2011-2012

Conception et constitution d'une base de données de scans laser terrestres et de données forestières de validation.

Maillet Cédric

Licence Professionnelle S.I.G. - Université de LA ROCHELLE

Maître de stage : Alexandre PIBOULE, chargé de recherche

ONF, pôle R&D de Nancy, Velaine-en-Haye

Conception et constitution d’une base de données de scans laser terrestres et de données forestières de validation.

Ce stage a été subventionné par le projet EMERGE, qui consiste en l’élaboration de modèles pour une estimation robuste et générique du bois énergie au niveau arbre. Ce projet est mis en place par l’ANR (agence nationale de la recherche).

Remerciements

Je tiens particulièrement à remercier mon maitre de stage Alexandre Piboule, pour son aide tout au long du stage et le temps qu’il a dépensé pour m’aider dans la réalisation du stage, mais aussi dans la rédaction du rapport et la préparation de l’oral.

Je tiens aussi à remercier Anne Jolly pour les conseils donnés dans la rédaction de mon rapport et sur l’analyse de mon plan.

Je remercie également Jean-Pierre Renaud pour l’aide sur la réalisation de la base de données avec PostgreSQL, qu’il m’a apporté.

Je tiens à remercier aussi tous les membres du Pôle R&D de Nancy ainsi que les stagiaires y ayant fait un séjour, pour m’avoir accueilli et pour leur bonne humeur qui m’ont permis de passer un bon stage au sein du pôle.

Conception et constitution d'une base de données de scans laser terrestres et de données forestières de validation.

Table des matières

Introduction

J'ai réalisé mon stage de fin d'étude en licence professionnelle SIG de 4 mois, au pôle recherche et développement de Nancy de l'Office National des Forêts. Ce service a pour thèmes de recherche la télédétection et le changement climatique.

Récemment le service a acquis un scanner lidar terrestre, pour permettre d'estimer dans un peuplement forestier le volume de bois, la surface terrière ainsi que d'autres données de référence utile au gestionnaire de la forêt. Depuis l'achat, le service acquière de nombreux scans et données de validation.

Mon travail consistait à réaliser une base de données permettant de croiser les données des scans laser avec les données terrain de validation. Cette base devait être géographique étant donné que les arbres, les placettes et les scans sont localisés. L'objectif de cette base est de pouvoir recenser les données disponibles ou à acquérir, ainsi que de faire le lien automatiquement entre données numériques et terrain.

Pour cela j'ai réalisé une interface en C++ à l'aide du Framework Qt, qui permet d'accéder à chaque table, ajouter des données soit ligne par ligne, soit par un import. Mais qui permet aussi de réaliser des requêtes, des exports et de recalculer des coordonnées absolues, pour les arbres qui dont les coordonnées mesurées sont relatives au centre de la placette.

J'ai aussi réalisé un plugin Qgis, également en C++ avec le Framework Qt, qui permet de visualiser les données géographiques stockées dans la base de données, mais aussi de réaliser des cartes automatisées pour chaque placette avec une visualisation des essences et du diamètre pour les arbres.

I. Contexte du stage

A. Présentation de l'entreprise

1. Office National des Forêts

L'**Office national des forêts** (**ONF**) est l'établissement public français chargé de la gestion des forêts publiques, placé sous la tutelle du ministère de l'Agriculture et de la Pêche et du ministère de l'Écologie, de l'Énergie, du Développement durable et de l'Aménagement du Territoire. La Direction générale est basée à Paris, avenue de Saint-Mandé.

L'ONF est un établissement public à caractère industriel et commercial disposant de l'autonomie de gestion depuis sa création en 1964. Sa création s'inscrit dans la continuité de l'histoire du service forestier public en France. L'ONF a donc en partie succédé à l'Administration des Eaux et Forêts créée en 1291 par le roi Philippe le Bel.

L'ONF assure quatre missions principales : la production de bois, l'accueil du public, la biodiversité et la protection du territoire et de la forêt. Il a également une activité de prestataire de services pour la gestion et l'entretien des espaces naturels.

L'ONF gère au total 12 250 000 ha de forêts publiques dont 4 500 000 ha en France métropolitaine et 7 750 000 ha dans les départements d'outre-mer, pour l'essentiel en Guyane française. Les forêts domaniales gérées par l'ONF sont toutes certifiées PEFC. L'ONF gère également des écosystèmes associés à la forêt tels que tourbières, dunes, pelouses alpines, pour une surface de 534 000 ha.

L'ONF est organisé sur le terrain depuis le 1er janvier 2009 en :

- 9 directions territoriales (on peut voir la répartition sur la carte 1), elles-mêmes subdivisées en 60 agences,
- 5 directions régionales (Corse et outre-mer) qui assurent à la fois les attributions habituelles des directions territoriales et des agences,

Chaque agence est composée d'une direction et de services. Les forêts gérées par l'O.N.F. en France métropolitaine sont réparties dans 368 Unités Territoriales elle mêmes divisées en 3 000 secteurs sous la responsabilité des Agents patrimoniaux.

Carte 1 : Direction Territoriale, source : ONF

2. Département recherche et développement

Les Pôles recherche et développement sont tous rattachés au département R&D de la direction technique et commerciale bois. Il y a 10 pôles dans le département recherche et

développement, chacun ayant une problématique particulière. On peut voir l'organisation des pôles sur l'organigramme de la figure 1.

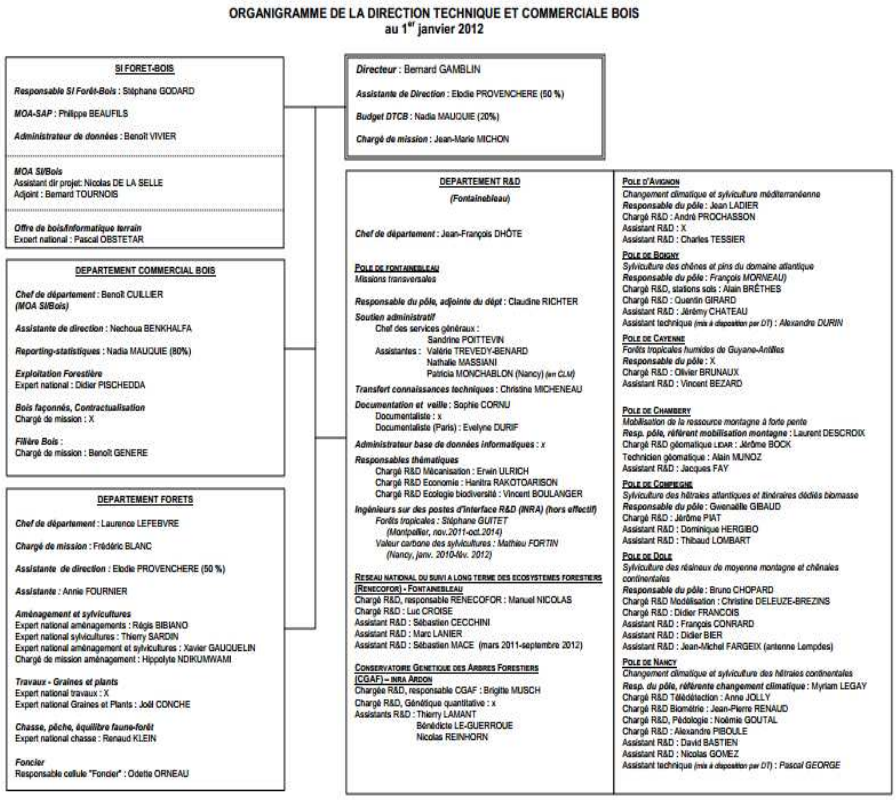


Figure 1 : Organigramme, source : ONF

Le Pôle de Nancy où j'ai réalisé mon stage a pour responsable Myriam Legay, spécialiste sur les questions de changement climatique. Le service est composé de 8 personnes ce qui en fait le plus gros pôle de recherche de l'ONF à l'exception du pôle de direction de Fontainebleau.

La spécialisation du service est la prise en compte des changements climatiques dans la gestion forestière. Un autre point important dont s'occupe le service, est l'utilisation de la télédétection (Lidar aérien et Lidar terrestre) comme outils d'aide à la prise de décision sylvicole (en particulier les inventaires d'aménagement).

B. Le cadre du projet

1. Le Lidar Terrestre

Le T-Lidar (Terrestrial Light Detection And Ranging), ou scanner laser terrestre, permet la numérisation d'un espace par balayage d'un rayon laser infrarouge émis dans quasiment toutes les directions grâce aux rotations de l'appareil.

Après rencontre avec un objet, le rayon incident est rétrodiffusé vers le récepteur du Lidar. La distance de l'objet est mesurée selon une technique de décalage de phase entre le faisceau laser émis et celui rétrodiffusé. En y associant la direction visée par l'appareil, on obtient les coordonnées en 3 dimensions de l'obstacle rencontré.

Une réflectance, indicateur de l'intensité avec laquelle le signal revient, est également mesurée et attribuée à chaque point. Une image en 3 dimensions de la scène est ainsi obtenue. On peut voir un aperçu du résultat d'un scan sur la figure 2.

Figure 2 : Scan T-Lidar

2. Démarche d'inventaire

Pour la réalisation de scan sur une placette, le scanner est normalement situé au centre de la placette. Mais le scanner peut-être décalé dans le cas d'un obstacle, présence d'un arbre, roncier imposant … Il peut aussi y avoir du multi-scan c'est-à-dire qu'une session de scan va être composée de plusieurs scans. Pour cela sur le terrain vont être disposés des sphères blanches qui vont permettre de fusionner les scans.

En plus des scans sur le terrain on va réaliser, si possible en même temps que les scans, un inventaire terrain sur la placette dont la position géographique est calculé précisément. Les placettes peuvent être relascopiques, rectangulaires, circulaires et il peut y avoir une ou deux placettes car l'une peut avoir un diamètre de précomptage différent de l'autre.

Les arbres peuvent être mesurés par un, deux diamètres, ou une circonférence, ainsi qu'une hauteur. Cela sera défini dans le protocole d'inventaire. Les arbres vont être localisés de façon relative par rapport au centre de la placette en prenant leur azimut et leur distance. De plus l'essence de chaque arbre sera notée par un code ONF.

Il faut savoir que suivant les marques et les modèles de scanner, des filtres matériels ou logiciels pourront être appliqués. De plus, des paramètres différents pourront être appliqués sur la même placette, la vitesse du scan, le nombre de points… Les scans peuvent être pourvus d'une boussole, d'un inclinomètre, d'un GPS …

Les scans pourront ensuite être fusionnés, filtrés. De plus, à l'aide du logiciel COMPUTREE développé par le département R&D de l'ONF, on pourra appliquer différents algorithmes et ainsi avoir la position de chaque arbre et calculer des données dendrométriques[1] et les comparer ensuite aux données terrain de référence.

Il y a trois types d'opérateurs qui seront enregistrés, les opérateurs de terrain qui ont fait les scans, mais aussi ceux qui auront réalisé l'inventaire des arbres. Ainsi que les opérateurs informatiques qui auront réalisé les traitements informatiques. Le fait de noter l'opérateur permet en cas de problème de trouver plus facilement ce qui s'est passé.

3. Inventaire de l'existant

Les données étaient jusqu'à présent rangées sur un disque partagé en réseau local. Les données sont décomposées en dossiers par sites, puis dans chaque site il y a un dossier pour les scans et un dossier pour les données terrain de validation.

Le problème c'est que les données sont consultables mais ne sont ni organisées de façon à pouvoir tout interroger en une seule fois, ni indexées. De plus, différent types de fichiers existent suivant qu'ils ont été traités ou non (.xyb, .fls).

La solution que nous allons mettre en place ne résout pas tous les problèmes, mais en corrige certain, l'interrogation des données sera résolue ainsi que la sauvegarde des données de validation terrain. Pour la sauvegarde des scans il faudrait soit les mettre dans une base de données PostGIS soit sur un serveur qui serait sauvegardé régulièrement. La possibilité d'un serveur pour tous les pôles R&D est en discussion.

4. Objectifs

Mon maitre de stage Alexandre Piboule est spécialiste quant à lui du Lidar Terrestre et est chargé de la coordination du développement d'un logiciel collaboratif COMPUTREE, qui permet grâce aux scans de Lidar Terrestre de détecter les arbres dans un nuage de point généré, et ainsi pouvoir réaliser des mesures dendrométriques variées. Plusieurs plugins ont été réalisés par des partenaires et mutualisent ainsi les ressources.

L'acquisition récente d'un scanner Lidar Terrestre par l'ONF a entrainé la réalisation de nombreux scans et données de validation pour la recherche. De ce fait une question s'est posée comment organiser et référencer ces données ? Cette question soulève deux point, le stockage des données mais aussi l'interrogation de ses données.

De plus le scanner apporte de nombreuses informations qui pourront ainsi être stockées et auxquelles on pourra accéder plus facilement. Et toutes les données pourront être stockées et interrogées au même endroit.

[1] Dendrométrie : Science de la mesure des bois sur pied et des bois abattus.

5. Analyse des besoins

Certains besoins ont été émis quant au résultat de mon travail au sein du pôle. Le premier point est la réalisation d'une base de données géographique, qui permette de stocker les inventaires terrain réalisés, mais aussi les scans, ainsi que les résultats de traitement des scans suivant différents algorithmes utilisés.

Le second point est la réalisation d'une interface permettant la consultation des données, mais aussi leur modification, leur suppression, l'ajout soit manuellement, soit par un import automatisé, ainsi que l'export de données en texte ou csv. Il fallait aussi permettre la réalisation de requêtes afin de pouvoir analyser les données.

Le dernier point est la réalisation d'un plugin dans un logiciel SIG afin de pouvoir afficher les données géographiques de la base, avec un affichage pour les arbres selon l'essence et le diamètre. Mais aussi la possibilité de réaliser une carte automatique standardisée des données affichées.

C. Choix logiciel

1. Base de donnée

Nous avons choisi d'utiliser PostgreSQL 9.1 avec l'extension spatiale PostGIS 2.0, car c'est un logiciel gratuit open source et performant. C'est un logiciel stable car il date des années 1985, il possède une certaine ancienneté et dispose d'une communauté de développeurs qui ont permis cette stabilité et des évolutions régulières. De plus PostGIS contient de nombreuses fonctions de traitement géographique intégrées.

PostgreSQL fonctionne selon une architecture client/serveur, il est ainsi constitué :

- D'une partie serveur, c'est-à-dire une application fonctionnant sur la machine hébergeant la base de données (le serveur de base de données) capable de traiter les requêtes des clients. Il s'agit dans le cas de PostgreSQL d'un programme résident en mémoire.
- D'une partie client devant être installé sur toutes les machines nécessitant d'accéder au serveur de base de données (un client peut fonctionner sur le serveur lui-même).

Les clients peuvent interroger le serveur de base de données à l'aide de requêtes SQL.

2. Interface et plugin SIG

Au début du projet notre choix se portait sur wavemaker pour l'interface utilisateur, car simple d'utilisation, gratuit et vu en cours. Mais nous nous sommes rendu compte que ce logiciel permettait de faire uniquement des interfaces simples, mais si nous voulions réaliser des interfaces avec des actions spécifiques, telles que le calcul de coordonnées, wavemaker n'est pas optimisé. Il nous fallait pour cela programmer nous-même l'interface et choisir un langage à utiliser.

Pour le plugin SIG, nous voulions au début le réaliser sous arcgis (logiciel SIG dont ce sert l'ONF en gestion). Mais pour connecter une base de données PostgreSQL il faut posséder arcgis server qui est un logiciel très couteux. A cause de cela nous sommes reportés sur un logiciel SIG gratuit Qgis qui permet de lire et écrire dans une base de données PostgreSQL avec PostGIS nativement.

Pour le choix du langage de programmation, nous nous sommes rapidement décidés d'utiliser comme langage le C++ avec le Framework Qt. Ce langage est très bien documenté sur internet. De plus le logiciel COMPUTREE de l'ONF, développer en C++ avec Qt ce qui permettra si on le souhaite, de pouvoir réaliser des connexions dans le futur. Un autre point important est le fait que Qgis est conçu en C++ avec le Framework Qt, ce qui va permettre de faciliter la conception d'un plugin, mais aussi de réaliser tout le travail avec le même langage ce qui simplifie la réalisation.

a) Langage C++

Le C++ est un langage de programmation permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique. Le langage C++ est un langage amélioré du C et qui est orienté objet contrairement au C. C'est un langage multiplateforme, compilé (opposé aux langages interprétés), ce qui le rend très performant.

b) Framework Qt

Qt est une bibliothèque logicielle orientée objet développée en C++ par Qt Development Framework, filiale de Nokia.

Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les Unix (dont Linux), Windows et Mac OS X. Le fait d'être une bibliothèque logicielle multiplateforme attire un grand nombre de personnes qui ont donc l'occasion de diffuser leurs programmes sur les principaux OS existants.

A partir de Qt 4.5, QT est sous la licence LGPL v2.1. Cette licence permet des développements de logiciels propriétaires sans nécessiter l'achat d'une licence commerciale auprès de Qt Development Framework.

Parmi ses caractéristiques intéressantes, Qt met en place le principe des signaux et des slots. Les signaux et slots permettent d'interconnecter des objets Qt entre eux. Un signal est un évènement envoyé par un objet (exemple : clic de la souris sur un bouton), un slot est une fonction réalisant l'action associée à un signal. Tout objet Qt peut définir des signaux, et des slots pour recevoir des signaux en provenance d'autres objets Qt. C'est ce principe qui nous a permis de réaliser des classes générique, comme la classe pour l'affichage des données.

Qt offre également des composants d'interface graphique, d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, SQL, etc… Voici les classes principales que j'ai utilisées. Toutes les classes Qt présentent dans ce document seront souligné :

- QDialog : permet de faire afficher une fenêtre.
- QWidget : Permet de mettre des widgets dans un QDialog, les widgets sont les éléments constitutif d'une fenêtre (ex : bouton, case à cocher, combo box …).
- QFile : permet de créer un fichier, utilisé par exemple pour sauvegarder les requêtes.
- QTextStream : permet de lire ou d'écrire dans un fichier, utilisé aussi pour sauvegarder les requêtes.

- QSqlDatabase : permet de réaliser la connexion à une base de données.
- QSqlQuery : permet d'envoyer une requête sql à une base de données, utilisée pour réaliser les requêtes de mise à jour des coordonnées.
- QSqlQueryModel : permet de récupérer le résultat d'une requête (exemple d'utilisation dans la partie requête).
- QSqlTableModel : permet de sauvegarder les données d'une table afin de les afficher.
- QSqlRelationnalTableModel : possède les mêmes fonctions que la classe précédente mais permet de préciser si on veut afficher une autre colonne. Par exemple cela permet au lieu d'afficher un id de mettre un nom, en cas de table liée pour une clé étrangère.
- QMessageBox : permet de faire afficher un message à l'utilisateur.

II. Méthode de travail

A. Démarche de travail

J'ai choisi d'organiser mon travail suivant la méthode SADT qui est une méthode simple et que j'ai vu en cours.

SADT (en anglais Structured Analysis and Design Technique) est une méthode mise au point par la société Softech aux Etats Unis. Elle se répandit vers la fin des années 1980 comme l'un des standards de description graphique d'un système complexe par une analyse fonctionnelle descendante. La méthode SADT est une méthode d'analyse par niveaux successifs d'approche descriptive d'un ensemble quel qu'il soit. On peut appliquer le SADT à la gestion d'une entreprise tout comme à un système automatisé. C'est-à-dire que l'analyse chemine du général (dit "niveau A-0") vers le particulier et le détaillé. Le SADT est une démarche systémique de modélisation d'un système complexe ou d'un processus opératoire.

Le SADT a été réalisé une fois le projet bien défini. Ce schéma a permis de prévoir les étapes importantes du projet, ainsi de gagner du temps et de répartir les tâches et les étapes dans le temps. Le SADT est un moyen simple de décomposer les différentes fonctions d'un projet.

Se SADT a été réalisé avec DIA, un logiciel libre. Le SADT comptent 7 pages et a été décomposé jusqu'au niveau 3.

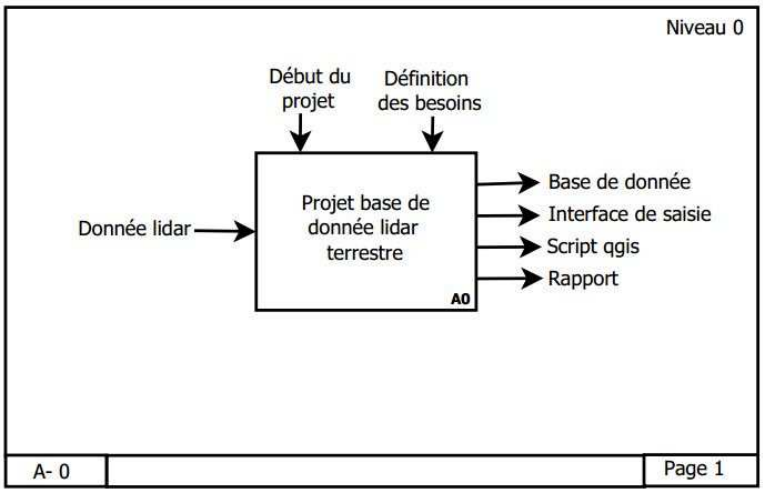


Figure 3 : SADT niveau 0

Le niveau 0 que l'on peut voir en figure 3, permet d'analyser le projet dans son ensemble et de voir ce que l'on a besoin en entrée ce que l'on aura créé en sortie et les conditions préalables à l'exécution du projet dans son ensemble.

Le niveau 1 que l'on peut voir en figure 4, est découpé en 3 entités qui reprennent un peu le diagramme de Gantt présenté dans la partie suivante :

- Réalisation de la base de données
- Interface de saisie
- Liens Qgis

Ensuite chacune de c'est trois entités a été décomposée dans le niveau 2 en sous entités. Le niveau 3 reprend deux entités dans la réalisation de la base de données.

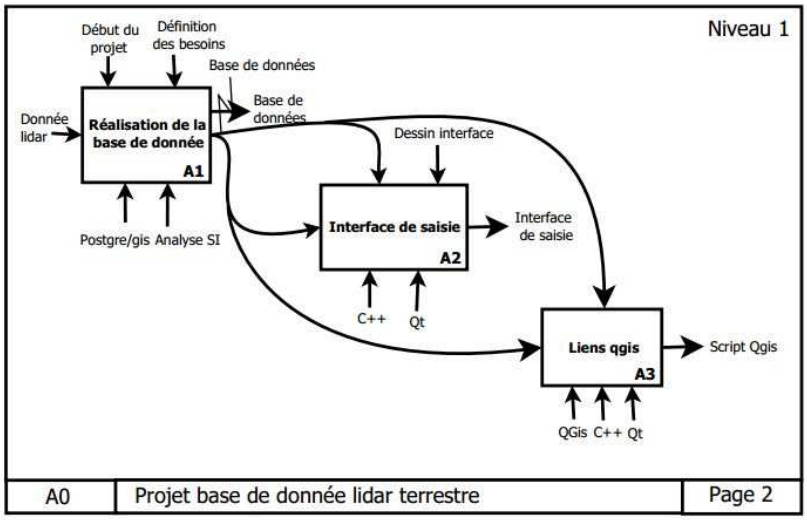


Figure 4 : SADT Niveau 1

Cette partie du SADT est pour moi la partie la plus intéressante. Car elle permet d'avoir une vue d'ensemble sur tout le projet. Et ainsi voir les entrées/sortie, les logiciels utilisés à chaque étape et les conditions préalables à chaque étape. Les niveaux suivants du SADT sont donnés en annexe.

B. Déroulement du travail

Le diagramme de GANTT est une méthode permettant de planifier son travail afin de pouvoir gérer son temps et ainsi gagner en productivité mais surtout de ne pas dépasser les délais imposés.

