface Homme-Machine (Itération 2) :</u>

La personne est représentée par une image d'un personnage (voir ci-dessus) de différentes couleurs selon l'état où elle se trouve. A chaque changement d'état une notification apparaîtra dans le champ Alarme déjà mis en place à l'itération précédente. Enfin le sens de passage sera indiqué à l'aide des images en forme de flèche (voir ci-dessus).



Projet SuPer – LSBB

Partie personnelle

Organisation des Données – itération 2
Spécification de la Base de données

Pour la revue deux nous avons besoin de :

Personne :

(primaire)[entier] num_pers	→ numéro de la personne
[chaîne de caractères] nom	→ nom de la personne
[chaîne de caractères] prenom	→ prénom de la personne
[chaîne de caractères] societe sur le site	→ société de la personne intervenant
[date] dateDebut	→ date de début d'intervention
[date] dateFin	→ date de fin d'intervention

Badge :

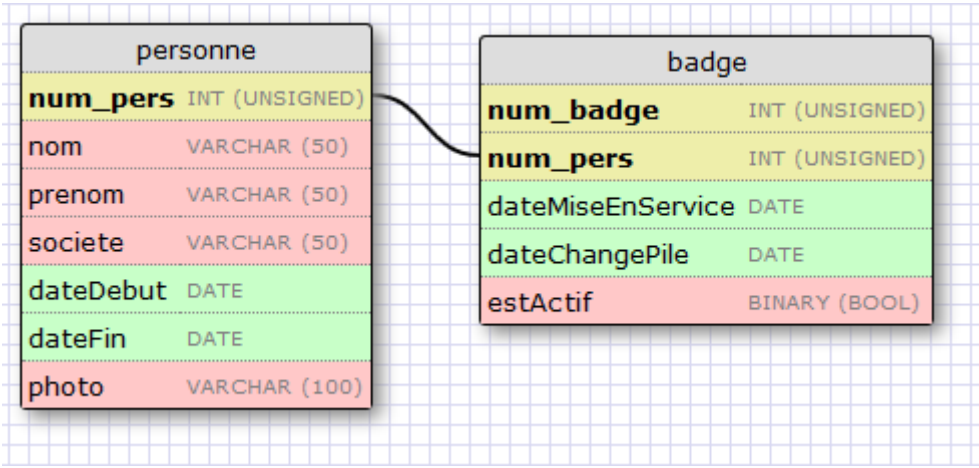
(primaire)[entier] num_badge	→ numéro du badge
(étrangère)[entier] num_pers	→ numéro de la personne
[date] dateMiseEnService	→ date de mise en service du badge
[date] dateChangePile	→ date probable de changement de la pile
[booléen] estActif	→ état du badge (en activité ou non)

Modèle de Base de données (SQL Designer) :

Légende :