1. Planning prévisionnel

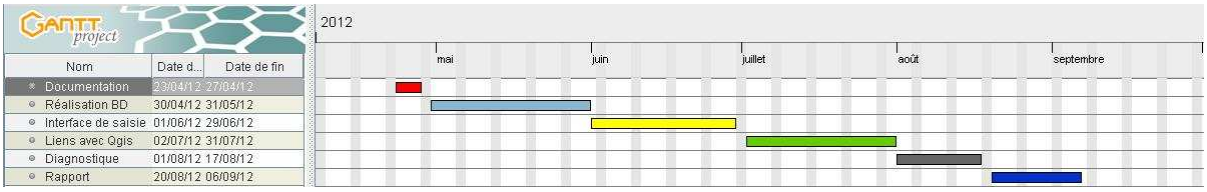


Figure 5 : Diagramme de Gantt prévisionnel

Le planning de la figure 5 m'a permi d'organiser mon travail sur toute la durée de mon stage, afin de me permettre de ne pas perdre de temps. Il y a 6 grandes périodes de travail :

- La partie 1 : la partie documentation m'a permi de m'informer sur le travail à réaliser, et de faire des choix techniques, par l'essai des fonctionnalités des logiciels et leur utilisation concrète, afin de comprendre leur fonctionnement et s'ils permettent de faire ce que l'on souhaite.
- La partie 2 : réalisation de la base de données.
- La partie 3 : interface de saisie ou interface utilisateur.
- La partie 4 : plugin Qgis.
- La partie 5 : diagnostique des inventaires réalisés et à réaliser

- La partie 6 : rédactions du rapport.

2. Planning final

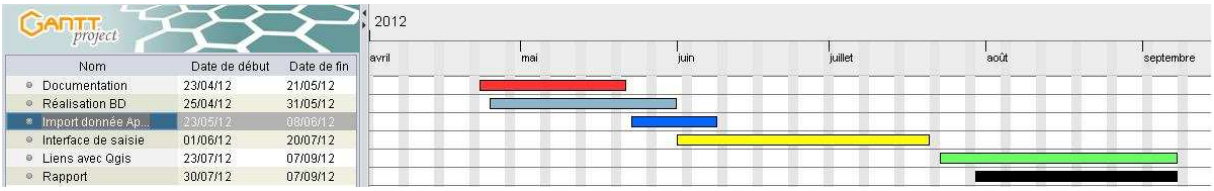


Figure 6 : Diagramme de Gantt final

Le planning final de la figure 6 permet d'observer le déroulement réel du stage. Dans le planning une partie a été enlevée, la partie diagnostique. Une nouvelle partie a été rajoutée c'est la partie sur l'import de données et l'apprentissage du C++.

La partie rajoutée consiste en la création d'une base de données test afin de voir la cohérence de la base de données lors de l'import de donnée. Mais aussi de permettre lors de la programmation des deux interfaces d'avoir une base qui fonctionne. J'ai souhaité avoir une base de donnée cohérente et qui fonctionne correctement lors de la programmation, afin qu'à la fin du projet aucun problème ne puisse apparaitre.

L'apprentissage du C++ a été une partie importante du stage, il a fallu faire un choix entre :

- Soit réaliser une interface avec peu de fonctionnalité avec wavemaker,
- Soit de prendre le parti d'investir du temps dans un nouveau langage de programmation et avoir plus de fonctionnalité au final. C'est ce dernier choix que nous avons fait.

La partie diagnostique, qui visait à utiliser la base terminé pour alimenter une thèse en cours a été supprimée. En effet, ce besoin a été reporté dans la thèse en question. Nous avons donc décidé de récupérer ce temps, afin d'enrichir l'interface utilisateur de fonction telle que le passage en mode formulaire, ou l'export de données.

3. Analyse des plannings

De plus de nombreuses recherches sur la faisabilité, ont dû être faites pour déterminer quel logiciel/langage correspondait le mieux à nos besoins.

La partie documentation a été plus longue que prévue, car le choix de réaliser l'interface par programmation ma obligé à me renseigner bien plus sur les interactions entre C++/Qt et le logiciel de base de données PostgreSQL, cela est dû aux choix fait dans le chapitre I C.

La durée pour la création de la base de données a été respectée, mais il y a eu l'ajout que je n'avais pas prévu du test d'import de donnée et l'apprentissage du C++ que j'avais inclus dans la création de l'interface utilisateur. Celle-ci a été bien plus longue que prévu et a entamé sur la partie interface SIG, la suppression de la partie diagnostique nous a permi de compenser ce temps supplémentaire.

III. Travail effectué

La première grande étape avant la réalisation de la base de données a été de comprendre le projet et de m'imprégner du travail à réaliser. Pour cela mon maitre de stage Alexandre Piboule m'a expliquer son travail, comment il fonctionnait et ce qu'il attendait de mon stage. On peut retrouver les éléments métier issus de cette étape dans la partie B.2, « démarche d'inventaire ».

A. La base de données

1. Modèle Conceptuel de Données (MCD)

Lors de la réalisation du mcd sous Analyse SI, il y a eu 3 grandes étapes successives. Car à la fin de chaque réalisation, je présentais le résultat de mon mcd à mon maitre de stage, afin de savoir s'il convenait. Ensuite nous discutions de la réalisation et nous convenions des modifications à réaliser.

Nous avons choisis d'utiliser pour réaliser la base de données la méthode Merise. C'est une méthode d'analyse, de conception et de gestion de projet informatique. Dans notre cas nous l'avons utilisé pour réaliser la base de données.

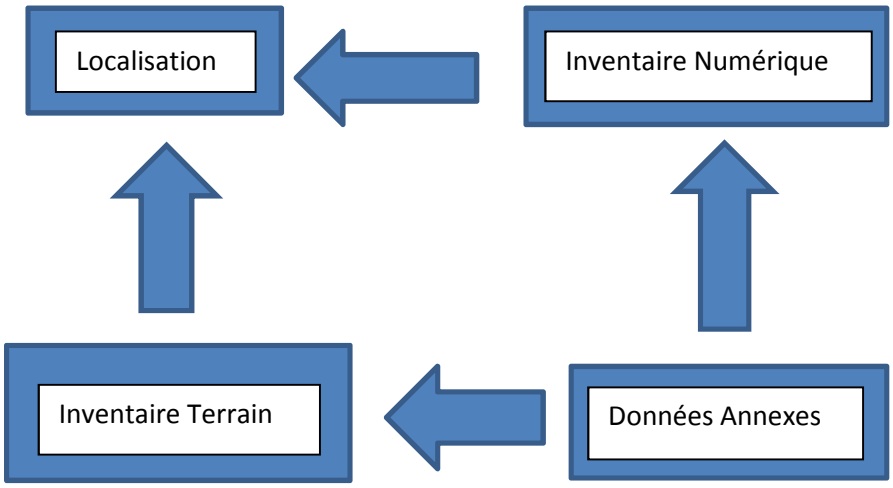


Figure 7 : Schéma du MCD

Chaque version du mcd possède quatre grandes parties, comme on peut le voir dans la figure 7. Une partie inventaire terrain qui regroupe les données terrain, une partie inventaire numérique qui regroupe les scans la localisation des fichiers et les traitements réalisés, une autre partie pour les tables annexes qui n'appartiennent pas à un seul groupe et une dernière partie pour la localisation des placettes.

a) Etape 1

La partie inventaire numérique, est encore assez simple. La partie principale de l'inventaire numérique est la table scan, l'association fusionner permet à un scan d'être associé à d'autres scans. Ainsi un scan peut être la fusion de plusieurs, car certaines placettes peuvent être inventoriées en multi-scan. Il y a aussi le scanner, la société et les filtres qui ont pu être utilisés, il y a 2 types de filtres : matériel et logiciel.

Pour la partie localisation l'élément central est la placette d'inventaire qui sera géolocalisée. La placette appartient à un site et un site à un réseau éventuel. Mais elle appartient aussi à une parcelle forestière, à une forêt, une agence et une direction territoriale, c'est le système de classification que l'ONF utilise afin de découper le territoire.

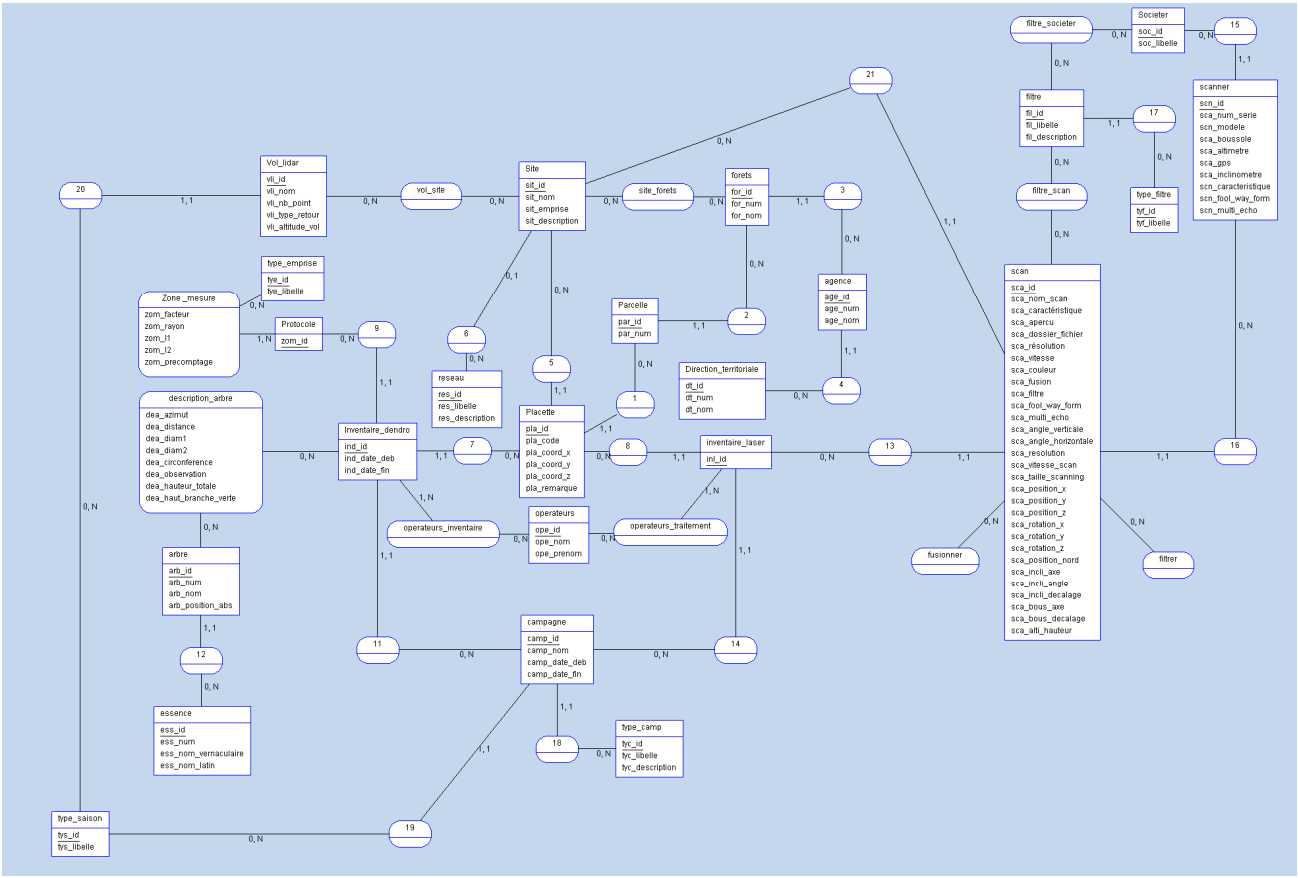


Figure 8 : MCD 1

Pour l'inventaire terrain, il y a deux parties : une partie protocole, qui va recueillir les informations sur les zones de mesure avec les types d'emprise et le diamètre de précomptage. Les types d'emprise peuvent être circulaire rectangulaire ou relascopique[2]. Une partie inventaire proprement dit avec chaque arbre unique géolocalisé et une association qui recueille les données pour chaque inventaire successif ce qui permet de voir l'évolution de l'arbre.

On peut voir dans la figure 8 la présence d'une table pour les essences des arbres. Une partie vol lidar a aussi été créée, afin de recenser les différents vols lidar ayant pu être réalisés sur l'emprise des placettes. Mais cette partie sera abandonnée plus tard. La dernière partie est l'organisation des opérateurs et des campagnes qui sont reliés à un inventaire dendrométrique et un inventaire laser. Une campagne d'inventaire va être un ensemble d'inventaires soit dendrométriques, soit lasers sur une période plus ou moins longue.

[2] Placette relascopique : cette technique permet de limiter le relevés aux arbres sélectionnés à l'aide d'une jauge d'angle projetant un angle constant : chaque classe de diamètre est inventoriée sur une superficie proportionnelle à sa surface terrière (surface à l'hectare de tous les arbres à 1.30 mètres).

b) Etape 2

Après réflexion et échange avec mon maitre de stage nous sommes convenus de certaines modifications à réaliser. Afin de mieux représenter la structure des données. La partie vol lidar à disparue pour ne se concentrer que sur la partie lidar terrestre. La partie localisation a été réduite, car une base de données géographique existe déjà à l'ONF pour la gestion des parcelles, forets, etc… Il a en effet été jugé préférable de ne pas dupliquer les données. Une nouvelle table localisation apparait par contre, qui permettra de faire une liaison avec la base de l'ONF.

On peut voir dans la figure 9, le découpage des campagnes en deux parties distinctes. Une partie campagne laser et une partie campagne terrain, mais qui ont la même fonction que précédemment.

La partie inventaire numérique a été modifiée, en rajoutant des champs dans la table scan et en créant une table fichier. Cette table permet de sauvegarder les types de fichier pour un scan car ils peuvent être sous différentes extension xyb, fls, etc… De plus les coordonnées de chaque arbre seront extraites par traitement automatique et ajouter dans la table arbre numérique. Les simulations et les algorithmes utilisés seront stockés, car selon l'algorithme la détection des arbres peut changer, et donc pour deux simulation le nombre et l'emplacement des arbres peut variée, c'est pour cela qu'un arbre appartient à une simulation.

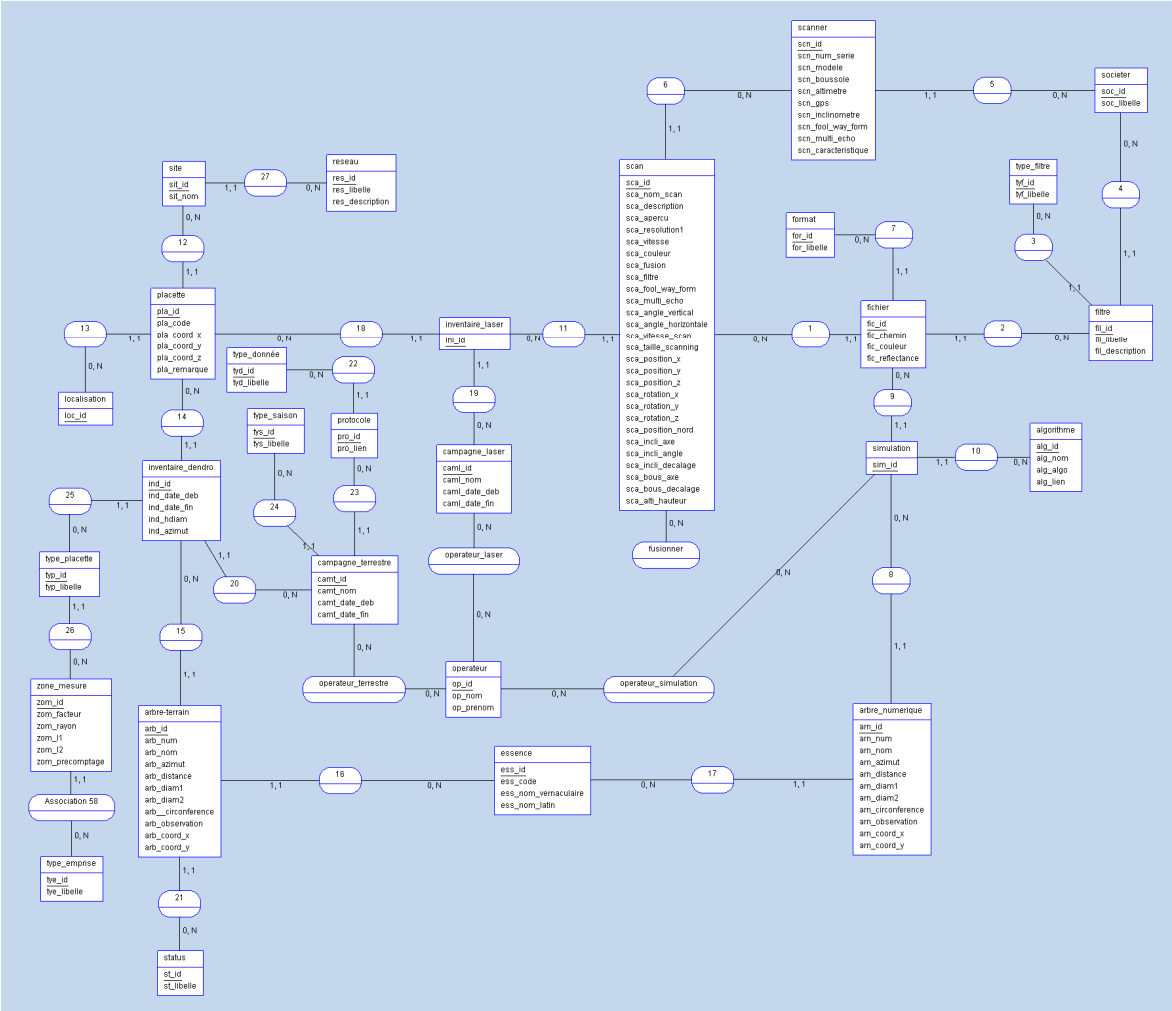


Figure 9 : MCD 2

Pour la partie inventaire terrain, les modifications réalisées : sur la partie protocole simplification de la gestion par la sauvegarde d'un lien vers le fichier de protocole, pour la table arbre terrain, chaque arbre ne sera plus unique mais rajouté à chaque nouvel inventaire et pour table statuts, qui permettra que chaque arbre est un statut particulier.

c) Etape 3

Après la création du précédent MCD de la base de données correspondante, nous avons réalisé quelques modifications supplémentaires. Car lors des tests d'import de données et la création de l'IHM nous avons constatés que quelques détails avaient été oubliés. Pour cela un dernier mcd a été créé.

Lors de l'ajout de données dans la base, les modifications ont été l'ajout de champs, la modification des type de champ ou décider quel unité de mesure serait le mieux adaptés dans tel ou tel cas.

La modification importante de la base a été réalisée lors de la création de l'IHM, car la mise à jour des coordonnées des points nous a révélé un problème. Les points qui ont été relevée au GPS sont les placettes qui ont donc des coordonnées absolues. Les autres points, les arbres et les scans ont des coordonnées relatives au centre de la placette. Les arbres terrain ont été relevés avec leur azimut distance et les scans par des coordonnées x,y et les coordonnées des arbres numériques sont relatives au centre du scan en x,y.

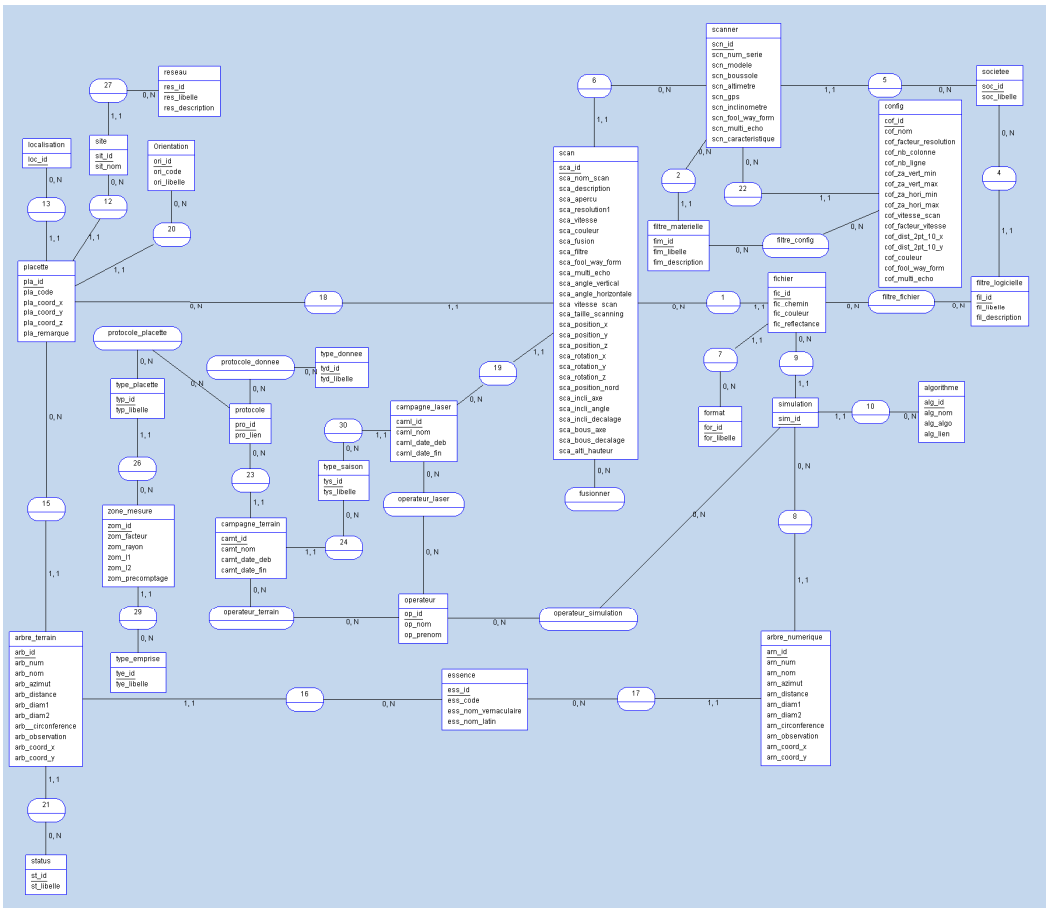


Figure 10 : MCD 3

Pour optimiser la rapidité des requêtes SQL de calcul de coordonnées nous avons enlevé des tables intermédiaires qui au final n'avait que peu d'intérêt. Les tables qui ont été enlevées sont inventaire laser et inventaire terrain qui ne servait que de liens entre trois tables et réalisé ainsi des nouveaux liens plus direct.

2. Création de la base de données

Une fois le MCD terminé il existe dans Analyse SI une fonction qui permet de concevoir un modèle logique de donnée (MLD) et ensuite à partir de ce MLD de créer un script avec pour extension .SQL.

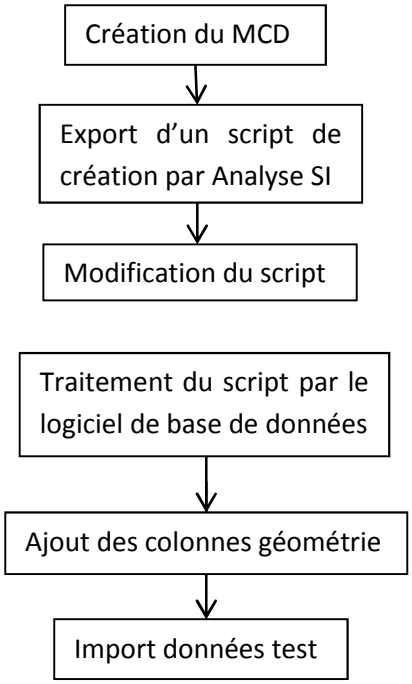


Figure 11 : Etape de création de la base de données

Avant la création de la base certaines modifications ont dû être faites, car le script généré est en sql standard et il a fallu s'adapter aux caractéristiques spécifiques de PostgreSQL. De plus il y a eu beaucoup de problèmes avec les types de données : le type auto-incréments qu'il a fallu remplacer par le type Serial, utiliser uniquement comme type du texte, des booléens, dates, doubles et des Integer car les autres posaient des problèmes lors de l'import.

Pour chaque MCD réalisé nous avons fait une base de données pour voir si tous se passait bien à l'import. De plus les champs géographiques ne sont pas réalisables directement, avec le logiciel qui permet de réaliser les MCD analyse SI. Il faut pour cela les ajouter après l'import de données avec PostgreSQL.

Puis ensuite nous avons effectué des vérifications avec DbVisualizer qui permet de créer un Modèle Physique de Données (MPD que l'on peut voir en figure 12), avec lequel nous allons pouvoir observer la conformité de la base.

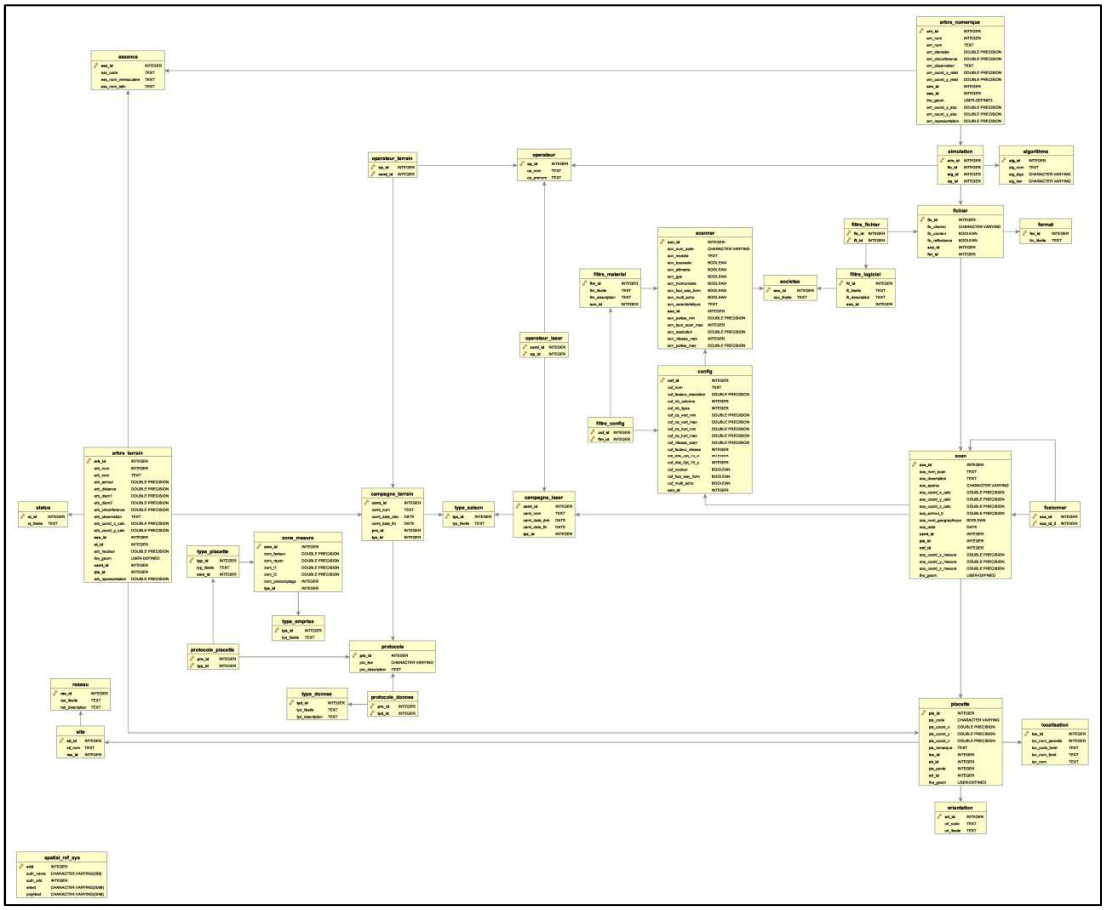


Figure 12 : MPD final de la base de données

Après la réalisation de la base de données un test d'import de données à été fait, afin de vérifier la conformité de la base de données.

3. Description de la base de données

Pour la description de la base de données j'ai réalisé un dictionnaire de données qui permet pour chaque table de connaitre les champs ainsi que le type des clefs et une description pour chaque champs. Le dictionnaire est donné en annexe n°2.

Il y a une ligne pour chaque champ et 5 colonnes :

- Champs : le nom du champ.
- Type : le type de la donnée.
- Caractéristique : clef primaire, étrangère, les deux ou aucune.

- Définition : explication sur l'utilisation du champ.
- Remarque : Si un détail particulier est à signaler.

Champs	Type	Caractéristique	Définition	Remarque
sca_id	Integer	Clef Primaire	Id	
sca_description	Text		Description du scan	
sca_apercu	Varchar		lien vers le fichier d'apercu	
sca_coord_x_calc	Double Précision		Coordonnee absolue calculer par rapport au centre de la placette en Lambert 93	
sca_coord_y_calc	Double Précision		Coordonnee absolue calculer par rapport au centre de la placette en Lambert 93	
sca_coord_z_calc	Double Précision		Coordonnee absolue calculer par rapport au centre de la placette en Lambert 93	
sca_azimut_0	Double Précision		Azimut au zero du scan	En grade
sca_nord_geographique	Boolean		Si azimut = 0 alors le scan est au nord geographique	
sca_date	Date		Date de prise du scan	
caml_id	Integer	Clef Etrangère	Campagne lidar du scan	
pla_id	Integer	Clef Etrangère	Placette ou ce situe le scan	
cof_id	Integer	Clef Etrangère	Configuration du scan	
sca_coord_x_mesure	Double Précision		Coordonnée relative du scan au centre de la placette	
sca_coord_y_mesure	Double Précision		Coordonnée relative du scan au centre de la placette	
sca_coord_z_mesure	Double Précision		Coordonnée relative du scan au centre de la placette	
the_geom	Géométrie		Colonne géographique pour affichage SIG	

Figure 13 : Extrait du dictionnaire de données : table scan.

Le dictionnaire de données dont on peut voir un extrait à la figure 13, permet de décrire la base de données afin de connaitre l'utilisation de chaque champ. Cela permet à un utilisateur qui utilise la base pour la première fois de ne pas se tromper.

B. Interface Utilisateur

L'environnement de développement utilisé est Qt Creator qui regroupe toute les fonctions pour programmer en C++ avec Qt et qui possèdent aussi Qt Designer qui permet de réaliser des interfaces en faisant des drags et drops.

Nous avons choisi de réaliser le programme suivant le principe du Modèle Vue Controller ou MVC qui permet d'organiser la programmation autour d'un controller. Cela permet de réaliser des classes génériques qui pourront être utilisées par différentes classes. Les classes génériques permettent d'avoir un programme structuré réutilisable par d'autres. Et surtout la gestion des données est séparée de l'interface graphique, ce qui permet de faire évoluer chaque partie indépendamment.

J'ai choisi de vous présenter chaque partie, de façon à montrer ce que j'ai réalisé, mais sans le code afin que le rapport ne soit pas trop fastidieux, mais le code est disponible en annexe n°4.

1. Connexion à la base de données

A l'ouverture de l'application une première fenêtre de connexion que l'on peut voir à la figure 14, s'ouvre en même temps que la fenêtre d'accueil. C'est une sécurité pour que tout le monde ne modifie pas la base, mais aussi un moyen de modifier les paramètres de connexion.

Cette fenêtre permet de recueillir les cinq paramètres de connexion indispensables. Ceux-ci vont servir lors de l'appel de la classe qui permet de réaliser la connexion. De plus le mot de passe et l'utilisateur peuvent être par défaut, ces paramètres permettent seulement de consulter la base mais pas d'ajouter de données ou de la modifier. Le nom, le port et hôte sont par défaut aussi mais peuvent être modifiés, au cas où l'on souhaite déplacer la base.

Cette classe, lors de la validation, va appeler une autre classe qui elle va permettre la connexion. Pour réaliser cette connexion il faut avoir les drivers de la base à laquelle on souhaite se connecter. Un problème sous Windows c'est posé pour avoir les drivers, car ils ne sont pas fournis avec Qt, contrairement à Linux. Il a fallu les télécharger à part pour que cela fonctionne.

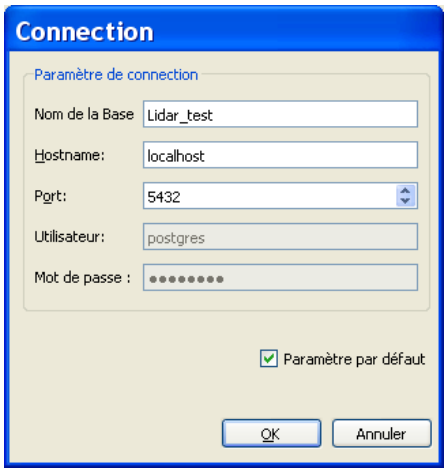


Figure 14 : Fenêtre de connexion

Pour la connexion, la classe databaseConnection que j'ai créé utilise une fonction QSqlDatabase fonction de Qt pour la connexion à la base de données.

2. La fenêtre principale

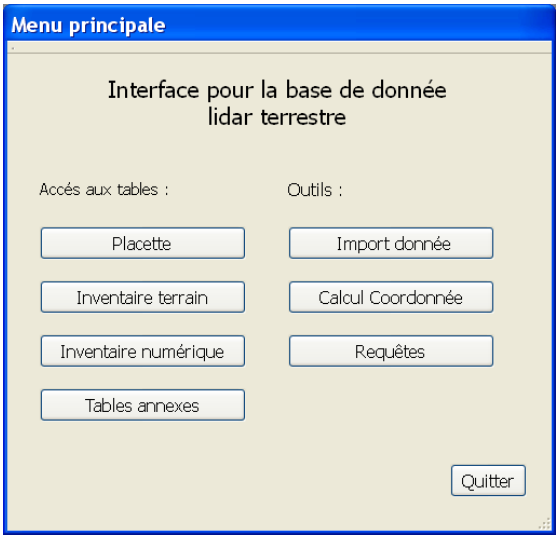


Figure 15 : Fenêtre principale

La fenêtre principale que l'on peut voir à la figure 15, permet d'accéder aux différentes fonctions de l'application. Il y a deux parties bien définies l'accès aux tables et les Outils.

- Accès aux tables : La base est composée de 4 grandes parties : la localisation (la placette), la partie inventaire terrain et la partie inventaire numérique. La partie tables annexes regroupe les tables, qui contiennent les codes de référence et qui interagissent avec toutes les parties.

- Outils : Trois outils ont été créés :
  - import de données,
  - calcul de coordonnées pour les arbres, les placettes,
  - la réalisation de requêtes.

3. Accès aux tables

a) Ecrans d'accès aux tables

Il y a, comme vu précédemment, quatre parties qui représentent les quatre grandes parties de la base de données créée. Pour chaque partie, un écran donne accès aux différentes tables.

Ces différents écrans ne sont composés que de boutons comme on peut le voir sur la figure 16 qui permettent d'accéder à la table choisie, en appelant une seule classe générique. Ainsi tous les boutons « tables » appellent en fait une unique classe, avec un paramétrage différent. Cette classe générique permet l'affichage et la modification de n'importe quelle table de la base. De plus les boutons ont été disposés de façon à représenter le modèle de données, afin que l'utilisateur puisse se repérer dans l'utilisation et qu'il sache sans regarder un autre document l'organisation des données.

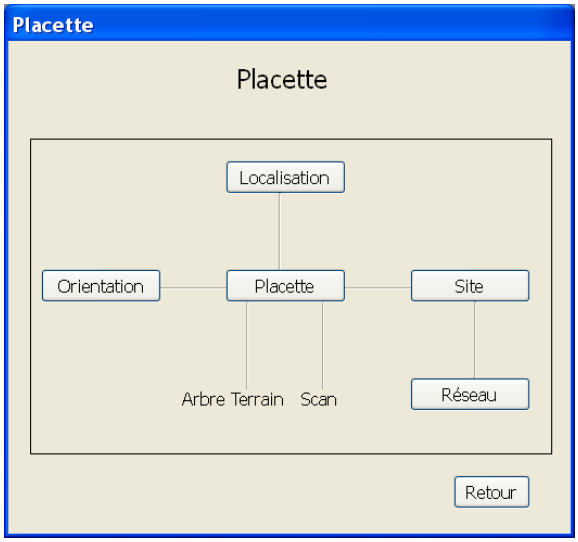


Figure 16 : Accès table placette

Voici en exemple la partie localisation avec en table centrale la placette, avec autour les tables annexes qui appartiennent à la localisation de la placette. Seule la table placette est géographique. On peut voir des tables qui ne sont pas des boutons (dans cet exemple, Arbre Terrain et Scan), mais qui permettent de voir les liens vers les autres parties.

Lors du clic sur une table on va utiliser via le controller un QSqlRelationnalTableModel qui va prendre en paramètre le nom de la table à afficher. Ce model sera utilisé en paramètre pour l'affichage de la table.

Vous pouvez voir en annexe n°3 les autres parties (inventaire terrain, inventaire numérique et autre tables).

b) *Affichage Tables et requêtes*

Dans cette partie je vais traiter de la classe qui permet l'affichage des données mais aussi des requêtes (cf partie 4.c). Cette classe est un peu particulière car elle est générique c'est-à-dire que quel que soit le type de table ou de requête elle va permettre de l'afficher, mais aussi quel que soit la base de données en entrée.

Cette classe est composée de deux parties, une partie visualisation tableau (ou toute les lignes de la table sont affichées voir figure 17) et une partie visualisation formulaire (une seule ligne est affichée voir figure 18). Cela permet de passer d'un affichage à l'autre facilement. De plus si une ligne est sélectionnée dans le tableau elle sera visible dans le formulaire et la ligne où on est dans la partie formulaire sera sélectionnée en passant en affichage tableau. Dans les deux cas, chaque case s'adapte au type de champs :

- Saisie manuel pour les champs numérique,
- Liste de choix en cas de clef étrangère.

Les autres fonctions sont : la suppression de ligne unique en mode formulaire et multiple en mode tableau, la modification de ligne et l'ajout de ligne. Pour activer ces fonctions il faut passer en mode modification, ce qui garantit de ne pas réaliser de modification par erreur. Cela nécessite de s'être connecté avec des droits de modification.

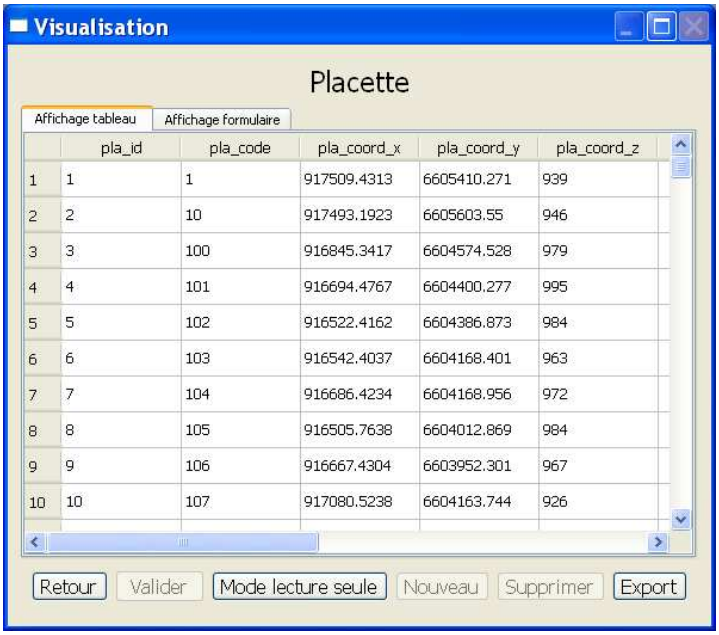


Figure 17 : Visualisation tableau

Pour l'affichage en mode tableau on fait passer un QSqlRelationnalTableModel dans un QTableWidget. Il y a trois types de model en Qt : le QSqlQueryModel qui permet de faire afficher le résultat d'une requête, le QSqlTableModel qui permet d'afficher une table de la base et le QSqlRelationnalTableModel qui permet de faire afficher le nom d'un champ au lieu de l'ID de la même table. Le bouton valider permet de valider la création d'une nouvelle ligne, d'abord on fait nouveau, puis il faut remplir les champs qui ne doivent pas être nul, et enfin valider.

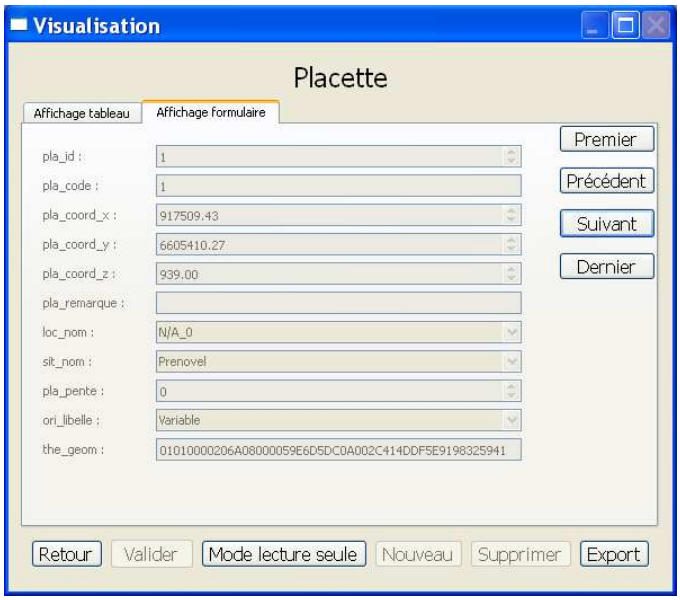


Figure 18 : Visualisation formulaire

Pour l'affichage en mode formulaire, on part du modèle créé précédemment et on utilise un QDataWidgetMapper, qui permet d'organiser les données en formulaire. Sur le côté on peut voir des boutons qui permettent de naviguer dans le widget mapper. On a début, fin, avancer et reculer. Les boutons du dessous se comportent de la même façon que pour le mode tableau mais leur programmation en est différente. On peut voir sur les deux types de visualisation, un bouton pour l'export.

Le bouton export permet d'exporter dans un fichier selon trois types de fichier :

- CSV avec point-virgule,
- Csv avec tabulation,
- Txt avec tabulation.

Pour cela on parcours le model utilisé lors de l'affichage. Entre chaque valeur on met une tabulation ou un point-virgule selon le choix, à la fin de chaque ligne du model on saute une ligne dans le fichier, grâce à deux boucles itératives une pour les colonnes et une pour les lignes.

4. Outils

a) Import de données

La partie import de données que l'on peut voir en Figure 19, est une des grandes parties outils du menu principal. Elle permet de faire un import de données pour chaque table. On peut utiliser les mêmes types que pour l'export de donnée (Csv point-virgule ou tabulation et Txt tabulation). Chaque fichier doit avoir le titre des colonnes qui peut être mis dans n'importe quel ordre dans le fichier. En effet le logiciel utilise les titres des colonnes pour identifier le champ.

Pour réaliser l'import c'est une fonction copy en sql qui est utilisée. Pour cela on utilise un QSqlQuery qui permet d'envoyer une requête à la base de données. Si une erreur se produit on aura en retour le message d'erreur de la base de données.

Figure 19 : Import de données

b) *Calcul des coordonnées*

Cette partie permet de calculer automatiquement et de remplir les champs the_geom (champs PostgreSQL qui permettent de stocker des informations géographiques). Le calcul se fait à partir du centre de la placette dont on connait la position précise et suivant le cas des coordonnées x,y ou azimut/distance pour les arbres.

Cette partie devait être à l'origine un import de fichier Shape dans la base de données mais il fallait pour cela créer des points dans le logiciel SIG et réaliser quelques traitements. Pour simplifier cela et que tout le monde soit capable de l'utiliser nous avons préféré réaliser le calcul grâce à l'interface. De plus non seulement on simplifie le processus mais on a un gain de temps par comparaison au traitement sous logiciel sig.

L'interface permet de choisir quelle table on souhaite mettre à jour comme on peut le voir sur la figure 20. Le bouton calculer permet suivant le cas d'envoyer une à trois requêtes de la même façon que pour l'import de donnée grâce à un QSqlQuery. Une requête par table.

Figure 20 : Calcul de coordonnée

Le problème que nous avons eu à ce moment est pour l'optimisation des requêtes, nous avons dû supprimer des tables intermédiaires, ce qui a permis dans la requête de réaliser moins de jointures et d'augmenter la rapidité du résultat. C'est principalement pour cette raison que le dernier

mcd a été créé. Nous avons modifié les tables mais aussi les requêtes en elle-même, car au début les requêtes avaient tendance à être vraiment longue (plusieurs minutes), mais cela était dû à une écriture non optimisée de la requête.

c) Requête

Voici la classe qui permet de créer des requêtes on peut la voir en figure 21, mais aussi de les enregistrer dans un format sql avec le nom que l'on souhaite, de choisir le dossier où elles sont enregistrées et aussi de charger les requêtes déjà crées.

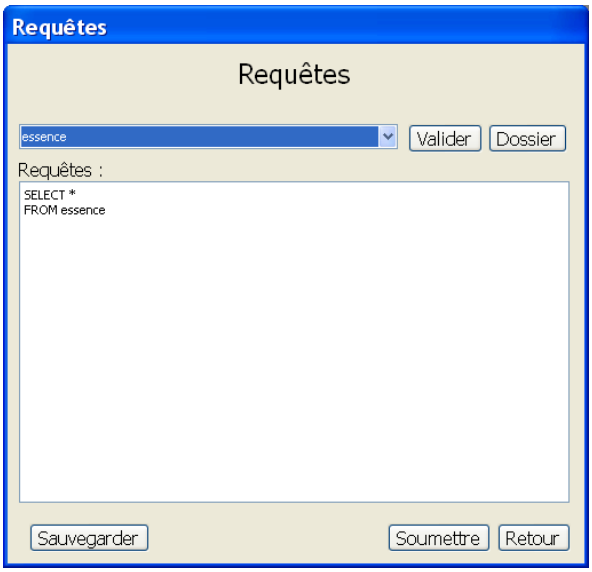


Figure 21 : Requêtes

Lors de la sauvegarde, le code SQL de la requête est enregistré dans une variable QString (type chaine de caractère de Qt). Après avoir renseigné son nom par l'intermédiaire d'une pop-up, le fichier va être enregistré dans le dossier sélectionné précédemment. La fonction va créer un fichier avec comme extension .sql, pour cela on utilise la classe QFile qui va prendre le chemin du dossier, ajouter le nom donné et l'extension.

Ensuite un flux de données QTextStream va permettre d'écrire dans le fichier le code de la requête écrite. Puis la comboBox va se mettre à jour avec le nom des fichiers déjà crées et le nouveau fichier. Pour cela le logiciel va lire le nom des fichiers dans le répertoire grâce à la classe QFileInfo et les ajouter à la comboBox avec une boucle itérative. Seuls les fichiers avec une extension .sql sont pris en compte.

Pour soumettre on va utiliser un QSqlQueryModel qui va enregistrer le résultat de la requête et qui va être mis en paramètre de la classe utilisée pour afficher la requête. (cf partie B.3.a)

C. Interface avec SIG

L'interface SIG regroupe les outils qui vont permettre de faciliter l'affichage des couches et leurs mises en page, uniquement sur la base de données. Elle va aussi permettre de créer l'emprise des placettes, mais aussi d'ajuster l'affichage de la circonférence des arbres en fonction de la précision recherchée.

Cette partie est réalisée en C++ avec le Framework Qt mais aussi avec l'api Qgis qui permet d'utiliser les fonctions présentes dans le logiciel SIG Qgis. Notre interface sera un plugin, qui se situera dans le menu des plugins de Qgis.

La plupart de la documentation que l'on trouve est faite pour les développeurs en python. J'ai donc du retransformer le code python afin qu'il s'adapte aux C++. Nous avons pu réaliser cela car ce sont les mêmes classes qui sont utilisé en python et en C++.

1. Connexion du plugin

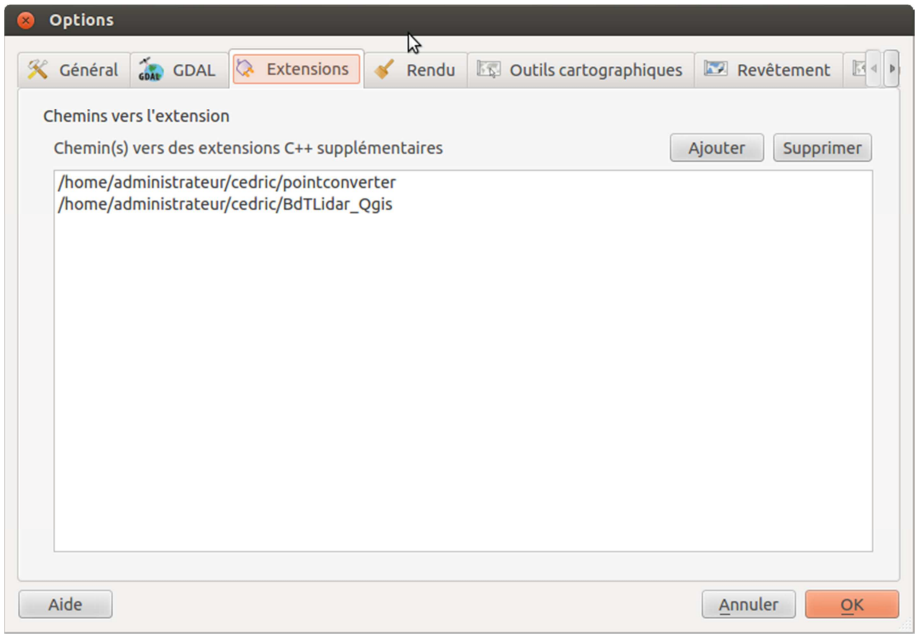


Figure 22 : ajout du plugin

La première étape est de connecter le plugin en C++. Pour les plugins en python le chargement est automatique, mais dans notre cas il faut indiquer a Qgis où se situe le plugin. Pour cela il faut aller dans les options de Qgis qui est situé dans préférence et donner le chemin où est situé le plugin comme on peut le voir à la figure 22.