en gras : clé primaire /étrangère

liaison : connexion clé étrangère/primaire

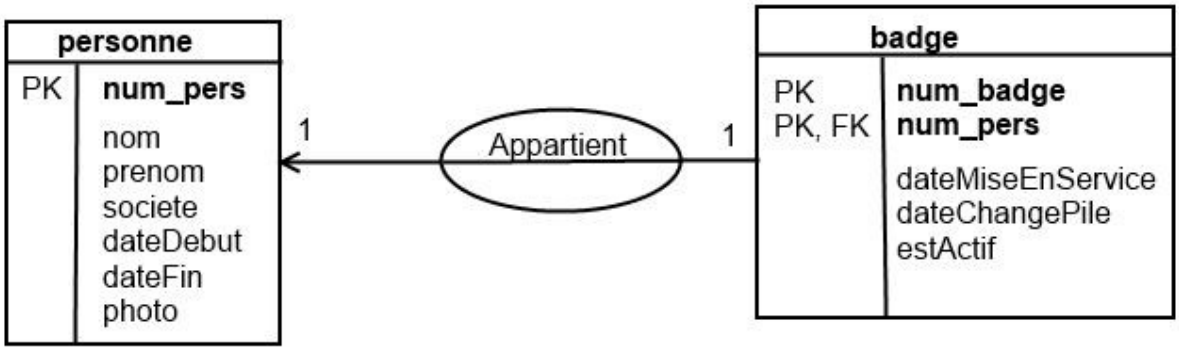


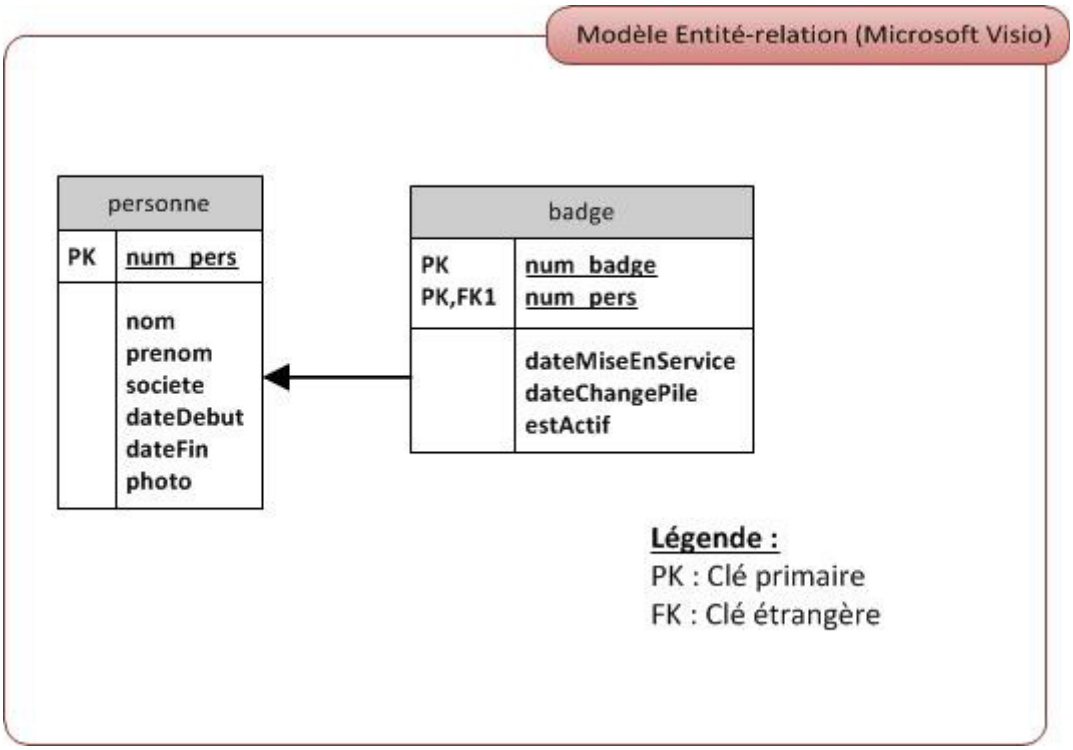
Projet SuPer – LSBB

Partie personnelle

Organisation des Données – itération 2
Spécification de la Base de données

Modèle Entité-Association (fait à la main) :





Projet SuPer – LSBB

Partie personnelle

Conception de visualiser itération 2

Il s'agit d'animer le synoptique avec la visualisation d'une personne (positionnement, sens de marche, mouvement, activité) et d'afficher les informations la concernant pour l'itération deux.