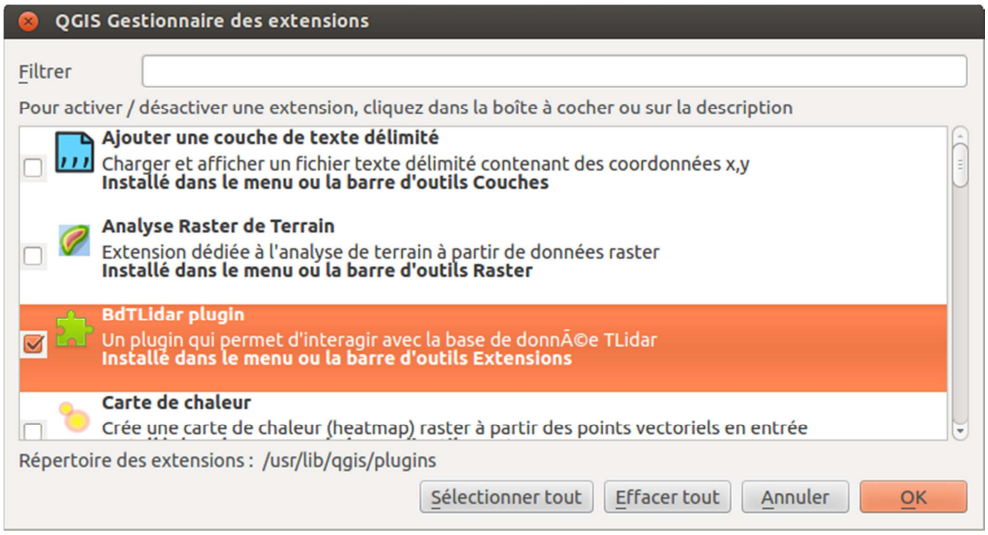


Figure 23 : connexion au plugin

La seconde étape est de charger le plugin qui a pour nom BdTLidar plugin comme on peut le voir à la figure 23, pour cela il suffit d'aller dans le menu extensions et le sous menu gestionnaire d'extension. Une fois cela fait un nouveau sous menu apparaitra dans le menu extensions.

2. Organisation du plugin

Le plugin BdTLidar à cinq menus qui permettent de réaliser les différentes fonctions du plugin. C'est fonctions sont :

- Connexion : permet de modifier les paramètres de connexion.
- Affichage : pour permettre l'affichage des données.
- Mise en page : afin de pouvoir réaliser une mise en page automatique.
- Emprise : permet de créer les emprises des placettes.
- Affichage : permet de régler l'affichage de la taille des arbres.

a) Connexion à la base de données

La connexion à la base de données suit le même processus que pour l'interface utilisateur, étant dans le même langage. Au début nous pensions possible d'utiliser les fonctions de Qgis, afin de réaliser la connexion et ainsi utiliser le travail déjà réaliser par les développeurs de Qgis sur la sauvegarde des connexions. Mais j'ai préféré réutiliser la classe créée dans le travail précédent.

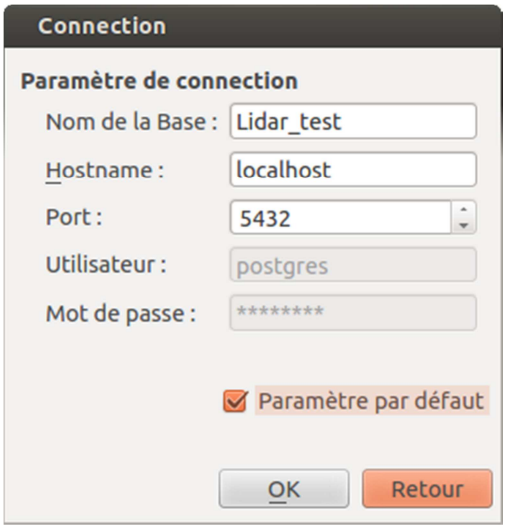


Figure 24 : connexion à la base de données

On peut voir à la figure 24, les mêmes outils à savoir la sauvegarde des paramètres et leur modification, ainsi que l'utilisation d'un paramètre par défaut. La fenêtre s'ouvre sur le premier onglet du plugin où l'on clique. Et si l'on souhaite réaliser une nouvelle connexion, il suffit de cliquer sur l'onglet connexion et la même fenêtre apparait de nouveau pour réaliser la connexion.

b) Affichage des données

Cette partie que l'on peut voir à la figure 25, est la plus importante du plugin car elle permet l'affichage des données contenues dans la base de données. On peut décider de choisir différentes

échelle d'affichage, toutes les forêts, une forêt précise, une parcelle dans la forêt mais aussi une placette particulière de la parcelle.

Chaque combo box se met à jour via la sélection d'une ligne dans la combo box précédente par l'envoi d'une requête sql à la base de données. Ensuite on peut choisir d'afficher les différentes tables qui contiennent des informations géographiques à savoir la table scan, placette et les deux tables arbres.

Figure 25 : Onglet affichage

L'affichage se réalise avec une mise en page. Particulière pour cela nous chargeons en même temps que les données un fichier de style Qgis, qui a comme extension qml. Ce fichier de style est créé en sauvegardant un style qui a été déjà créé dans un projet. Chaque couche aura donc un style différent, les placettes seront représentées par des étoiles rouges, les scans par des triangles jaunes et les arbres par des cercles auront une couleur pour l'essence, une taille en fonction du diamètre et le contour en rouge pour les arbres numérique et noir pour les arbres terrain.

c) Cartographie automatique

Cette partie permet de réaliser automatiquement une carte à partir des éléments choisis dans la partie affichage, on peut voir un exemple à la figure 26. Pour cela il suffit de cliquer sur l'onglet mise en page dans le plugin. Une fenêtre s'ouvre et demande le titre que l'on souhaite mettre sur la carte et la carte apparait.

Pour réaliser cette partie on utilise une classe Qgscomposition qui permet d'utiliser la fonction de Qgis composeur d'impression. Puis on ajoute les différentes classes à l'intérieur, c'est-à-dire :

- La carte à afficher,
- Un label pour le titre,
- Une flèche pour indiquer la direction du nord,
- Une légende,
- Une échelle.

Ce sont les éléments indispensables lors de la réalisation d'une carte. Il a fallu pour chaque élément les situer dans l'espace afin qu'ils ne se superposent pas.

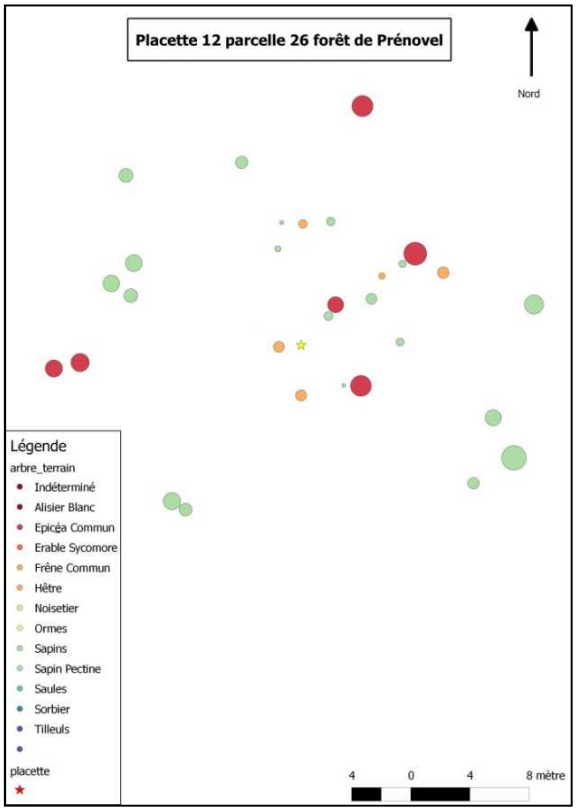


Figure 26 : Carte automatique

L'exemple montre la carte pour une parcelle avec affichage du centre de la placette avec une étoile rouge et arbre en fonction de l'essence et du diamètre, car la carte reprend l'affichage des données avec le style qui leur a été appliqué.

d) Création emprise des placettes

Pour réaliser l'emprise des placettes, il a fallu reprendre les différents protocoles réalisés sur la placette via une requête sql et récupérer le type de placette. En fonction du type de placette mais aussi des valeurs d'emprise on va réaliser différentes emprises.

Pour réaliser les emprises on va créer un buffer (zone tampon) sur le centre de la placette lorsque cette placette est circulaire, pour les placettes rectangulaires on va réalisées les emprises avec la fonction buffer carré et pour les placettes relascopique pas de moyen de représentation.

e) Calcul de coefficient d'affichage

Cette étape permet de changer l'affichage de la taille des cercles. En effet lorsque que l'on affiche une seul placette ou plusieurs, la taille des cercles doit être ajustées afin que certains arbre soi masquer par d'autre. Pour cela on va appliquer un coefficient pour l'affichage sur la taille du cercle.

Pour le calcul, la première étape est de vérifier si une valeur est entrée. La seconde étape est de vérifier qu'une circonférence a été rentrée, si il n'y a pas de circonférence il faut la calculer avec le diamètre, voir même faire une moyenne des deux diamètres qui ont été récolté. Ensuite à partir de cela on passe la circonférence en mètre car le logiciel ne comprend les données qu'en mètre et les données sont à l'origine en centimètre.

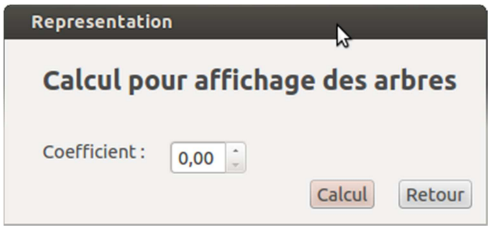


Figure 27 : Onglet représentation

L'étape suivante est d'ajouter le coefficient rentré par l'opérateur dans l'interface que l'on peut voir à la figure 27, et de le multiplier à la circonférence en mètre. Puis si il y a des données affichées de les mettre à jour afin que l'affichage soit actualiser.

IV. Perspectives

Lors du stage certains points que nous voulions réaliser non pas pu l'être à cause du temps ou à cause de moyen mais en voici certains que nous aurions pu explorer.

A. Base de données

Pour la base de données, trois points importants auraient pu être développés pour la mise en place de la base :

Le premier point que nous avons évoqué était d'étendre la base aux données Lidar aérien afin de pouvoir faire des croisements entre sources de données. Nous avons privilégié la partie lidar terrestre, car pour pouvoir faire une partie lidar aérien, nous aurions dû prendre du temps pour regarder les données et l'objectif premier était de réaliser une base de données pour le lidar terrestre. Mais cela n'empêche pas dans le futur d'exploiter cette possibilité.

Le second point aurait été de mettre la base sur un serveur afin de pouvoir y accéder plus facilement, ainsi que les données auquel se réfèrent la base, afin de pouvoir réaliser des sauvegardes des données.

Le troisième point serait de stocker les scans Lidar terrestre directement dans la base de données afin de pouvoir les stocker et les afficher mais surtout de pouvoir avoir une sauvegarde des données.

B. Interface utilisateur

Pour la partie interface utilisateur, il aurait été intéressant pour les requêtes, de développer un module d'assistant requête qui permettent à une personne ne connaissant pas le sql de pouvoir réaliser des requêtes. Même si un module de sauvegarde des requêtes a été créé afin de réaliser des requêtes que d'autre personne peuvent réutiliser, cette partie manque un peu d'ergonomie.

J'aurai souhaité aussi pouvoir rendre l'application un peu plus agréable. Même si l'application reste un produit professionnel, j'aurai voulu avoir le temps d'améliorer l'esthétique de l'application, mais j'ai fait le choix de privilégier le fonctionnel.

C. Interface SIG

Pour la partie plugin Qgis j'aurai aimé pouvoir utiliser la sauvegarde de connexion utilisé dans Qgis cela aurait permis au plugin de mieux correspondre au logiciel et d'avoir plus de fonctionnalités, telle que la sauvegarde de connexion multiple. Suite à un problème d'utilisation de la classe qui gère cette fonctionnalité sur l'api Qgis, je n'ai pas pu le faire.

Cette partie répond aux objectifs que l'on avait fixé en début de stage avec mon maitre de stage, je ne vois pas d'évolution possible sauf si le stockage des scans se faisait sur la base ou si on étendait la base aux scans lidar aérien, là il faudrait pouvoir afficher les scans aérien et terrestre.

Conclusion

Pour conclure on peut dire que le travail demandé a été atteint même si certain points demanderaient à être approfondis. Les objectifs fixés au début ont été réalisés, sauf pour la partie diagnostique. Pour la base de données, l'objectif a été atteint. Pour l'interface l'objectif a aussi été atteint malgré le fait qu'une amélioration de la partie requête serait souhaitable. Pour la partie Qgis l'objectif est atteint.

Ce stage ma permi de réaliser un projet en programmation de a à z et ainsi de voir comment on mène un projet informatique. Il m'a aussi permis d'apprendre un nouveau langage de programmation et qui plus est orienté objet. Cela me permet d'avoir acquis de vraies compétences en programmation, ainsi que des techniques d'organisation du code comme le modèle vue controller, malgré le fait qu'il me reste énormément de chose à apprendre.

Deux parties ont été dures durant le stage. La première a été l'apprentissage du C++ et de la programmation orientée objet et de toute les notions lié à la programmation objet comme l'héritage, le polymorphisme, l'encapsulation,… Mais le fait de travailler avec Qt à simplifié les choses car de nombreuses fonctions ont été créés et sont simples à réutiliser.

La seconde partie difficile que j'ai eu dans mon stage est sur la programmation en C++ avec Qgis, car peu de documentation sur l'utilisation des fonctions. Le peu de documentation que l'on trouve est obsolète et en python, il faut donc modifier le code, ce qui ne marche pas toujours du faite l'évolution des classes au fil des mises à jour.

Bibliographie

Livre :

- Guide du développeur PostgreSQL par Ewald Geschwinde et Hans_Jürgen Schönig édition CampusPress.
- Les cahiers du programmeur PostgreSQL par Stéphane Mariel édition EYROLLES.
- Le QuickGuide Développeur SQL par Ben Forta édition CampusPress.
- Qt4 et C++ Programmation d'interfaces GUI par Jasmin Blanchette et Mark Summerfield édition CampusPress.
- Vade-Mecum du forestier par la société forestière de franche-comté.
- Mémoire de fin d'étude Emilie Paris
- Mémoire de stage d'Aurélie Colin

PDF :

- Quantum GIS Coding and Compilation Guide 1.6
- Quantum GIS Manuel de l'utilisateur 1.7.2
- Quantum GIS Manuel Installation, Utilisation, Programmation 1.0.0

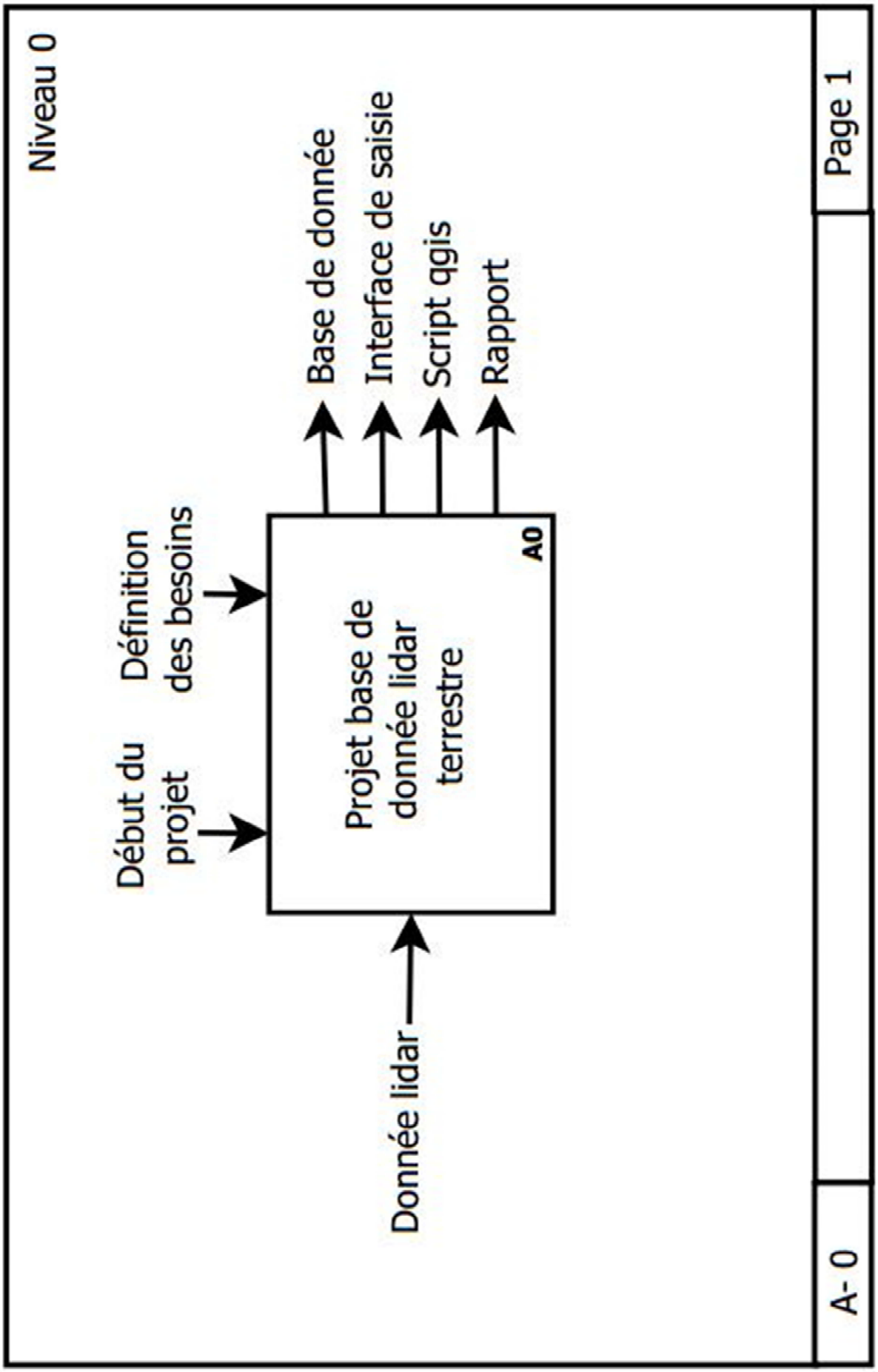
Internet :

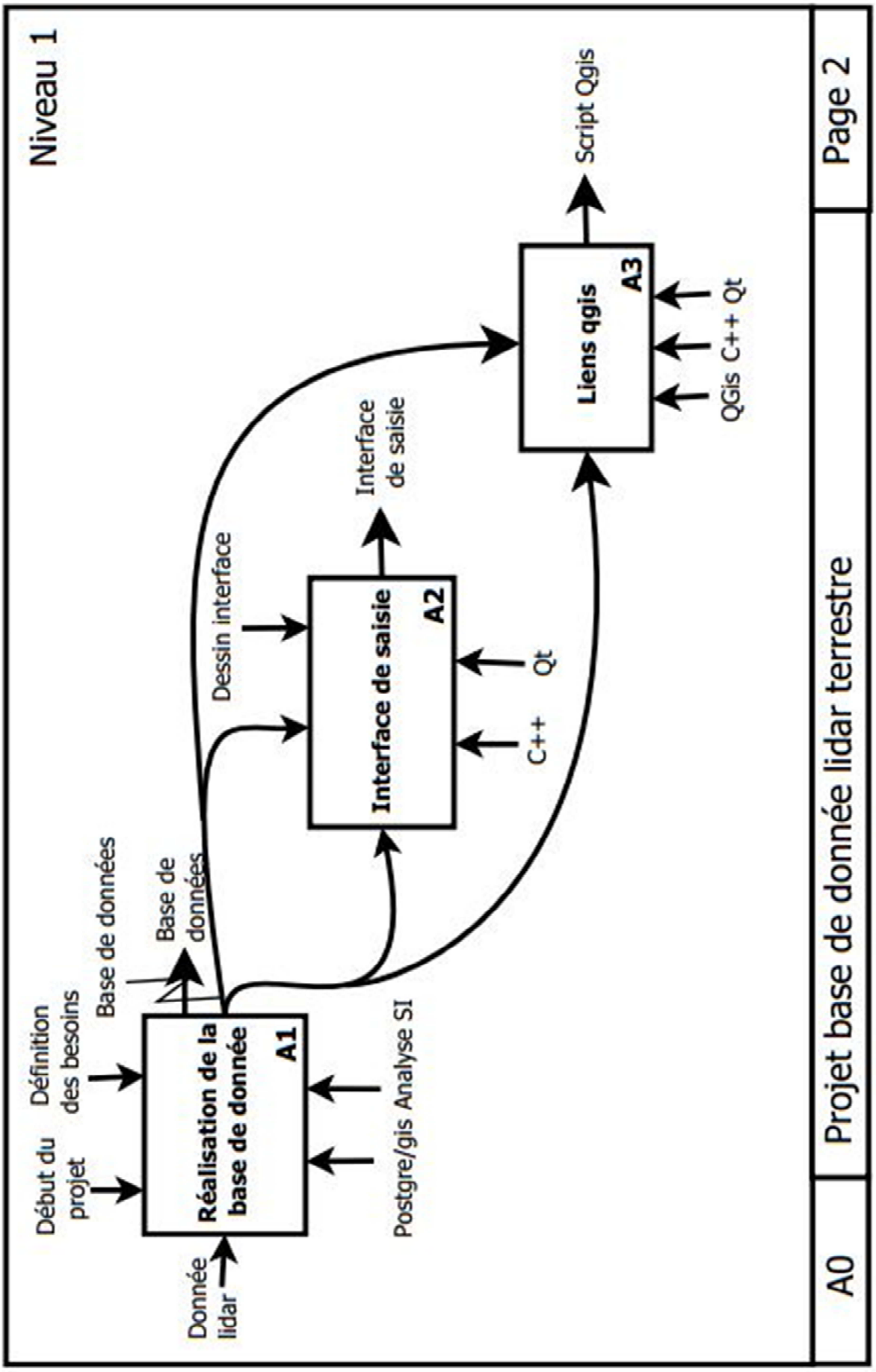
- Le site du Zéro qui ma permit d'apprendre le C++ assez facilement via des exercices et un tutoriel agréable.
- Le site du Framework Qt qui ma permit d'avoir une documentation sur toute les classes du Framework.
- Developper.com forum pour les développeurs en informatique ou j'ai pu trouver des solutions à mes problèmes, mais aussi de la documentation en français sur les classes de Qt.
- Forumsig ou j'ai pu trouver des solutions à mes problèmes.
- Qt France forum sur qt ou j'ai pu trouver des solutions à mes problèmes.
- Site du logiciel PostgreSQL ou on a ont a pu trouver le documentation en ligne.
- Qt Quarterly propose des articles sur comment utiliser des fonctions de Qt cela ma permit notamment de réaliser l'affichage formulaire.
- Qgis API ou l'on trouve une documentation sur les classes de Qgis.
- Pyqgis-cookbook site ou l'on trouve une documentation sur l'utilisation de l'api Qgis en python.

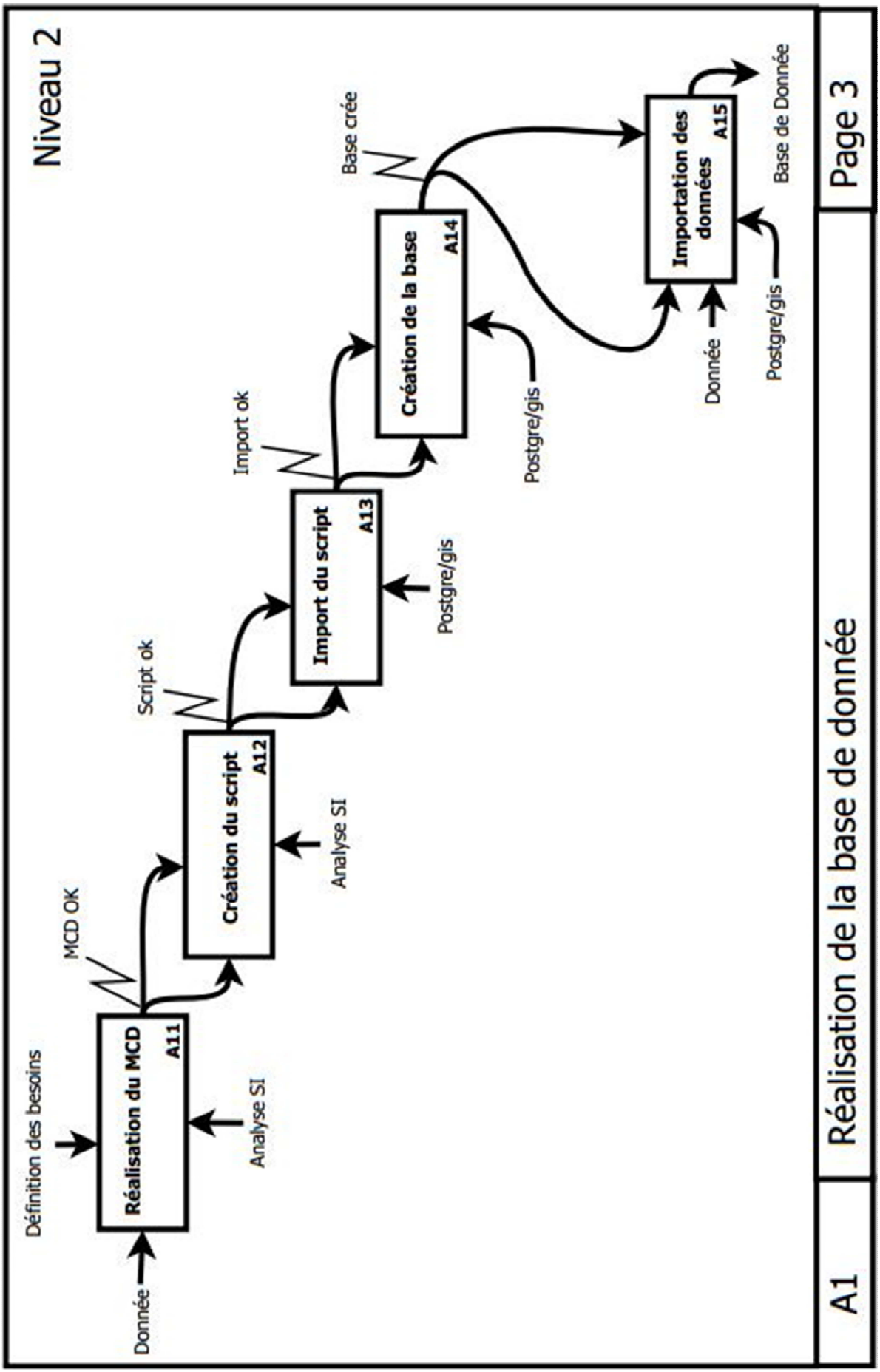
Tables des illustrations

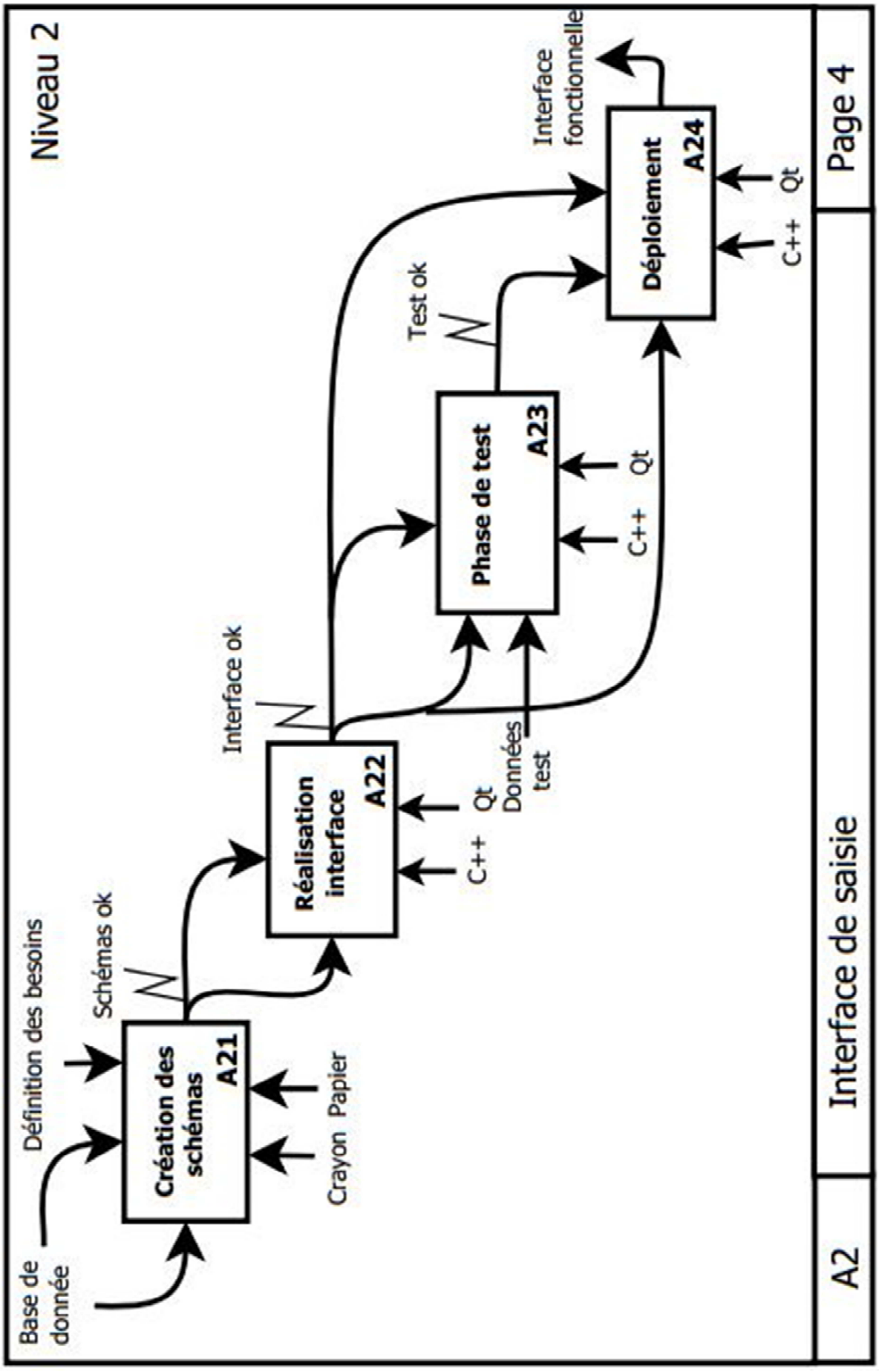
Annexes

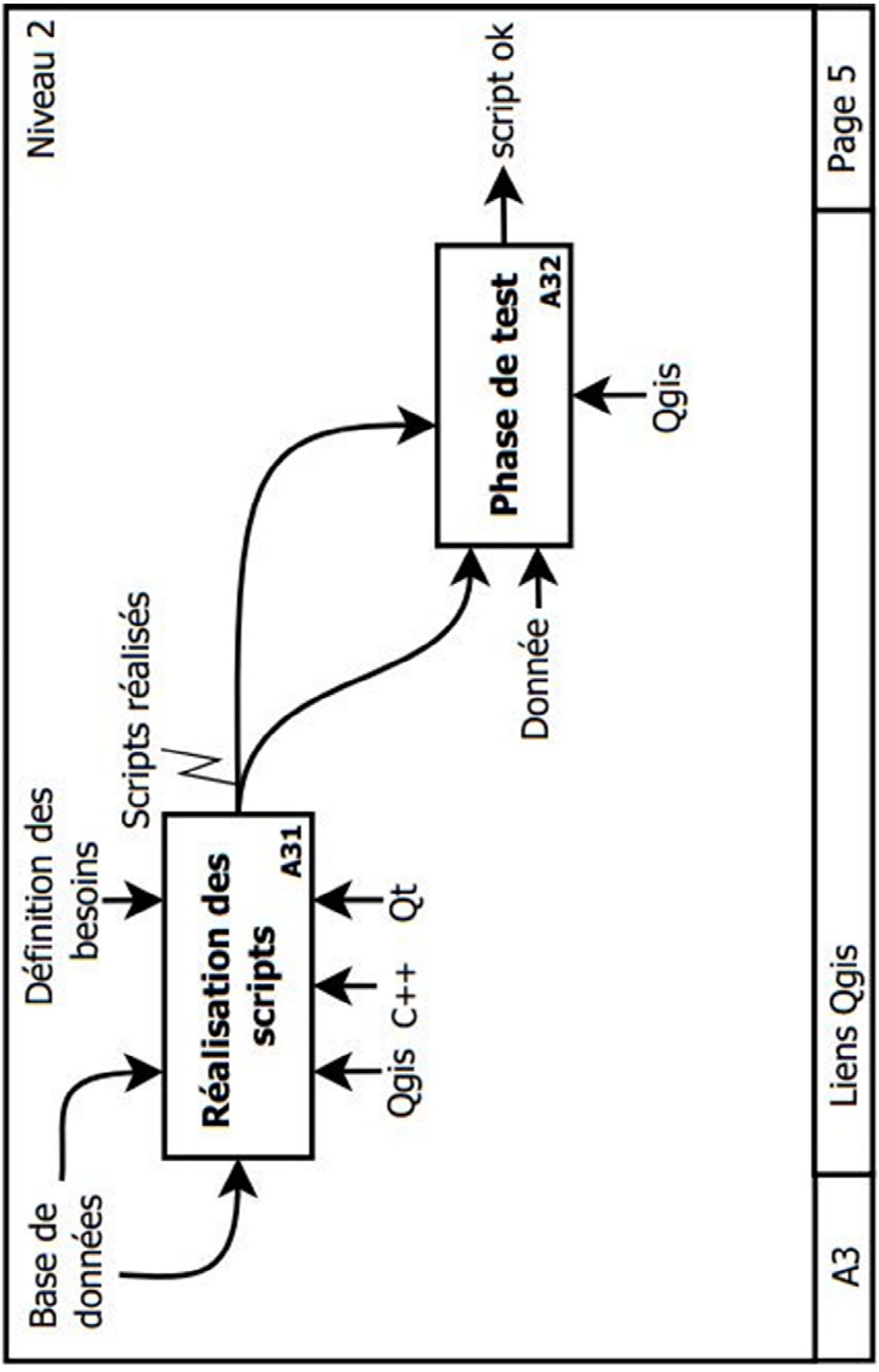
Annexe 1 : SADT

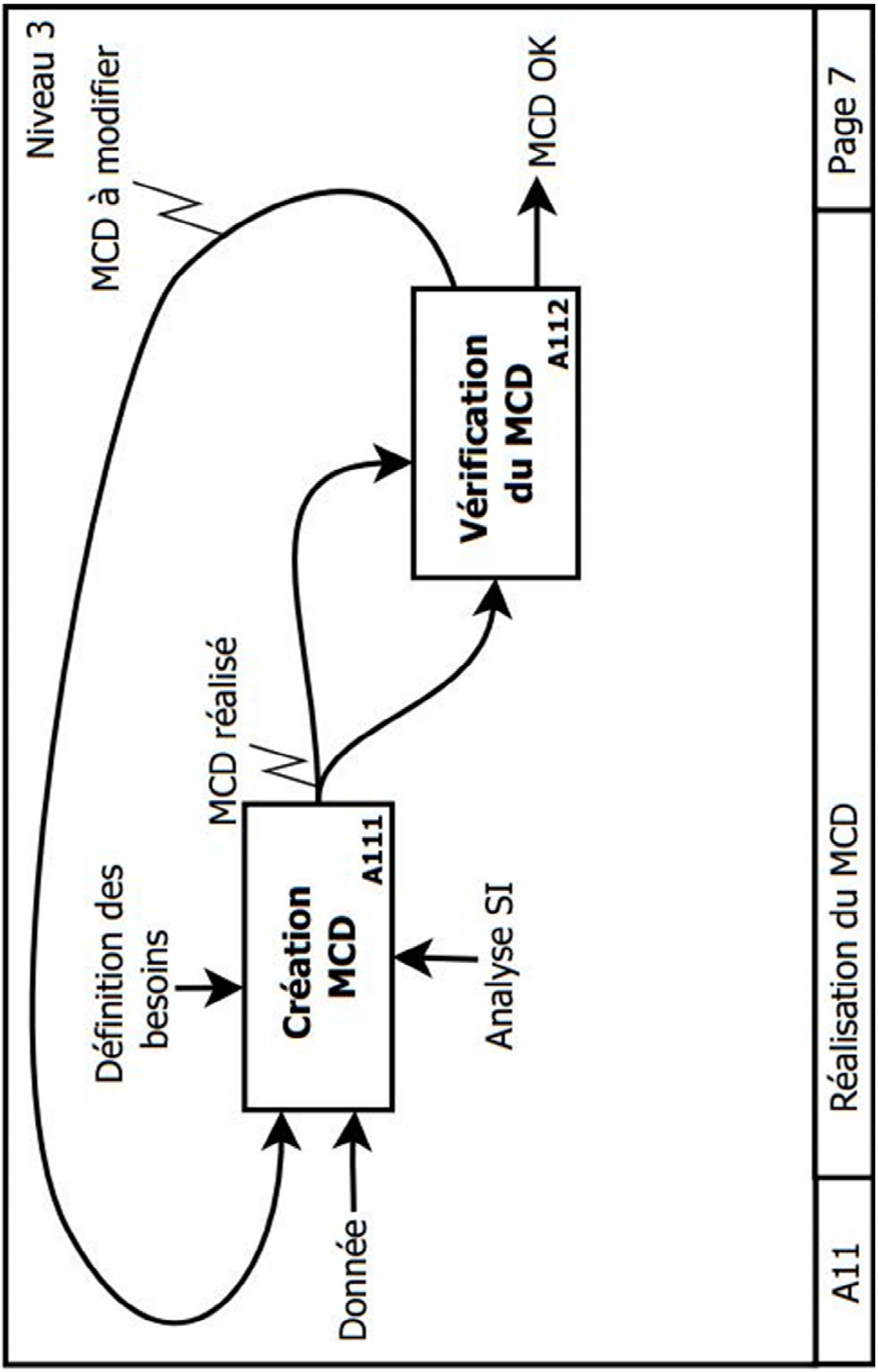

Niveau 0
Page 1
A- 0
Base de donnée
Interface de saisie
Script qgis
Rapport
Définition des besoins
Début du projet
Projet base de donnée lidar terrestre
A0
Donnée lidar

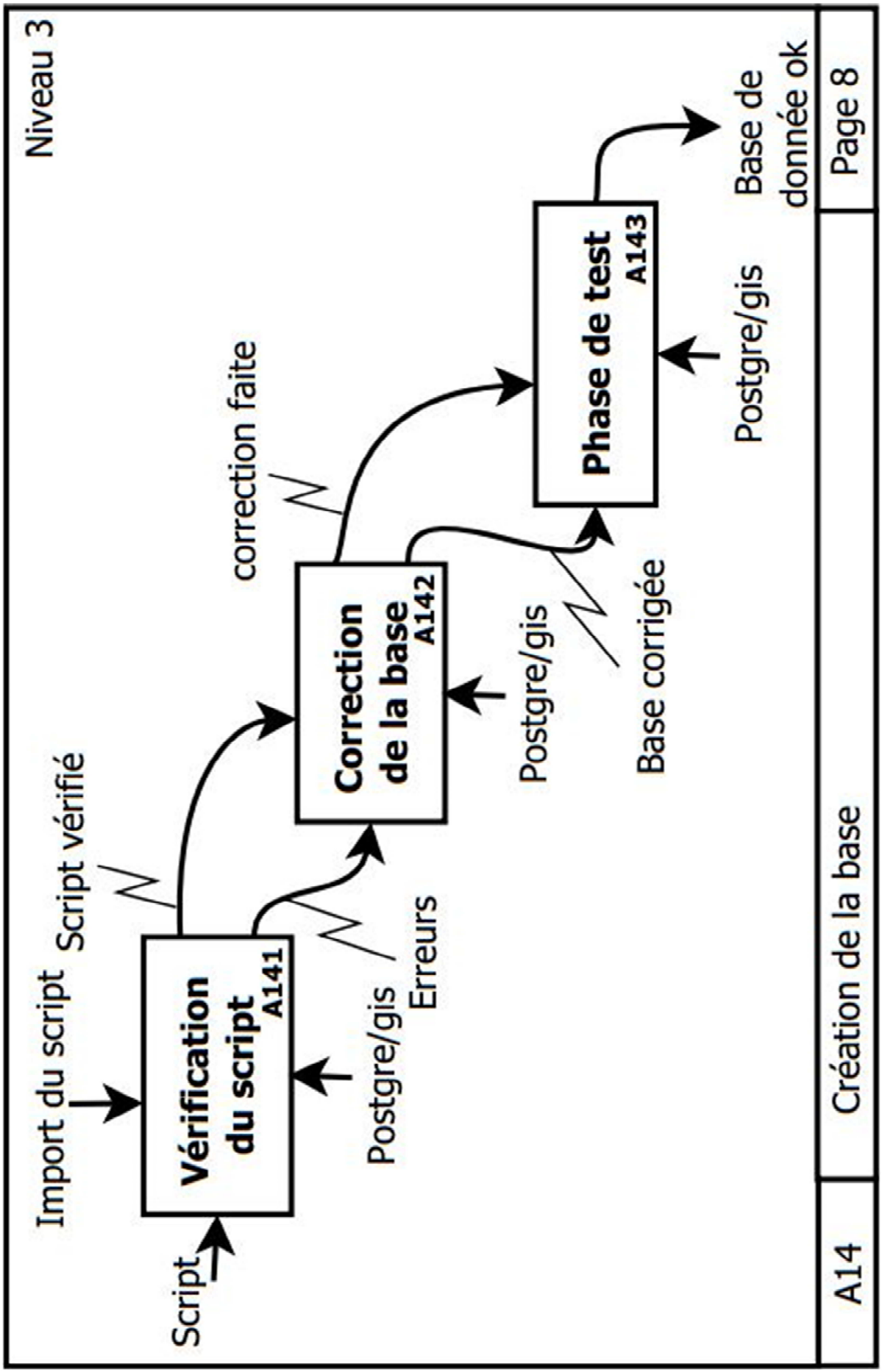

Niveau 1
A0
Projet base de donnée lidar terrestre
Page 2
Début du projet
Définition des besoins
Donnée lidar
Réalisation de la base de donnée
A1
Postgre/gis Analyse SI
Base de données
Base de données
Dessin interface
Interface de saisie
A2
C++
Qt
Interface de saisie
Liens qgis
A3
QGis C++ Qt
Script Qgis


Niveau 2
A1
Réalisation de la base de donnée
Page 3
Définition des besoins
Réalisation du MCD A11
Analyse SI
Donnée
MCD OK
Création du script A12
Analyse SI
Script ok
Import du script A13
Postgre/gis
Import ok
Création de la base A14
Postgre/gis
Base crée
Importation des données A15
Donnée
Postgre/gis
Base de Donnée


Niveau 2
A2
Interface de saisie
Page 4
Base de donnée
Définition des besoins
Création des schémas A21
Crayon Papier
Schémas ok
Réalisation interface A22
C++ Qt
Interface ok
Phase de test A23
C++ Qt
Données test
Test ok
Déploiement A24
C++ Qt
Interface fonctionnelle


Niveau 2
Définition des besoins
Scripts réalisés
Base de données
Réalisation des scripts
A31
Qgis
C++
Qt
Phase de test
A32
Donnée
Qgis
script ok
A3
Liens Qgis
Page 5


Niveau 3
MCD à modifier
MCD OK
Vérification du MCD
A112
MCD réalisé
Définition des besoins
Création MCD
A111
Analyse SI
Donnée
A11
Réalisation du MCD
Page 7


Niveau 3
A14
Création de la base
Page 8
Script
Import du script
Vérification du script A141
Script vérifié
Postgre/gis
Erreurs
Correction de la base A142
correction faite
Postgre/gis
Base corrigée
Phase de test A143
Postgre/gis
Base de donnée ok