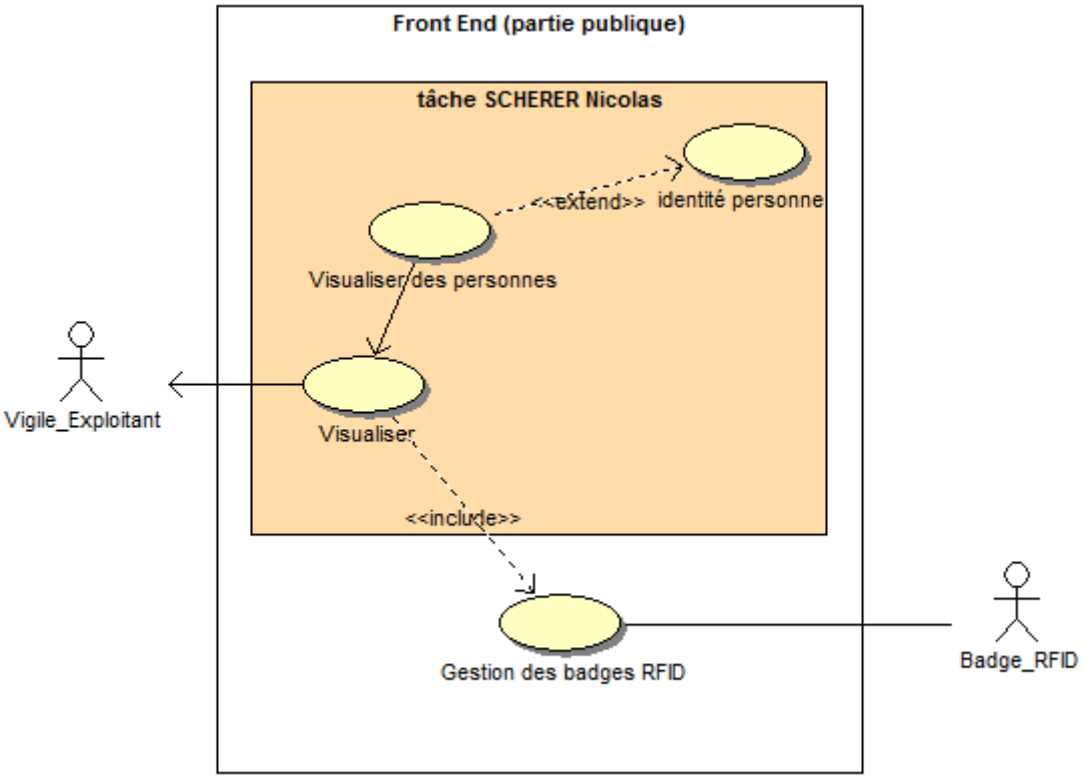


Diagramme des cas d'utilisation

<u>Scénario nominal :</u>

<u>Pré-condition :</u> Réception d'un événement indiquant un badge (avec trame bien formatée)

→ Capter l'événement

→ obtenir la trame (chaîne de caractères envoyée en argument de méthode)

→ décoder la trame (séparation des parties de la trame –n° badge, niveau de réception, mouvement, n° lecteur–)

→ test si le badge existe dans la base de données

→ test si le badge existe déjà sur l'IHM

Projet SuPer – LSBB

Partie personnelle

Conception de visualiser itération 2

→ gestion du mouvement (→ personne en danger)
→ gestion de la réception (→ personne invisible)
→ rechercher identité de la personne
→ chercher position pour affichage ou déplacement
→ sauvegarder historique (dans « log »)

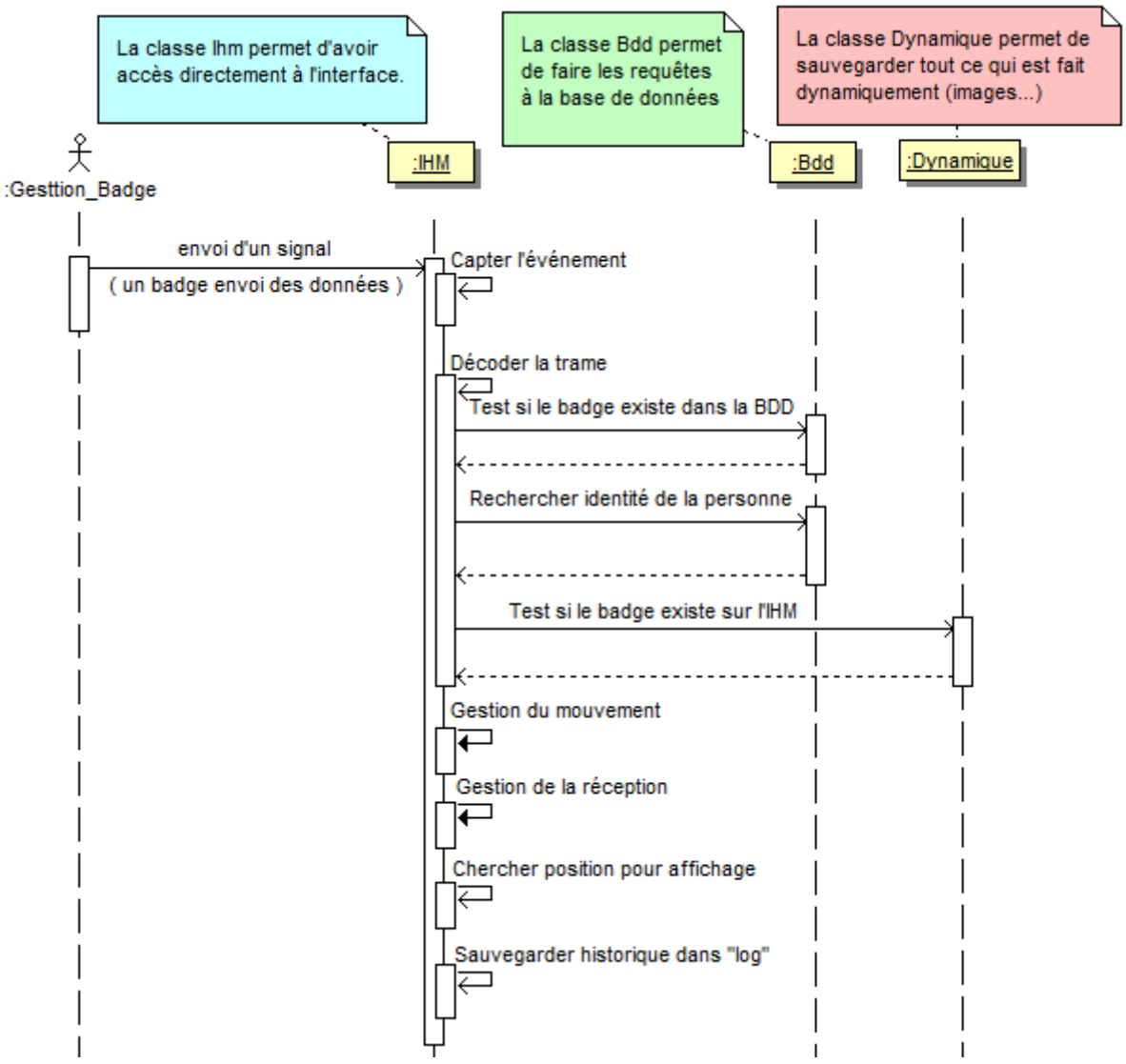


Diagramme de séquence : scénario nominal (conception préliminaire)

Projet SuPer – LSBB

Partie personnelle

Conception de visualiser itération 2

Scénario alternatif :