Annexe 2 : Dictionnaire de données

algorithme				
Champs	Type	Caractéristique	Définition	Remarque
alg_id	Integer	Clef Primaire	id	
alg_nom	Text		nom de l'algorithme	
alg_algo	Varchar		algorithme	
alg_lien	Varchar		lien ver l'algorithme	

arbre_numerique				
Champs	Type	Caractéristique	Définition	Remarque
arn_id	Integer	Clef Primaire	Id	
arn_num	Integer		Numéro de l'arbre	
arn_nom	Text		Nom si arbre remarquable	
arn_diametre	Double Précision		Diametre de l'arbre	
arn_circonference	Double Précision		Circonférence de l'arbre	
arn_observation	Text		Observation	
arn_coord_x_relat	Double Précision		Coordonnée relatif au centre du scan	
arn_coord_y_relat	Double Précision		Coordonnée relatif au centre du scan	
sim_id	Integer	Clef Etrangère	Simulation auquelle se rapport l'arbre	
ess_id	Integer	Clef Etrangère	Essence de l'arbre	
the_geom	Géométrie		Colonne géographique pour affichage SIG	
arn_coord_x_abs	Double Précision		Coordonnée absolue Lambert 93	
arn_coord_y_abs	Double Précision		Coordonnée absolue Lambert 94	
arn_representation	Double Précision		Permet de représenter le diametre ou la circonférence	

arbre_terrain				
Champs	Type	Caractéristique	Définition	Remarque
arb_id	Integer	Clef Primaire	Id	
arb_num	Integer		Numéro de l'arbre	
arb_nom	Text		Nom si arbre remarquable	
arb_azimut	Double Précision		Azimut par rapport au centre de la placette	En grade
arb_distance	Double Précision		Distance par rapport au centre de la placette	
arb_diam1	Double Précision		Diametre de l'arbre	
arb_diam2	Double Précision		Second diametre de l'arbre	
arb_circonference	Double Précision		Circonférence de l'arbre	
arb_coord_x_calc	Double Précision		Coordonnée absolue calculer avec coordonnée de la placette et azimut distance	
arb_coord_y_calc	Double Précision		Coordonnée absolue calculer avec coordonnée de la placette et azimut distance	
ess_id	Integer	Clef Etrangère	Essence de l'arbre	
st_id	Integer	Clef Etrangère	Statut de l'arbre	
arb_hauteur	Double Précision		Hauteur de l'arbre	
the_geom	Géométrie		Colonne géographique pour affichage SIG	
camt_id	Integer	Clef Etrangère	Campagne auquelle se rapporte l'arbre	
pla_id	Integer	Clef Etrangère	Placette de l'arbre	
arb_representation	Double Précision		Permet de représenter le diametre ou la circonférence	

campagne_laser				
Champs	Type	Caractéristique	Définition	Remarque
caml_id	Integre	Clef Primaire	Id	
caml_nom	Text		Nom de la campagne	
caml_date_deb	Date		Date de début	
caml_date_fin	Date		Date de fin	
tys_id	Integer	Clef_Etrangere	Type de la saison	

campagne_terrain				
Champs	Type	Caractéristique	Définition	Remarque
camt_id	Integer	Clef Primaire	Id	
camt_nom	Text		Nom de la campagne	
camt_date_deb	Date		Date de début	
camt_date_fin	Date		Date de fin	
pro_id	Integer	Clef Etrangère	Protocole utiliser lors de l'inventaire	
tys_id	Integer	Clef Etrangère	Type de la saison	

config				
Champs	Type	Caractéristique	Définition	Remarque
cof_id	Integer	Clef Primaire	Id	
cof_nom	Text		Nom de la configuration	
cof_facteur_resolution	Double Précision		Rélolution en facteur	Proportion de la résolution maximum
cof_nb_colonne	Integer		Résolution en colonne	
cof_nb_ligne	Integer		Résolution en ligne	
cof_za_vert_min	Double Précision		Zone angulaire verticale minimum	
cof_za_vert_max	Double Précision		Zone angulaire verticale maximum	
cof_za_hori_min	Double Précision		Zone angulaire horizontale minimum	
cof_za_hori_max	Double Précision		Zone angulaire horizontale maximum	
cof_vitesse_scan	Double Précision		Vitesse de scan	En point par seconde
cof_facteur_vitesse	Integer		Facteur de la vitesse	
cof_dist_2pt_10_x	Integer		Distance en x entre 2 point à dix mètres	
cof_dist_2pt_10_y	Integer		Distance entre en y 2 point à dix mètres	
cof_couleur	Boolean		Utilisation de la couleur	
cof_fool_way_form	Boolean		Utilisation du fool way form	
cof_multi_echo	Boolean		Utilisation du multi echo	
scn_id	Integer	Clef Etrangère	Scanner utilisée	

essence				
Champs	Type	Caractéristique	Définition	Remarque
ess_id	Integer	Clef Primaire	Id	
ess_code	Text		Code de l'espece	
ess_nom_vernaculaire	Text		Nom commun	
ess_nom_latin	Text		Nom latin	

fichier				
Champs	Type	Caractéristique	Définition	Remarque
fic_id	Integer	Clef Primaire	Id	
fic_chemin	Varchar		Chemin du fichier en relatif à partir du dossier data	
fic_couleur	Boolean		Fichier en couleur?	
fic_reflectance	Boolean		Fichier en reflectance?	
sca_id	Integer	Clef Etrangere	Scan du fichier	
for_id	Integer	Clef Etrangere	Format du fichier	

filtre_config				
Champs	Type	Caractéristique	Définition	Remarque
cof_id	Integer	Clef Primaire et Etrangere	Configuration	
fim_id	Integer	Clef Primaire et Etrangere	Filtre materiel	

filtre_fichier				
Champs	Type	Caractéristique	Définition	Remarque
fic_id	Integer	Clef Primaire et Etrangere	Fichier	
fil_id	Integer	Clef Primaire et Etrangere	Filtre logiciel	

filtre_logiciel				
Champs	Type	Caractéristique	Définition	Remarque
fil_id	Integer	Clef Primaire	Id	
fil_libelle	Text		Nom du filtre	
fil_description	Text		Description	
soc_id	Integer	Clef Etrangere	Societer qui le produit	

filtre_materiel				
Champs	Type	Caractéristique	Définition	Remarque
fim_id	Integer	Clef Primaire	Id	
fim_libelle	Text		Nom du filtre	
fim_description	Text		Description	
scn_id	Integer	Clef Etrangere	Scanner	

format				
Champs	Type	Caractéristique	Définition	Remarque
for_id	Integer	Clef Primaire	Id	
for_libelle	Text		format utilisé	

fusionner				
Champs	Type	Caractéristique	Définition	Remarque
sca_id	Integer	Clef_Primaire et Etrangere	scan fusionner	
sca_id_2	Integer	Clef_Primaire et Etrangere	scan avec lequel il a été fusionner	

localisation				
Champs	Type	Caractéristique	Définition	Remarque
loc_id	Integer	Clef Primaire	Id	
loc_num_parcelle	Text		Numéro de la parcelle	
loc_code_foret	Text		Code de la foret	
loc_nom_foret	Text		Nom de la foret	
loc_nom	Text		code foret concatener avec un _ et numero de parcelle	

operateur				
Champs	Type	Caractéristique	Définition	Remarque
op_id	Integer	Clef Primaire	Id	
op_nom	Text		Nom de l'opérateur	
op_prenom	Text		Prénom de l'opérateur	

operateur_laser				
Champs	Type	Caractéristique	Définition	Remarque
caml_id	Integer	Clef_Primaire et Etrangere	campagne laser sur laquelle à travailler l'opérateur ci-dessous	
op_id	Integer	Clef_Primaire et Etrangere	Opérateur sur la campagne donner ci-dessus	

operateur_terrain				
Champs	Type	Caractéristique	Définition	Remarque
op_id	Integer	Clef_Primaire et Etrangere	Opérateur sur la campagne donner ci-dessous	
camt_id	Integer	Clef_Primaire et Etrangere	campagne terrain sur laquelle à travailler l'opérateur ci-dessus	

orientation				
Champs	Type	Caractéristique	Définition	Remarque
ori_id	Integer	Clef Primaire	Id	
ori_code	Text		Code point cardinaux	
ori_libelle	Text		Point cardinaux en toute lettre	

placette				
Champs	Type	Caractéristique	Définition	Remarque
pla_id	Integer	Clef Primaire	Id	
pla_code	Text		Code ou numero de la placette	
pla_coord_x	Double_precision		Coordonnée absolue en Lambert 93	
pla_coord_y	Double_precision		Coordonnée absolue en Lambert 94	
pla_coord_z	Double_precision		Coordonnée absolue en Lambert 95	
pla_remarque	Text		Remarque	
loc_id	Integer	Clef Etrangere	Localisation	
sit_id	Integer	Clef Etrangere	Site auquelle appartient la placette	
pla_pente	Integer		Pente au niveau de la placette	En %
ori_id	Integer	Clef Etrangere	Orientation de la placette	
the_geom	Géométrie		Colonne géographique pour affichage SIG	

protocole				
Champs	Type	Caractéristique	Définition	Remarque
pro_id	Integer	Clef Primaire	Id	
pro_lien	Varchar		Lienvers le fichier de protocole	
pro_description	Text		Description	

protocole_donnee				
Champs	Type	Caractéristique	Définition	Remarque
pro_id	Integer	Clef_Primaire et Etrangere	Lien vers protocole	
tyd_id	Integer	Clef_Primaire et Etrangere	Lien vers type de donnée	

protocole_placette				
Champs	Type	Caractéristique	Définition	Remarque
pro_id	Integer	Clef_Primaire et Etrangere	Lien vers protocole	
typ_id	Integer	Clef_Primaire et Etrangere	Lien vers type de placette	

reseau				
Champs	Type	Caractéristique	Définition	Remarque
res_id	Integer	Clef Primaire	Id	
res_libelle	Text		Nom du reseau de site	
res_description	Text		Description	

scan				
Champs	Type	Caractéristique	Définition	Remarque
sca_id	Integer	Clef Primaire	Id	
sca_description	Text		Description du scan	
sca_apercu	Varchar		lien vers le fichier d'apercu	
sca_coord_x_calc	Double Précision		Coordonnee absolue calculer par rapport au centre de la placette en Lambert 93	
sca_coord_y_calc	Double Précision		Coordonnee absolue calculer par rapport au centre de la placette en Lambert 93	
sca_coord_z_calc	Double Précision		Coordonnee absolue calculer par rapport au centre de la placette en Lambert 93	
sca_azimut_0	Double Précision		Azimut au zero du scan	En grade
sca_nord_geographique	Boolean		Si azimut = 0 alors le scan est au nord geographique	
sca_date	Date		Date de prise du scan	
caml_id	Integer	Clef Etrangère	Campagne lidar du scan	
pla_id	Integer	Clef Etrangère	Placette ou ce situe le scan	
cof_id	Integer	Clef Etrangère	Configuration du scan	
sca_coord_x_mesure	Double Précision		Coordonnée relative du scan au centre de la placette	
sca_coord_y_mesure	Double Précision		Coordonnée relative du scan au centre de la placette	
sca_coord_z_mesure	Double Précision		Coordonnée relative du scan au centre de la placette	
the_geom	Géométrie		Colonne géographique pour affichage SIG	

scanner				
Champs	Type	Caractéristique	Définition	Remarque
scn_id	Integer	Clef_Primaire	Id	
scn_num_serie	Varchar		Numéro de série de l'appareil	
scn_modele	Text		Nom du modele	
scn_boussole	Boolean		Possède t'il une boussole?	
scn_altimetre	Boolean		Possède t'il un altimètre?	
scn_gps	Boolean		Possède t'il un GPS?	
scn_inclinometre	Boolean		Possède t'il un inclinomètre?	
scn_fool_way_form	Boolean		Possède t'il un filtre fool way form?	
scn_multi_echo	Boolean		Possède t'il un filtre multi echo?	
scn_caracteristique	Text		Caracteristique particulière	
soc_id	Integer	Clef Etrangère	Societer de fabrication	
scn_portee_min	Double Précision		Portée minimum du scan	
scn_taux_scan_max	Integer		Taux de scan maximum	
scn_resolution	Double Précision		Résolution maximum	
scn_vitesse_max	Integer		Vitesse maximum de scan	
scn_portee_max	Double Précision		Portée maximum du scan	

simulation				
Champs	Type	Caractéristique	Définition	Remarque
sim_id	Integer	Clef Primaire	Id	
fic_id	Integer	Clef Etrangère	Lien avec un fichier	
alg_id	Integer	Clef Etrangère	Lien avec un algorithme	
op_id	Integer	Clef Etrangère	Operateur pour la simulation	

site				
Champs	Type	Caractéristique	Définition	Remarque
sit_id	Integer	Clef Primaire	Id	
sit_nom	Text		Nom du site	
res_id	Integer	Clef Etrangère	Reseau auquelle appartient le site	

societee				
Champs	Type	Caractéristique	Définition	Remarque
soc_id	Integer	Clef Primaire	Id	
soc_libelle	Text		Nom de la société	

status				
Champs	Type	Caractéristique	Définition	Remarque
st_id	Integer	Clef Primaire	Id	
st_libelle	Text		Status	

type_donnee				
Champs	Type	Caractéristique	Définition	Remarque
tyd_id	Integer	Clef Primaire	Id	
tyd_libelle	Text		Type de donnée	
tyd_description	Text		Description sur le type de donnée	

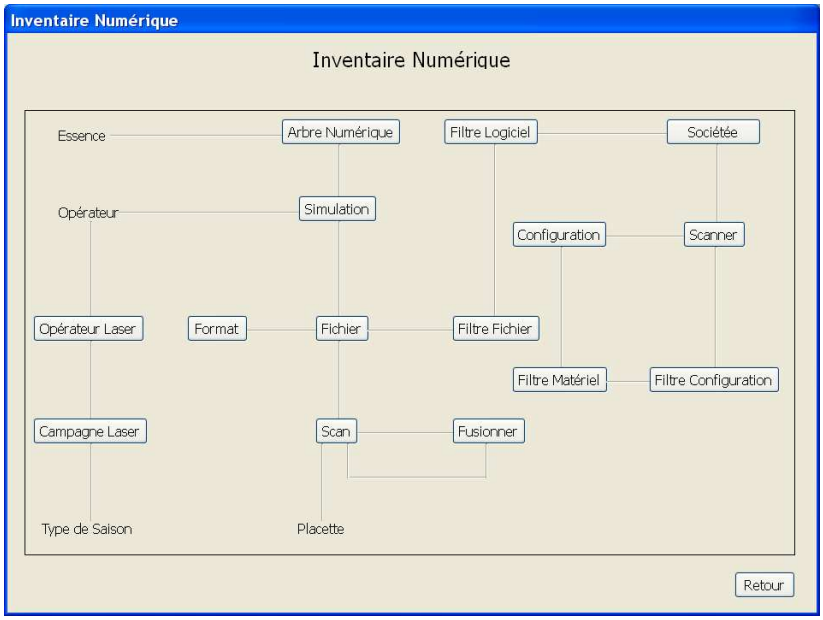
type_emprise				
Champs	Type	Caractéristique	Définition	Remarque
tye_id	Integer	Clef Primaire	Id	
tye_libelle	Text		Type d'emprise	

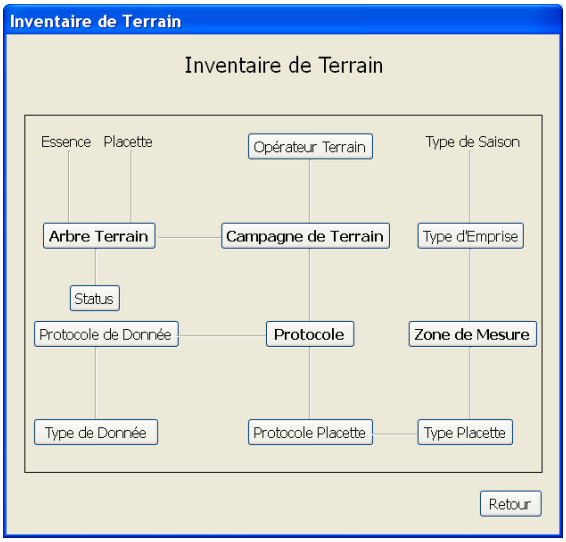
type_placette				
Champs	Type	Caractéristique	Définition	Remarque
typ_id	Integer	Clef Primaire	Id	
typ_libelle	Text		Type d'emprise	
zom_id	Integer	Clef Etrangère	Lien avec zone de mesure	

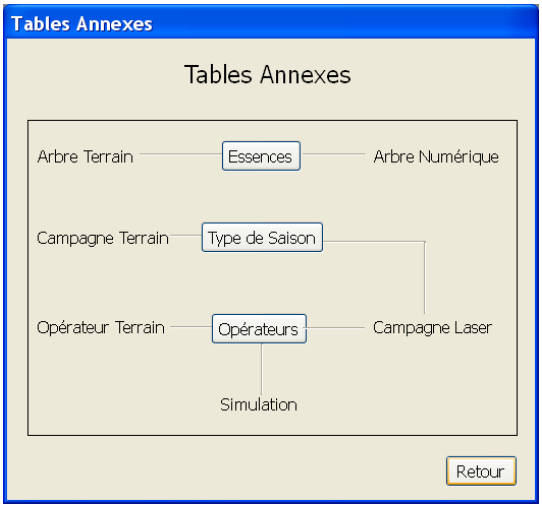
type_saison				
Champs	Type	Caractéristique	Définition	Remarque
tys_id	Integer	Clef Primaire	Id	
tys_libelle	Text		Type de saison	

zone_mesure				
Champs	Type	Caractéristique	Définition	Remarque
zom_id	Integer	Clef Primaire	Id	
zom_facteur	Double_précision		Facteur relascopique	Pour placette relascopique
zom_rayon	Double_précision		Diamètre du rayon	Pour placette circulaire
zom_l1	Double_précision		Largeur	Pour placette rectangulaire
zom_l2	Double_précision		Longueur	Pour placette rectangulaire
zom_precomptage	Integer		Diamètre de précomptage	
tye_id	Integer	Clef Etrangère	Type d'emprise	

Annexe 3 : Accès aux tables


Inventaire Numérique
Inventaire Numérique
Essence
Arbre Numérique
Filtre Logiciel
Sociétée
Opérateur
Simulation
Configuration
Scanner
Opérateur Laser
Format
Fichier
Filtre Fichier
Filtre Matériel
Filtre Configuration
Campagne Laser
Scan
Fusionner
Type de Saison
Placette
Retour


Inventaire de Terrain
Inventaire de Terrain
Essence
Placette
Opérateur Terrain
Type de Saison
Arbre Terrain
Campagne de Terrain
Type d'Emprise
Status
Protocole de Donnée
Protocole
Zone de Mesure
Type de Donnée
Protocole Placette
Type Placette
Retour


Tables Annexes
Tables Annexes
Arbre Terrain
Essences
Arbre Numérique
Campagne Terrain
Type de Saison
Opérateur Terrain
Opérateurs
Campagne Laser
Simulation
Retour

Annexe 4 : Interface utilisateur

main.cpp :

```
#include <QtGui/QApplication>
#include "gmainwindow.h"
#include "controller.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    Controller c;
    c.exec();
    return a.exec();
}
```

controller.h :

```
#ifndef CONTROLLER_H
#define CONTROLLER_H
#include "gmainwindow.h"
#include "databaseconnection.h"
#include "gconnectiondialog.h"
#include "databaseexport.h"
#include "calculcoord.h"
class Controller : public QObject
{
    Q_OBJECT
public:
    Controller();
    ~Controller();
    void exec();
private:
    GMainWindow*            _w;
    DataBaseConnection*     _dbc;
    GConnectionDialog*      _d;
    QMap<QString, QString>
_descriptionFields;
public slots:
    void quit();
    void showConnectionDialog();
    void closeConnectionDialog();
    void createConnection(QString
host, QString userName, QString
password, QString dbName, int
port);
    void showPlotManager();
    void chooseTableToShow(QDialog
* parent, QString code);
    void showTableDialog(QDialog*
parent, QString tableName, QString
title);
    void
showFieldInventoryManager();
    void
showNumericalInventoryManager();
    void
showOthersTablesManager();
    void
createDataBaseExport(QSqlRelationa
lTableModel *model,QString
filter,QString fileName);
    void
createQueryExport(QSqlQueryModel
*modelQuery,QString filter,QString
fileName);
    void showQueryDialog();
    void showQuery(QDialog
*parent,QSqlQueryModel
*modelQuery);
    void showImportData();
    void showCalcCoord();
    void calcCoord(QDialog
*parent,bool art,bool arn,bool
scan);
signals:
};
#endif // CONTROLLER_H
```

controller.cpp :

```
#include "controller.h"
#include <QDebug>
#include <QtGui/QMessageBox>
#include <QtSql>
#include "gconnectiondialog.h"
#include "gplotmanagementdialog.h"
#include "gtabledialog.h"
#include
"gfieldinventorymanagerdialog.h"
#include
"gnumericalinventorymanagerdialog.
h"
#include
"gotherstablesmanagerdialog.h"
#include "gquerydialog.h"
#include "gimportdatadialog.h"
#include "gcalccoorddialog.h"
#include <QFileDialog>
Controller::Controller()
{
}
Controller::~Controller()
{
    if (_w != NULL)     {delete
_w;}
    if (_dbc != NULL)   {delete
_dbc;}
}
void Controller::exec()
{
    _w = new GMainWindow();
    _dbc = NULL;
    // Connections
    connect(_w,
SIGNAL(closeMainWindows()), this,
SLOT(quit()));
    connect(_w,
SIGNAL(openPlotManager()), this,
SLOT(showPlotManager()));

connect(_w,SIGNAL(openFieldInvento
```

```
ryManager()),this,SLOT(showFieldIn
ventoryManager()));

connect(_w,SIGNAL(openNumericalInv
entoryManager()),this,SLOT(showNum
ericalInventoryManager()));

connect(_w,SIGNAL(openOthersTables
Manager()),this,SLOT(showOthersTab
lesManager()));

connect(_w,SIGNAL(openQueryDialog(
)),this,SLOT(showQueryDialog()));

connect(_w,SIGNAL(openImportDataDi
alog()),this,SLOT(showImportData()
));

connect(_w,SIGNAL(openCalcCoordDia
log()),this,SLOT(showCalcCoord()))
;
    _w->show();
    showConnectionDialog();
}
void Controller::quit()
{
    _w->close();
    if (_dbc != NULL) {_dbc-
>disconnect();}
}
void
Controller::showConnectionDialog()
{
    _d = new GConnectionDialog(_w,
"localhost", "postgres",
"Lidar159", "Lidar_test", 5432);
    connect(_d,
SIGNAL(connectionData(QString,QStr
ing,QString,QString,int)), this,
SLOT(createConnection(QString,QStr
ing,QString,QString,int)));
    connect(_d, SIGNAL(cancel()),
this,
SLOT(closeConnectionDialog()));
    _d->open();
}
void
Controller::closeConnectionDialog(
)
{
    _d->close();
    quit();
}
void
Controller::createConnection(QStri
ng host, QString userName, QString
password, QString dbName, int
port)
{
    if (_dbc != NULL) {delete
_dbc;}
    _dbc = new
DataBaseConnection(host, userName,
password, dbName, port, "QPSQL");
    bool ok = _dbc->connect();
    qDebug() << "Connection : " <<
ok;
    if (ok == true)
    {

_descriptionFields.clear();
        QSqlQuery
foreignKey(QString("SELECT
ccu.table_name AS
foreign_table_name,
ccu.column_name AS
foreign_column_name FROM
information_schema.table_constrain
ts AS tc JOIN
information_schema.key_column_usag
e AS kcu ON tc.constraint_name =
kcu.constraint_name JOIN
information_schema.constraint_colu
mn_usage AS ccu ON
ccu.constraint_name =
tc.constraint_name WHERE
constraint_type = 'FOREIGN KEY'
GROUP BY foreign_table_name,
foreign_column_name"));
        QSqlQueryModel* modelQuery
= new QSqlQueryModel();
        modelQuery-
>setQuery(foreignKey);
        int count = modelQuery-
>rowCount();
        for (int  i = 0; i <
count; i++)
        {
            QString
table_etrangere = modelQuery-
>record(i).value("foreign_table_na
me").toString();
            QSqlQuery oid
(QString("SELECT oid FROM pg_class
WHERE relname =
'%1'").arg(table_etrangere));
            QSqlQueryModel*
modelOid = new QSqlQueryModel();
            modelOid-
>setQuery(oid);
            int oid_table =
modelOid-
>record(0).value("oid").toInt();
            QSqlQuery description
(QString("SELECT obj_description
AS description_query FROM
obj_description(%1)").arg(oid_tabl
e));
            QSqlQueryModel*
modelDescription = new
QSqlQueryModel();
```

```
            modelDescription-
>setQuery(description);
            QString
description_table =
modelDescription-
>record(0).value("description_quer
y").toString();

_descriptionFields.insert(table_et
rangere, description_table);
        }

QMessageBox::information(0,
"Connection", "Connection réussie
!");
    }
    else
    {
        QMessageBox::warning(0,
"Connection", "Paramètre de
connexion invalide !");
        showConnectionDialog();
    }
}
void Controller::showPlotManager()
{
   GPlotManagementDialog *e = new
GPlotManagementDialog();
   connect(e,
SIGNAL(askForTable(QDialog*,
QString)), this,
SLOT(chooseTableToShow(QDialog*,
QString)));
   e->show();
}
void
Controller::showFieldInventoryMana
ger()
{
    GFieldInventoryManagerDialog
*f = new
GFieldInventoryManagerDialog();
    connect(f,
SIGNAL(askForTable(QDialog*,
QString)), this,
SLOT(chooseTableToShow(QDialog*,
QString)));
    f->show();
}
void
Controller::showNumericalInventory
Manager()
{

GNumericalInventoryManagerDialog
*g = new
GNumericalInventoryManagerDialog()
;
    connect(g,
SIGNAL(askForTable(QDialog*,
QString)), this,
SLOT(chooseTableToShow(QDialog*,
QString)));
    g->show();
}
void
Controller::showOthersTablesManage
r()
{
    GOthersTablesManagerDialog *h
= new
GOthersTablesManagerDialog();
    connect(h,
SIGNAL(askForTable(QDialog*,
QString)), this,
SLOT(chooseTableToShow(QDialog*,
QString)));
    h->show();
}
void Controller::showQueryDialog()
{
    GQueryDialog *i = new
GQueryDialog();

connect(i,SIGNAL(submit(QDialog*,Q
SqlQueryModel*)),this,SLOT(showQue
ry(QDialog*,QSqlQueryModel*)));
    i->show();
}
void Controller::showQuery(QDialog
*parent, QSqlQueryModel
*modelQuery)
{
    QString title = "Requête" ;
    GTableDialog* ui= new
GTableDialog(parent, title,
modelQuery, _descriptionFields);

connect(ui,SIGNAL(exportModeQuery(
QSqlQueryModel*,QString,QString)),
this,SLOT(createQueryExport(QSqlQu
eryModel*,QString,QString)));
    ui->show();
}
void Controller::showImportData()
{
    GImportDataDialog *j = new
GImportDataDialog();
    j->show();
}
void Controller::showCalcCoord()
{
    GCalcCoordDialog *k = new
GCalcCoordDialog();

connect(k,SIGNAL(calc(QDialog*,boo
l,bool,bool)),this,SLOT(calcCoord(
QDialog*,bool,bool,bool)));
    k->show();
}
```

```
void
Controller::chooseTableToShow(QDia
log* parent, QString code)
{
    if      (code == "Placette")
{showTableDialog(parent,
"placette", code);}
    else if (code ==
"Orientation")
{showTableDialog(parent,
"orientation", code);}
    else if (code ==
"Localisation")
{showTableDialog(parent,
"localisation", code);}
    else if (code == "Site")
{showTableDialog(parent, "site",
code);}
    else if (code == "Réseau")
{showTableDialog(parent, "reseau",
code);}
    else if (code == "Status")
{showTableDialog(parent, "status",
code);}
    else if (code == "Arbre
Terrain")
{showTableDialog(parent,
"arbre_terrain", code);}
    else if (code == "Campagne de
Terrain")
{showTableDialog(parent,
"campagne_terrain", code);}
    else if (code == "Type
d'Emprise")
{showTableDialog(parent,
"type_emprise", code);}
    else if (code == "Protocole de
Donnée")  {showTableDialog(parent,
"protocole_donnee", code);}
    else if (code == "Protocole")
{showTableDialog(parent,
"protocole", code);}
    else if (code == "Zone de
Mesure")
{showTableDialog(parent,
"zone_mesure", code);}
    else if (code == "Type de
Donnée")
{showTableDialog(parent,
"type_donnee", code);}
    else if (code == "Protocole
Placette")
{showTableDialog(parent,
"protocole_placette", code);}
    else if (code == "Type
Placette")
{showTableDialog(parent,
"type_placette", code);}
    else if (code == "Opérateur
Terrain")
{showTableDialog(parent,
"operateur_terrain", code);}
    else if (code == "Arbre
Numérique")
{showTableDialog(parent,
"arbre_numerique", code);}
    else if (code == "Filtre
Logiciel")
{showTableDialog(parent,
"filtre_logiciel", code);}
    else if (code == "Sociétée")
{showTableDialog(parent,
"societee", code);}
    else if (code == "Simulation")
{showTableDialog(parent,
"simulation", code);}
    else if (code ==
"Configuration")
{showTableDialog(parent, "config",
code);}
    else if (code == "Scanner")
{showTableDialog(parent,
"scanner", code);}
    else if (code == "Opérateur
Laser")
{showTableDialog(parent,
"operateur_laser", code);}
    else if (code == "Format")
{showTableDialog(parent, "format",
code);}
    else if (code == "Fichier")
{showTableDialog(parent,
"fichier", code);}
    else if (code == "Filtre
Fichier")
{showTableDialog(parent,
"filtre_fichier", code);}
    else if (code == "Filtre
Configuration")
{showTableDialog(parent,
"filtre_configuration", code);}
    else if (code == "Filtre
Matériel")
{showTableDialog(parent,
"filtre_materiel", code);}
    else if (code == "Campagne
Laser")
{showTableDialog(parent,
"campagne_laser", code);}
    else if (code == "Scan")
{showTableDialog(parent, "scan",
code);}
    else if (code == "Fusionner")
{showTableDialog(parent,
"fusionner", code);}
    else if (code == "Essences")
{showTableDialog(parent,
"essence", code);}
    else if (code == "Opérateurs")
{showTableDialog(parent,
"operateur", code);}
```

```
    else if (code == "Type de
Saison")
{showTableDialog(parent,
"type_saison", code);}
    else {
        QMessageBox msgBox;

msgBox.setText(QString("Pas de
table correspondant à la demande :
%1").arg(code));
        msgBox.exec();
    }
}
void
Controller::showTableDialog(QDialo
g* parent, QString tableName,
QString title)
{
    if (_dbc != NULL)
    {
        QSqlRelationalTableModel*
model = _dbc-
>getTableModel(tableName);
        if(model->rowCount()!=0)
        {
            GTableDialog* ui= new
GTableDialog(parent, title, model,
_descriptionFields);
            ui->show();

connect(ui,SIGNAL(exportMode(QSqlR
elationalTableModel*,QString,QStri
ng)),this,SLOT(createDataBaseExpor
t(QSqlRelationalTableModel*,QStrin
g,QString)));
        }
        else
        {

QMessageBox::information(0,
"Problème", "La table est vide
vous ne pouvez y accédez !");
        }
    }
    else {
        QMessageBox msgBox;
        msgBox.setText("Pas de
base connectée");
        msgBox.exec();
    }
}
void
Controller::createDataBaseExport(Q
SqlRelationalTableModel *model,
QString filter, QString fileName)
{
    DataBaseExport* dbe = new
DataBaseExport(fileName, filter);
    dbe->DataExport(model);
}
void
Controller::createQueryExport(QSql
QueryModel *modelQuery,QString
filter,QString fileName)
{
    DataBaseExport* dbe = new
DataBaseExport(fileName, filter);
    dbe->QueryExport(modelQuery);
}
void
Controller::calcCoord(QDialog*
parent, bool art, bool arn, bool
scan)
{
    new CalculCoord(parent ,art
,arn ,scan );
}
```