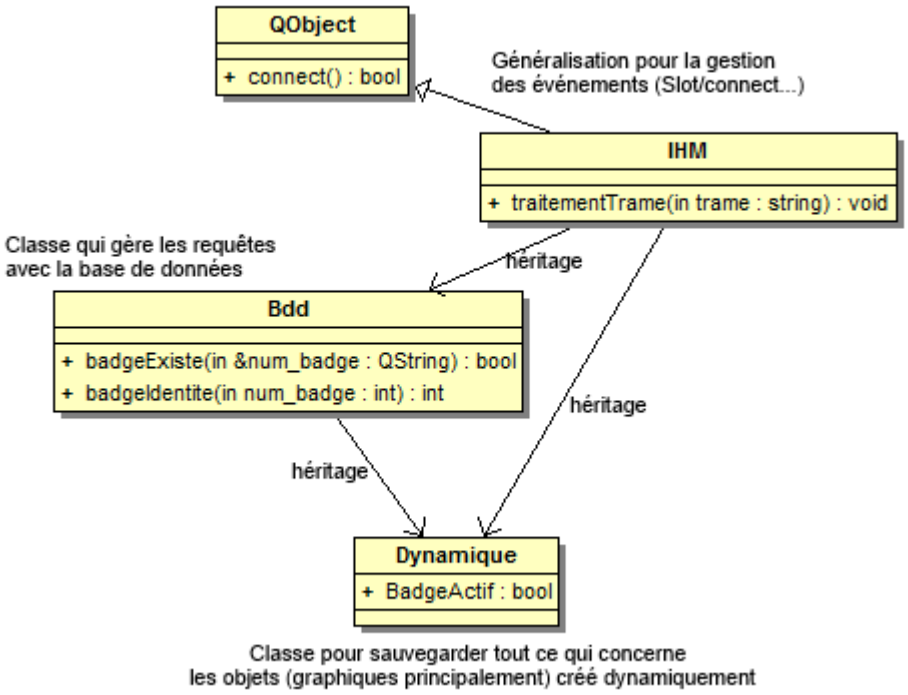
Pré-condition : Réception d'un événement indiquant le déplacement de la souris sur un widget « personne »

→ Capter l'événement

→ faire apparaître l'identité de la personne (à préciser ; piste possible : affichage dans une nouvelle fenêtre, dans un tableau, dans la ligne Alarme...)

Ce scénario n'a pas été traité en profondeur jusqu'à maintenant, dans le prochain dossier technique je notifierais l'avancée.

Il y a une différence notable avec cette itération, en effet je fais du code, je me documente en fonction et je réfléchi sur la conception par tâtonnement.



Extrait du diagramme des classes (itération 2), ***codage et analyse en cours...***

Projet SuPer – LSBB

Partie personnelle

Début de codage visualiser itération 2

A la première itération, j'avais effectué une analyse (théorique) puis lorsque j'ai programmé, mon analyse de départ ne correspondait plus vraiment avec les outils que l'on peut utiliser.
Alors pour éviter ce scénario, j'ai décidé d'avancer plus habilement, j'analyse le besoin, je fais une conception préliminaire sur une partie de la tâche, je teste avec Qt (cela s'appelle aussi les mises en œuvres) puis j'approfondis mon analyse dès que le besoin s'en fait sentir (conception détaillée).
Ainsi j'avance avec une analyse qui correspond vraiment aux outils utilisés.

Précision sur le type de trame reçue :

[ADD016A701]

- [et] délimiteur de trame
- AD est le niveau de réception « RSSI » du badge
- D01 est l'identifiant du badge codé sur 12 bits
- 6A7 est le niveau de mouvement mesuré
- 01 est l'identifiant du lecteur codé sur 8 bits

Tout est en hexadécimal.

```
//--décodage trame
QString num_badge, sens, mouvement, num_lecteur;

//séparation des parties de la trame
num_badge = trame.mid(2,3); //numéro de badge
sens = trame.mid(0,2); //niveau de réception du tag
mouvement = trame.mid(5,3); //niveau de mouvement mesuré
num_lecteur = trame.mid(8,2);   //numéro du lecteur

//conversion de l'hexadécimal en décimal
int num_badge_i = num_badge.toInt(0,16);
int sens_i = sens.toInt(0,16);
int num_lecteur_i = num_lecteur.toInt(0,16);
//test si le badge existe dans la BDD
if(!pBdd->badgeExiste(num_badge)){
        [...]
        return false;
    }
```

[…] Suite à la prochaine revue !

Projet SuPer – LSBB

Partie personnelle

Bilan et Conclusion

Bilan :

Encore beaucoup de retard sur le prévisionnel, mais il y a eu quand même des réalisations faites depuis la dernière revue.
Notamment il a fallu se documenter sur différents outils de Qt (comment fonctionnent des objets graphiques créés dynamiquement ?) , et cela prend énormément de temps !

Conclusion :

Même s'il y a du retard, je pense que prendre son temps pour bien réaliser les tâches, bien se documenter sont enrichissants au niveau de l'apprentissage dans notre cursus.