calculcoord.h :

```
#ifndef CALCULCOORD_H
#define CALCULCOORD_H
#include "controller.h"
#include "qsqlquery.h"
#include <QProgressDialog>
class CalculCoord : QObject
{
    Q_OBJECT
public:
    CalculCoord(QDialog
*parent,bool art,bool arn,bool
scan);
public slots:
    void perform();
    void cancel();
private:
    QTimer*          _t;
    QProgressDialog*
_progressDialog;
    int              _steps;
};
#endif // CALCULCOORD_H
```

calculcoord.cpp :

```
#include "calculcoord.h"
#include <QMessageBox>
#include <QtSql>
CalculCoord::CalculCoord(QDialog
*parent, bool art, bool arn, bool
scan)
{
    if (art == true)
    {
        QSqlQuery("UPDATE
arbre_terrain SET arb_coord_x_calc
= placette.pla_coord_x +
(arb_distance*SIN((arb_azimut/200)
*PI())),arb_coord_y_calc =
placette.pla_coord_y +
```

```
(arb_distance*COS((arb_azimut/200)
*PI())),the_geom =
st_setsrid(st_makepoint(arb_coord_
x_calc, arb_coord_y_calc),2154)
FROM placette WHERE
arbre_terrain.pla_id =
placette.pla_id");
    }
    if (scan == true)
    {
        QSqlQuery("UPDATE scan SET
sca_coord_x_calc =
placette.pla_coord_x +
scan.sca_coord_x_mesure,
sca_coord_y_calc =
placette.pla_coord_y +
scan.sca_coord_x_mesure, the_geom
=
st_setsrid(st_makepoint(sca_coord_
x_calc, sca_coord_y_calc),2154)
FROM placette WHERE scan.pla_id =
placette.pla_id ");
    }
    if (arn == true)
    {
        QSqlQuery("UPDATE
arbre_numerique SET
arn_coord_x_abs =
scan.sca_coord_x_calc +
arbre_numerique.arn_coord_x_relat,
arn_coord_y_abs =
scan.sca_coord_y_calc +
arbre_numerique.arn_coord_y_relat,
the_geom =
st_setsrid(st_makepoint(arn_coord_
x_abs, arn_coord_y_abs),2154) FROM
scan,fichier,simulation WHERE
(arbre_numerique.sim_id =
simulation.sim_id) AND
(simulation.fic_id =
fichier.fic_id) AND
(fichier.sca_id = scan.sca_id) ");
    }
    QMessageBox::information(0,
"Calcul Coordonnées", "Calcul
réussie");
}
void CalculCoord::perform()
{
    _progressDialog-
>setValue(_steps);
}
void CalculCoord::cancel()
{
    _t->stop();
}
```

databaseconnection.h :

```
#ifndef DATABASECONNECTION_H
#define DATABASECONNECTION_H
#include "qstring.h"
#include <QtSql>
class DataBaseConnection
{
public:
    DataBaseConnection(QString
host = "localhost", QString
userName = "postgres", QString
password = "Lidar159", QString
dbName = "Lidar_test", int port =
5432, QString driver = "QPSQL");
    bool connect();
    void disconnect();
    QSqlRelationalTableModel*
getTableModel(QString tableName,
QSqlRelationalTableModel::EditStra
tegy strategy =
QSqlRelationalTableModel::OnFieldC
hange);
private:
    QString     _host;
    QString     _password;
    QString     _userName;
    QString     _dbName;
    int         _port;
    QString     _driver;
    QSqlDatabase _db;
public slots:
};
#endif // DATABASECONNECTION_H
```

databaseconnection.cpp :

```
#include "databaseconnection.h"
#include "qdebug.h"
#include <QtSql>
#include <QString>
#include "qtableview.h"
#include
"qsqlrelationaldelegate.h"
DataBaseConnection::DataBaseConnec
tion(QString host, QString
userName, QString password,
QString dbName, int port, QString
driver)
{
    _host = host;
    _password = password;
    _userName = userName;
    _dbName = dbName;
    _port = port;
    _driver = driver;
}
bool DataBaseConnection::connect()
{
```

```
    _db =
QSqlDatabase::addDatabase(_driver)
;
    _db.setHostName(_host);
    _db.setDatabaseName(_dbName);
    _db.setUserName(_userName);
    _db.setPassword(_password);
    _db.setPort(_port);
    return _db.open();
}
void
DataBaseConnection::disconnect()
{
    _db.close();
}
QSqlRelationalTableModel*
DataBaseConnection::getTableModel(
QString tableName,
QSqlRelationalTableModel::EditStra
tegy strategy)
{
    QSqlRelationalTableModel
*model = new
QSqlRelationalTableModel(0, _db);
    model->setTable(tableName);
    model-
>setEditStrategy(strategy);
    model-
>setSort(0,Qt::AscendingOrder);
    QSqlQuery
foreignKey(QString("SELECT
kcu.column_name, ccu.table_name AS
foreign_table_name,
ccu.column_name AS
foreign_column_name FROM
information_schema.table_constrain
ts AS tc JOIN
information_schema.key_column_usag
e AS kcu ON tc.constraint_name =
kcu.constraint_name JOIN
information_schema.constraint_colu
mn_usage AS ccu ON
ccu.constraint_name =
tc.constraint_name WHERE
constraint_type = 'FOREIGN KEY'AND
tc.table_name='%1'").arg(tableName
));
    QSqlQueryModel* modelQuery =
new QSqlQueryModel();
    modelQuery-
>setQuery(foreignKey);
    int count = modelQuery-
>rowCount();
    for (int  i = 0; i < count;
i++)
    {
        QString nom_de_champ =
modelQuery-
>record(i).value("column_name").to
String();
        QString table_etrangere =
modelQuery-
>record(i).value("foreign_table_na
me").toString();
        QString champ_etranger =
modelQuery-
>record(i).value("foreign_column_n
ame").toString();
        QSqlQuery oid
(QString("SELECT oid FROM pg_class
WHERE relname =
'%1'").arg(table_etrangere));
        QSqlQueryModel* modelOid =
new QSqlQueryModel();
        modelOid->setQuery(oid);
        int oid_table = modelOid-
>record(0).value("oid").toInt();
        QSqlQuery description
(QString("SELECT obj_description
AS description_query FROM
obj_description(%1)").arg(oid_tabl
e));
        QSqlQueryModel*
modelDescription = new
QSqlQueryModel();
        modelDescription-
>setQuery(description);
        QString description_table
= modelDescription-
>record(0).value("description_quer
y").toString();
        int index = model-
>fieldIndex(nom_de_champ);
        model-
>setRelation(index,QSqlRelation(ta
ble_etrangere, champ_etranger,
description_table));
    }
    model->select();
    return model;
}
```

databaseexport.h :

```
#ifndef DATABASEEXPORT_H
#define DATABASEEXPORT_H
#include <QString>
#include <QtSql>
class DataBaseExport
{
public:
    DataBaseExport(QString
fileName, QString filter);
    void
QueryExport(QSqlQueryModel
*modelQuery);
    void DataExport
(QSqlRelationalTableModel *model);
private:
    QString
_fileName;
```

```
    QSqlRelationalTableModel*
_model;
    QSqlQueryModel*
_modelQuery;
    QString
_separateur;
    QString
_filter;
    QSqlQueryModel*
_model2;
};
#endif // DATABASEEXPORT_H
```

databaseexport.cpp :

```
#include "databaseexport.h"
#include <QFile>
#include <QTextStream>
#include <QMessageBox>
#include <QDebug>
DataBaseExport::DataBaseExport(QSt
ring fileName, QString filter)
{
    _fileName = fileName;
    _filter = filter;
    _separateur = "\t";
    if(_filter == "Fichier CSV
(séparateur ;) (*.csv)")
    {
        _separateur = ";";
    }
    else
    {
        _separateur = "\t";
    }
}
void
DataBaseExport::DataExport(QSqlRel
ationalTableModel *model)
{
    _model = model;
    QFile file(_fileName);
    if
(!file.open(QIODevice::WriteOnly |
QIODevice::Text))
        return;
    QTextStream out(&file);
    int countRow = _model-
>rowCount();
    int countColumn = _model-
>columnCount();
    for (int  k = 0; k <
countColumn; k++)
    {
        if (_model-
>relation(k).isValid())
        {
            out << _model-
>relation(k).indexColumn()<<_separ
ateur;
            out << _model-
>headerData(k,Qt::Horizontal).toSt
ring() << _separateur;
        }
        else
        {
            out << _model-
>headerData(k,Qt::Horizontal).toSt
ring() << _separateur;
        }
    }
    out << "\n";
    for (int  i = 0; i < countRow;
i++)
    {
        for (int  j = 0; j <
countColumn; j++)
        {
            if (_model-
>relation(j).isValid())
            {
                QModelIndex index
= _model->index(i,j);
                QSqlTableModel*
model2 = _model->relationModel(j);
                QString tbl =
model2->tableName();
                QString cle =
_model->relation(j).indexColumn();
                QString rel =
_model->data(index).toString();
                QString header =
_model-
>headerData(j,Qt::Horizontal).toSt
ring();
                QSqlQueryModel* id
= new QSqlQueryModel();
                id-
>setQuery(QString("SELECT %1 FROM
%2 WHERE %3 = '%4' GROUP BY
%1").arg(cle).arg(tbl).arg(header)
.arg(rel));
                QString result =
id-
>record(0).value(cle).toString();
                out << result <<
_separateur;
                out << rel <<
_separateur ;
            }
            else
            {
                QModelIndex index
= _model->index(i,j);
                out << _model-
>data(index).toString() <<
_separateur ;
            }
        }
        out << "\n";
    }
```

```
    QMessageBox::information(0,
"Export", "Export réussie");
}
void
DataBaseExport::QueryExport(QSqlQu
eryModel *modelQuery)
{
    _modelQuery = modelQuery;
    QFile file(_fileName);
    if
(!file.open(QIODevice::WriteOnly |
QIODevice::Text))
        return;
    QTextStream out(&file);
    int countRow = _modelQuery-
>rowCount();
    int countColumn = _modelQuery-
>columnCount();
    for (int  k = 0; k <
countColumn; k++)
    {
       out << _modelQuery-
>headerData(k,Qt::Horizontal).toSt
ring() << _separateur;
    }
    out << "\n";
    for (int  i = 0; i < countRow;
i++)
    {
        for (int  j = 0; j <
countColumn; j++)
        {
            QModelIndex index =
_modelQuery->index(i,j);
            out << _modelQuery-
>data(index).toString() <<
_separateur ;
        }
        out << "\n";
    }
    QMessageBox::information(0,
"Export", "Export réussie");
}
```

gcalccoorddialog.h :

```
#ifndef GCALCCOORDDIALOG_H
#define GCALCCOORDDIALOG_H
#include <QDialog>
namespace Ui {
class GCalcCoordDialog;
}
class GCalcCoordDialog : public
QDialog
{
    Q_OBJECT

public:
    explicit
GCalcCoordDialog(QWidget *parent =
0);
    ~GCalcCoordDialog();

private slots:
    void on_btn_calcul_clicked();
private:
    Ui::GCalcCoordDialog *ui;
signals:
    void
calc(QDialog*,bool,bool,bool);
};
#endif // GCALCCOORDDIALOG_H
```

gcalccoorddialog.cpp :

```
#include "gcalccoorddialog.h"
#include "ui_gcalccoorddialog.h"
#include <QDebug>
#include <QMessageBox>
GCalcCoordDialog::GCalcCoordDialog
(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::GCalcCoordDialog)
{
    Qt::WindowFlags flags = 0;
            flags = Qt::Dialog;
               flags |=
Qt::WindowTitleHint;
               flags |=
Qt::CustomizeWindowHint;
        setWindowFlags(flags);
        ui->setupUi(this);
        this-
>setAttribute(Qt::WA_DeleteOnClose
);
        connect(this->ui-
>btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
}
GCalcCoordDialog::~GCalcCoordDialo
g()
{
    delete ui;
}
void
GCalcCoordDialog::on_btn_calcul_cl
icked()
{
    bool art = false;
    bool arn = false;
    bool scan = false;
    if (ui->ckb_art-
>isChecked()){art = true;}
    if (ui->ckb_arn-
>isChecked()){arn = true;}
    if (ui->ckb_scan-
>isChecked()){scan = true;}
    if (art == false and arn ==
false and scan == false)
    {
```

```
        QMessageBox::warning(0,
"Calcul Coordonnées", "Vous devez
cocher au moins un champ !!!");
    }
    else
    {
        emit
calc(this,art,arn,scan);
    }
}
```

gconnectiondialog.h :

```
#ifndef GCONNECTIONDIALOG_H
#define GCONNECTIONDIALOG_H
#include <QDialog>
namespace Ui {
class GConnectionDialog;
}
class GConnectionDialog : public
QDialog
{
    Q_OBJECT

public:
    explicit
GConnectionDialog(QWidget *parent,
QString host, QString
defaultUserName, QString
defaultPassword, QString dbName,
int port);
    ~GConnectionDialog();

private:
    Ui::GConnectionDialog *ui;
    QString
_defaultUserName;
    QString
_defaultPassword;
public slots:
    void validate();
    void invite(bool checked);
signals:
    void connectionData(QString
host, QString userName, QString
password, QString dbName, int
port);
    void cancel();
};
#endif // GCONNECTIONDIALOG_H
```

gconnectiondialog.cpp :

```
#include "gconnectiondialog.h"
#include "ui_gconnectiondialog.h"
GConnectionDialog::GConnectionDial
og(QWidget *parent, QString host,
QString defaultUserName, QString
defaultPassword, QString dbName,
int port) :
    QDialog(parent),
    ui(new Ui::GConnectionDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    connect(this->ui->btn_ok,
SIGNAL(clicked()), this,
SLOT(validate()));
    connect(this->ui->btn_annuler,
SIGNAL(clicked()), this,
SIGNAL(cancel()));
    connect(this->ui-
>chb_parametre,
SIGNAL(toggled(bool)), this,
SLOT(invite(bool)));
    ui->lie_hostname-
>setText(host);
    ui->lie_nom->setText(dbName);
    ui->spb_port->setValue(port);
    _defaultUserName =
defaultUserName;
    _defaultPassword =
defaultPassword;
}
GConnectionDialog::~GConnectionDia
log()
{
    delete ui;
}
void GConnectionDialog::validate()
{
    emit connectionData(ui-
>lie_hostname->text(),
                        ui-
>lie_utilisateur->text(),
                        ui-
>lie_passe->text(),
                        ui-
>lie_nom->text(),
                        ui-
>spb_port->value());
    close();
}
void
GConnectionDialog::invite(bool
checked)
{
    if (checked) {
        ui->lie_utilisateur-
>setText(_defaultUserName);
        ui->lie_passe-
>setText(_defaultPassword);
```

```
        ui->lie_utilisateur-
>setEnabled(false);
        ui->lie_passe-
>setEnabled(false);
    } else {
        ui->lie_utilisateur-
>setText("");
        ui->lie_passe-
>setText("");
        ui->lie_utilisateur-
>setEnabled(true);
        ui->lie_passe-
>setEnabled(true);
    }
}
```

gfieldinventorymanagerdialog.h :

```
#ifndef
GFIELDINVENTORYMANAGERDIALOG_H
#define
GFIELDINVENTORYMANAGERDIALOG_H
#include <QDialog>
namespace Ui {
class
GFieldInventoryManagerDialog;
}
class GFieldInventoryManagerDialog
: public QDialog
{
    Q_OBJECT

public:
    explicit
GFieldInventoryManagerDialog(QWidg
et *parent = 0);

~GFieldInventoryManagerDialog();

private slots:
    void
on_toolButton_2_clicked();
    void on_toolButton_clicked();
    void
on_toolButton_3_clicked();
    void
on_toolButton_17_clicked();
    void
on_toolButton_5_clicked();
    void
on_toolButton_4_clicked();
    void
on_toolButton_16_clicked();
    void
on_toolButton_6_clicked();
    void
on_toolButton_7_clicked();
    void
on_toolButton_15_clicked();
    void
on_toolButton_21_clicked();
signals:
    void askForTable(QDialog*,
QString);
private:

Ui::GFieldInventoryManagerDialog
*ui;
};
#endif //
GFIELDINVENTORYMANAGERDIALOG_H
```

gfieldinventorymanagerdialog.cpp :

```
#include
"gfieldinventorymanagerdialog.h"
#include
"ui_gfieldinventorymanagerdialog.h
"
GFieldInventoryManagerDialog::GFie
ldInventoryManagerDialog(QWidget
*parent) :
    QDialog(parent),
    ui(new
Ui::GFieldInventoryManagerDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Dialog;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    connect(this->ui->btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
}
GFieldInventoryManagerDialog::~GFi
eldInventoryManagerDialog()
{
    delete ui;
}
void
GFieldInventoryManagerDialog::on_t
oolButton_2_clicked()
{
    emit askForTable(this, ui-
>toolButton_2->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_clicked()
{
    emit askForTable(this, ui-
>toolButton->text());
}
```

```
void
GFieldInventoryManagerDialog::on_t
oolButton_3_clicked()
{
    emit askForTable(this, ui-
>toolButton_3->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_17_clicked()
{
    emit askForTable(this, ui-
>toolButton_17->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_5_clicked()
{
    emit askForTable(this, ui-
>toolButton_5->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_4_clicked()
{
    emit askForTable(this, ui-
>toolButton_4->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_16_clicked()
{
    emit askForTable(this, ui-
>toolButton_16->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_6_clicked()
{
    emit askForTable(this, ui-
>toolButton_6->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_7_clicked()
{
    emit askForTable(this, ui-
>toolButton_7->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_15_clicked()
{
    emit askForTable(this, ui-
>toolButton_15->text());
}
void
GFieldInventoryManagerDialog::on_t
oolButton_21_clicked()
{
    emit askForTable(this, ui-
>toolButton_21->text());
}
```

gimportdatadialog.h :

```
#ifndef GIMPORTDATADIALOG_H
#define GIMPORTDATADIALOG_H
#include <QDialog>
namespace Ui {
class GImportDataDialog;
}
class GImportDataDialog : public
QDialog
{
    Q_OBJECT

public:
    explicit
GImportDataDialog(QWidget *parent
= 0);
    ~GImportDataDialog();

private slots:
    void on_btn_charge_clicked();
    void on_btn_import_clicked();
    void tableChoice();
private:
    Ui::GImportDataDialog *ui;
    QString _fileName;
    QString _delimiter;
    QString _colonne;
    QString _table;
};
#endif // GIMPORTDATADIALOG_H
```

gimportdatadialog.cpp :

```
#include "gimportdatadialog.h"
#include "ui_gimportdatadialog.h"
#include <QFileDialog>
#include <QDebug>
#include <QtSql>
#include <QMessageBox>
GImportDataDialog::GImportDataDial
og(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::GImportDataDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Dialog;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
```

```
    connect(this->ui->btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
    tableChoice();
}
GImportDataDialog::~GImportDataDia
log()
{
    delete ui;
}
void
GImportDataDialog::on_btn_charge_c
licked()
{
    QString filter;
    _fileName =
QFileDialog::getOpenFileName(this,
tr("Import"),"",tr("Fichier CSV
(*.csv);;Fichier Text
(*.txt)"),&filter);
    ui->lineEdit-
>setText(_fileName);
}
void
GImportDataDialog::on_btn_import_c
licked()
{
    _colonne.clear();
    _table = ui->cmb_table-
>currentText();
    if (ui->rdb_commaPoint-
>isChecked())
    {
        _delimiter = ";";
    }
    else
    {
        _delimiter = "\t";
    }
    QFile file(_fileName);
    if (file.open(QFile::ReadOnly
| QIODevice::Text))
    {
        QTextStream flux(&file);
        _colonne =
flux.readLine();

_colonne.replace(_delimiter,
QString(","));
        _colonne =
"("+_colonne+")";
    }
    QSqlQuery* query = new
QSqlQuery;
    query->exec(QString("COPY %1
%2 FROM '%3' DELIMITER '%4' CSV
HEADER").arg(_table).arg(_colonne)
.arg(_fileName).arg(_delimiter));
    if (query-
>lastError().isValid())
    {

QMessageBox::information(0,
"Import", query-
>lastError().text());
    }
    else
    {

QMessageBox::information(0,
"Import", "Import Réussie");
    }
}
void
GImportDataDialog::tableChoice()
{
    QSqlQueryModel* modelQuery =
new QSqlQueryModel;
    modelQuery->setQuery("SELECT
table_name FROM
INFORMATION_SCHEMA.COLUMNS WHERE
table_schema = 'public' AND
table_name <> 'spatial_ref_sys'
AND table_name <>
'geography_columns' AND table_name
<> 'geometry_columns' AND
table_name <> 'raster_columns' AND
table_name <> 'raster_overviews'
GROUP BY table_name ORDER BY
table_name ASC");
    int count = modelQuery-
>rowCount();
    for (int  i = 0; i < count;
i++)
    {
        ui->cmb_table-
>addItem(modelQuery-
>record(i).value("table_name").toS
tring());
    }
}
```

gmainwindow.h :

```
#ifndef GMAINWINDOW_H
#define GMAINWINDOW_H
#include <QMainWindow>
namespace Ui {
class GMainWindow;
}
class GMainWindow : public
QMainWindow
{
    Q_OBJECT

public:
    explicit GMainWindow(QWidget
*parent = 0);
    ~GMainWindow();

private:
    Ui::GMainWindow *ui;
```

```
signals:
    void needConnection();
    void openPlotManager();
    void
openFieldInventoryManager();
    void
openNumericalInventoryManager();
    void
openOthersTablesManager();
    void openImportDataDialog();
    void openCalcCoordDialog();
    void openQueryDialog();
    void closeMainWindows();
};
#endif // GMAINWINDOW_H
```

gmainwindow.cpp :

```
#include "gmainwindow.h"
#include "ui_gmainwindow.h"
GMainWindow::GMainWindow(QWidget
*parent) :
    QMainWindow(parent),
    ui(new Ui::GMainWindow)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    connect(this->ui->btn_quitter,
SIGNAL(clicked()), this,
SIGNAL(closeMainWindows()));
    connect(this->ui-
>btn_placette, SIGNAL(clicked()),
this, SIGNAL(openPlotManager()));
    connect(this->ui->btn_int,
SIGNAL(clicked()), this,
SIGNAL(openFieldInventoryManager()
));
    connect(this->ui->btn_inn,
SIGNAL(clicked()), this,
SIGNAL(openNumericalInventoryManag
er()));
    connect(this->ui->btn_cco,
SIGNAL(clicked()), this,
SIGNAL(openCalcCoordDialog()));
    connect(this->ui->btn_import,
SIGNAL(clicked()), this,
SIGNAL(openImportDataDialog()));
    connect(this->ui->btn_taa,
SIGNAL(clicked()), this,
SIGNAL(openOthersTablesManager()))
;
    connect(this->ui->btn_requete,
SIGNAL(clicked()), this,
SIGNAL(openQueryDialog()));
}

GMainWindow::~GMainWindow()
{
    delete ui;
}
```

gnumericalinventorymanagerdialog.h :

```
#ifndef
GNUMERICALINVENTORYMANAGERDIALOG_H
#define
GNUMERICALINVENTORYMANAGERDIALOG_H
#include <QDialog>
namespace Ui {
class
GNumericalInventoryManagerDialog;
}
class
GNumericalInventoryManagerDialog :
public QDialog
{
    Q_OBJECT

public:
    explicit
GNumericalInventoryManagerDialog(Q
Widget *parent = 0);

~GNumericalInventoryManagerDialog(
);

private slots:
    void
on_toolButton_8_clicked();
    void
on_toolButton_20_clicked();
    void
on_toolButton_21_clicked();
    void
on_toolButton_7_clicked();
    void
on_toolButton_22_clicked();
    void
on_toolButton_19_clicked();
    void on_toolButton_clicked();
    void
on_toolButton_2_clicked();
    void
on_toolButton_6_clicked();
    void
on_toolButton_9_clicked();
    void
on_toolButton_24_clicked();
    void
on_toolButton_23_clicked();
    void
on_toolButton_5_clicked();
    void
on_toolButton_3_clicked();
    void
on_toolButton_4_clicked();
signals:
```

```
    void askForTable(QDialog*,
QString);
private:

Ui::GNumericalInventoryManagerDial
og *ui;
};
#endif //
GNUMERICALINVENTORYMANAGERDIALOG_H
```

gnumericalinventorymanagerdialog.cpp :

```
#include
"gnumericalinventorymanagerdialog.
h"
#include
"ui_gnumericalinventorymanagerdial
og.h"
GNumericalInventoryManagerDialog::
GNumericalInventoryManagerDialog(Q
Widget *parent) :
    QDialog(parent),
    ui(new
Ui::GNumericalInventoryManagerDial
og)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Dialog;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    connect(this->ui->btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
}
GNumericalInventoryManagerDialog::
~GNumericalInventoryManagerDialog(
)
{
    delete ui;
}
void
GNumericalInventoryManagerDialog::
on_toolButton_8_clicked()
{
    emit askForTable(this, ui-
>toolButton_8->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_20_clicked()
{
    emit askForTable(this, ui-
>toolButton_20->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_21_clicked()
{
    emit askForTable(this, ui-
>toolButton_21->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_7_clicked()
{
    emit askForTable(this, ui-
>toolButton_7->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_22_clicked()
{
    emit askForTable(this, ui-
>toolButton_22->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_19_clicked()
{
    emit askForTable(this, ui-
>toolButton_19->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_clicked()
{
    emit askForTable(this, ui-
>toolButton->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_2_clicked()
{
    emit askForTable(this, ui-
>toolButton_2->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_6_clicked()
{
    emit askForTable(this, ui-
>toolButton_6->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_9_clicked()
{
    emit askForTable(this, ui-
>toolButton_9->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_24_clicked()
{
```

```
    emit askForTable(this, ui-
>toolButton_24->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_23_clicked()
{
    emit askForTable(this, ui-
>toolButton_23->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_5_clicked()
{
    emit askForTable(this, ui-
>toolButton_5->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_3_clicked()
{
    emit askForTable(this, ui-
>toolButton_3->text());
}
void
GNumericalInventoryManagerDialog::
on_toolButton_4_clicked()
{
    emit askForTable(this, ui-
>toolButton_4->text());
}
```

gotherstablesmanagerdialog.h :

```
#ifndef
GOTHERSTABLESMANAGERDIALOG_H
#define
GOTHERSTABLESMANAGERDIALOG_H
#include <QDialog>
namespace Ui {
class GOthersTablesManagerDialog;
}
class GOthersTablesManagerDialog :
public QDialog
{
    Q_OBJECT

public:
    explicit
GOthersTablesManagerDialog(QWidget
*parent = 0);
    ~GOthersTablesManagerDialog();

private:
    Ui::GOthersTablesManagerDialog
*ui;
signals:
    void askForTable(QDialog*,
QString);
private slots:
    void on_toolButton_clicked();
    void
on_toolButton_3_clicked();
    void
on_toolButton_2_clicked();
};
#endif //
GOTHERSTABLESMANAGERDIALOG_H
```

gotherstablesmanagerdialog.cpp :

```
#include
"gotherstablesmanagerdialog.h"
#include
"ui_gotherstablesmanagerdialog.h"
GOthersTablesManagerDialog::GOther
sTablesManagerDialog(QWidget
*parent) :
    QDialog(parent),
    ui(new
Ui::GOthersTablesManagerDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Dialog;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    connect(this->ui->btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
}
GOthersTablesManagerDialog::~GOthe
rsTablesManagerDialog()
{
    delete ui;
}
void
GOthersTablesManagerDialog::on_too
lButton_clicked()
{
    emit askForTable(this, ui-
>toolButton->text());
}
void
GOthersTablesManagerDialog::on_too
lButton_3_clicked()
{
    emit askForTable(this, ui-
>toolButton_3->text());
}
void
GOthersTablesManagerDialog::on_too
lButton_2_clicked()
{
    emit askForTable(this, ui-
>toolButton_2->text());
```

```
}
```

gplotmanagementdialog.h :

```
#ifndef GPLOTMANAGEMENTDIALOG_H
#define GPLOTMANAGEMENTDIALOG_H
#include <QDialog>
namespace Ui {
class GPlotManagementDialog;
}
class GPlotManagementDialog :
public QDialog
{
    Q_OBJECT

public:
    explicit
GPlotManagementDialog(QWidget
*parent = 0);
    ~GPlotManagementDialog();

private:
    Ui::GPlotManagementDialog *ui;
signals:
    void askForTable(QDialog*,
QString);
private slots:
    void
on_btn_placette_clicked();
    void
on_btn_localisation_clicked();
    void
on_btn_orientation_clicked();
    void on_btn_site_clicked();
    void on_btn_reseau_clicked();
};
#endif // GPLOTMANAGEMENTDIALOG_H
```

gplotmanagementdialog.cpp :

```
#include "gplotmanagementdialog.h"
#include
"ui_gplotmanagementdialog.h"
GPlotManagementDialog::GPlotManage
mentDialog(QWidget *parent) :
    QDialog(parent),
    ui(new
Ui::GPlotManagementDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Dialog;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    connect(this->ui->btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
}
GPlotManagementDialog::~GPlotManag
ementDialog()
{
    delete ui;
}
void
GPlotManagementDialog::on_btn_plac
ette_clicked()
{
    emit askForTable(this, ui-
>btn_placette->text());
}
void
GPlotManagementDialog::on_btn_loca
lisation_clicked()
{
    emit askForTable(this, ui-
>btn_localisation->text());
}
void
GPlotManagementDialog::on_btn_orie
ntation_clicked()
{
    emit askForTable(this, ui-
>btn_orientation->text());
}
void
GPlotManagementDialog::on_btn_site
_clicked()
{
    emit askForTable(this, ui-
>btn_site->text());
}
void
GPlotManagementDialog::on_btn_rese
au_clicked()
{
    emit askForTable(this, ui-
>btn_reseau->text());
}
```

gquerydialog.h :

```
#ifndef GQUERYDIALOG_H
#define GQUERYDIALOG_H
#include <QDialog>
#include <QtSql>
namespace Ui {
class GQueryDialog;
}
class GQueryDialog : public
QDialog
{
    Q_OBJECT

public:
```

```
    explicit GQueryDialog(QWidget
*parent = 0);
    ~GQueryDialog();
    QString _dossier;

private slots:
    void
on_btn_soumettre_clicked();
    void on_btn_valider_clicked();
    void on_btn_dossier_clicked();
    void comboBox();
    void on_btn_save_clicked();
private:
    Ui::GQueryDialog *ui;
    QString          _requete;
signals:
    void
submit(QDialog*,QSqlQueryModel*);
};
#endif // GQUERYDIALOG_H
```

gquerydialog.cpp :

```
#include "gquerydialog.h"
#include "ui_gquerydialog.h"
#include <QDebug>
#include <QtSql>
#include <QMessageBox>
#include <QFileDialog>
#include <QFileSystemModel>
#include <QInputDialog>
GQueryDialog::GQueryDialog(QWidget
*parent) :
    QDialog(parent),
    ui(new Ui::GQueryDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Dialog;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    _dossier =
"D:/donnees/Maillet_cedric/c++/Qt/
BdTLidar_Client/requete";
    comboBox();
    connect(this->ui->btn_retour,
SIGNAL(clicked()),this,
SLOT(close()));
}
GQueryDialog::~GQueryDialog()
{
    delete ui;
}
```

```
void
GQueryDialog::on_btn_soumettre_cli
cked()
{
    _requete = ui->txt_requete-
>toPlainText();
    QSqlQueryModel* modelQuery =
new QSqlQueryModel;
    modelQuery-
>setQuery(_requete);
    if(modelQuery->rowCount()==0)
    {
        QString error =
modelQuery->lastError().text();

QMessageBox::information(0,
"Requête", modelQuery-
>lastError().text());
    }
    else
    {
        emit submit(this,
modelQuery);
    }
}
void
GQueryDialog::on_btn_valider_click
ed()
{
    QString fileName =
_dossier+"/"+ui->cmb_requete-
>currentText()+".sql";
    QFile file(fileName);
    if (file.open(QFile::ReadOnly
| QIODevice::Text))
    {
        _requete = file.readAll();
    }
    ui->txt_requete-
>setPlainText(_requete);
}
void
GQueryDialog::on_btn_dossier_click
ed()
{
    QString dir =
QFileDialog::getExistingDirectory(
this, tr("Choix du Dossier"),
_dossier,QFileDialog::ShowDirsOnly
|
QFileDialog::DontResolveSymlinks);
    _dossier = dir;
    comboBox();
}
void GQueryDialog::comboBox()
{
    ui->cmb_requete->clear();
    QDir dir(_dossier);
    QFileInfoList list =
dir.entryInfoList();
    int count = list.count();
```

```
    for (int  i = 0; i < count;
i++)
    {
        QFileInfo fileInfo =
list.at(i);

if(fileInfo.completeSuffix() ==
"sql")
        {
            ui->cmb_requete-
>addItem(fileInfo.baseName());
        }
    }
}
void
GQueryDialog::on_btn_save_clicked(
)
{
    if(ui->txt_requete-
>toPlainText()=="")
    {

QMessageBox::information(0,
"Attention", "Aucune requête n'a
été saisie !");
    }
    else
    {
        QString nameFile =
_dossier + "/"
+QInputDialog::getText(this, "Nom
du fichier", "Saisir un nom pour
le fichier : ")+".sql";
        QFile file(nameFile);
        if
(!file.open(QIODevice::WriteOnly |
QIODevice::Text))
            return;
        QTextStream out(&file);
         _requete = ui-
>txt_requete->toPlainText();
         out << _requete;
         comboBox();
    }
}
```

gtabledialog.h :

```
#ifndef GTABLEDIALOG_H
#define GTABLEDIALOG_H
#include <QDialog>
#include <QtSql>
#include "qdatawidgetmapper.h"
namespace Ui {
class GTableDialog;
}
class GTableDialog : public
QDialog
{
    Q_OBJECT
public:
    explicit GTableDialog(QWidget
*parent, QString title,
QSqlRelationalTableModel *model,
QMap<QString, QString>
&descriptionFields);
    explicit GTableDialog(QWidget
*parent, QString title,
QSqlQueryModel *model,
QMap<QString, QString>
&descriptionFields);
    ~GTableDialog();

private:
    Ui::GTableDialog
*ui;
    bool
_editMode;
    QDataWidgetMapper*
_mapper;
    QSqlRelationalTableModel*
_model;
    QSqlQueryModel*
_modelQuery;
    QMap<QString, QString>*
_descriptionFields;
    bool
_tabMode;
    bool
_queryModel;
public slots:
    void editMode();
    void deleteLine();
    void newMode();
    void mapper();
    void mapperQuery();
private slots:
    void
on_pushButton_2_clicked();
    void
on_pushButton_3_clicked();
    void on_toolButton_clicked();
    void
on_toolButton_2_clicked();
    void
on_toolButton_3_clicked();
    void
on_tabWidget_currentChanged(int
index);
    void on_btn_export_clicked();
signals:
    void
exportMode(QSqlRelationalTableMode
l*,QString,QString);
    void
exportModeQuery(QSqlQueryModel*,QS
tring,QString);
};
#endif // GTABLEDIALOG_H
```

gtabledialog.cpp :

```
#include "gtabledialog.h"
#include "ui_gtabledialog.h"
#include "qmessagebox.h"
#include "qdebug.h"
#include <QtGui>
#include <limits>
GTableDialog::GTableDialog(QWidget
*parent, QString title,
QSqlRelationalTableModel* model,
QMap<QString, QString>
&descriptionFields) :
    QDialog(parent),
    ui(new Ui::GTableDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags
|=Qt::WindowMaximizeButtonHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    ui->lbl_titre->setText(title);
    ui->tableview-
>setModel(model);
    ui->tableview-
>setItemDelegate(new
QSqlRelationalDelegate(ui-
>tableview));
    _descriptionFields =
&descriptionFields;
    _editMode = false;
    _tabMode = true;
    _queryModel = false;
    _model = model;
    ui->btn_supprimer-
>setEnabled(false);
    ui->btn_nouveau-
>setEnabled(false);
    ui->btn_modifier-
>setText("Mode lecture seule");
    ui->pushButton_2-
>setEnabled(false);
    connect(ui->btn_retour,
SIGNAL(clicked()), this,
SLOT(close()));
    connect(ui->btn_modifier,
SIGNAL(clicked()), this,
SLOT(editMode()));
    connect(ui->btn_supprimer,
SIGNAL(clicked()), this,
SLOT(deleteLine()));
    connect(ui->btn_nouveau,
SIGNAL(clicked()), this,
SLOT(newMode()));
    mapper();
    ui->tabWidget-
>setCurrentIndex(0);
    ui->widget_5-
>setEnabled(false);
}
GTableDialog::GTableDialog(QWidget
*parent, QString title,
QSqlQueryModel *model,
QMap<QString, QString>
&descriptionFields):
    QDialog(parent),
    ui(new Ui::GTableDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags
|=Qt::WindowMaximizeButtonHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);
    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);
    ui->lbl_titre->setText(title);
    ui->tableview-
>setModel(model);
    _editMode = false;
    _tabMode = true;
    _queryModel = true;
    _modelQuery = model;
    ui->btn_supprimer-
>setEnabled(false);
    ui->btn_nouveau-
>setEnabled(false);
    ui->btn_modifier-
>setText("Mode lecture
uniquement");
    ui->btn_modifier-
>setEnabled(false);
    ui->pushButton_2-
>setEnabled(false);
    connect(ui->btn_retour,
SIGNAL(clicked()), this,
SLOT(close()));
    mapperQuery();
    ui->tabWidget-
>setCurrentIndex(0);
    ui->widget_5-
>setEnabled(false);
}
GTableDialog::~GTableDialog()
{
    delete ui;
    if(_queryModel)
```

```
    {
        delete _modelQuery;
    }
    else
    {
        delete _model;
    }
}
void GTableDialog::mapper()
{
    _mapper = new
QDataWidgetMapper(this);
    _mapper-
>setSubmitPolicy(QDataWidgetMapper
::AutoSubmit);
    _mapper->setModel(_model);
    _mapper->setItemDelegate(new
QSqlRelationalDelegate(this));
    int count = _model-
>columnCount();
    for (int  i = 0; i < count;
i++)
    {
        QModelIndex index =
_model->index(0,i);
        QVariant val = _model-
>data(index);
        QWidget * widget;
        QVariant header = _model-
>headerData(i,Qt::Horizontal);
        QWidget* widget2;
        widget2 = new QLabel();
        ((QLabel*) widget2)-
>setText(header.toString()+" : ");
        _mapper-
>addMapping(widget2,i);
        ui->widget_7->layout()-
>addWidget(widget2);
        switch (val.type())
        {
            case QVariant::Int: {
                widget = new
QSpinBox();
                ((QSpinBox*)
widget)-
>setMinimum(std::numeric_limits<in
t>::min());
                ((QSpinBox*)
widget)-
>setMaximum(std::numeric_limits<in
t>::max());
                ((QSpinBox*)
widget)->setValue(val.toInt());
                break;
            }
            case QVariant::Double:
{
                widget = new
QDoubleSpinBox();
                ((QDoubleSpinBox*)
widget)-
>setMinimum(std::numeric_limits<do
uble>::min());
                ((QDoubleSpinBox*)
widget)-
>setMaximum(std::numeric_limits<do
uble>::max());
                ((QDoubleSpinBox*)
widget)->setValue(val.toDouble());
                break;
            }
            case QVariant::String:
{
                if (_model-
>relation(i).isValid()) {
                    QSqlTableModel
*relModel =
((QSqlRelationalTableModel*)
_model)->relationModel(i);
                    widget = new
QComboBox();
                    ((QComboBox*)
widget)->setModel(relModel);
                    ((QComboBox*)
widget)->setModelColumn(relModel-
>fieldIndex(_descriptionFields-
>value(relModel->tableName())));
                } else {
                    widget = new
QLineEdit();
                    ((QLineEdit*)
widget)->setText(val.toString());
                }
                break;
            }
            case QVariant::Bool: {
                widget = new
QLineEdit();
                ((QLineEdit*)
widget)->setText(val.toString());
                break;
            }
            case QVariant::Date: {
                widget = new
QDateEdit;
                ((QDateEdit*)
widget)->setDate(val.toDate());
                break;
            }
        }
        _mapper-
>addMapping(widget, i);
        ui->widget_6->layout()-
>addWidget(widget);
        _mapper->toFirst();
    }
}
void GTableDialog::mapperQuery()
{
    _mapper = new
QDataWidgetMapper(this);
```

```
    _mapper->setSubmitPolicy(QDataWidgetMapper::AutoSubmit);
    _mapper->setModel(_modelQuery);
    int count = _modelQuery->columnCount();
    for (int  i = 0; i < count; i++)
    {
        QModelIndex index = _modelQuery->index(0,i);
        QVariant val = _modelQuery->data(index);
        QWidget * widget;
        QVariant header = _modelQuery->headerData(i,Qt::Horizontal);
        QWidget* widget2;
        widget2 = new QLabel();
        ((QLabel*) widget2)->setText(header.toString()+" : ");
        _mapper->addMapping(widget2,i);
        ui->widget_7->layout()->addWidget(widget2);
        switch (val.type())
        {
            case QVariant::Int: {
                widget = new QSpinBox();
                ((QSpinBox*) widget)->setMinimum(std::numeric_limits<int>::min());
                ((QSpinBox*) widget)->setMaximum(std::numeric_limits<int>::max());
                ((QSpinBox*) widget)->setValue(val.toInt());
                break;
            }
            case QVariant::Double: {
                widget = new QDoubleSpinBox();
                ((QDoubleSpinBox*) widget)->setMinimum(std::numeric_limits<double>::min());
                ((QDoubleSpinBox*) widget)->setMaximum(std::numeric_limits<double>::max());
                ((QDoubleSpinBox*) widget)->setValue(val.toDouble());
                break;
            }
            case QVariant::String: {
                widget = new QLineEdit();
                ((QLineEdit*) widget)->setText(val.toString());
                break;
            }
        }
        _mapper->addMapping(widget, i);
        ui->widget_6->layout()->addWidget(widget);
        _mapper->toFirst();
    }
}
void GTableDialog::editMode()
{
    if (_editMode)
    {
        _editMode = false;
        ui->tableview->setEditTriggers(QAbstractItemView::NoEditTriggers);
        ui->btn_supprimer->setEnabled(false);
        ui->btn_nouveau->setEnabled(false);
        ui->btn_modifier->setText("Mode lecture seule");
        ui->widget_5->setEnabled(false);
        ui->pushButton_2->setEnabled(false);
    }
    else
    {
       _editMode = true;
       ui->tableview->setEditTriggers(QAbstractItemView::DoubleClicked);
       ui->btn_supprimer->setEnabled(true);
       ui->btn_nouveau->setEnabled(true);
       ui->btn_modifier->setText("Mode modification");
       ui->widget_5->setEnabled(true);
       ui->pushButton_2->setEnabled(true);
    }
}
void GTableDialog::deleteLine()
{
    if (_editMode)
    {
        int reponse = QMessageBox::question(this, "Suppression", "Etes vous sur de vouloir supprimer ?", QMessageBox::Yes | QMessageBox::No);
```

```
        if (reponse ==
QMessageBox::Yes)
        {
            if (_tabMode)
            {
                QItemSelection
selection( ui->tableview-
>selectionModel()->selection() );
                QList<int> rows;
                foreach( const
QModelIndex & index,
selection.indexes() )
                {
                    rows.append(
index.row() );
                }
                qSort( rows );
                int prev = -1;
                for( int i =
rows.count() - 1; i >= 0; i -= 1 )
                {
                    int current =
rows[i];
                    if( current !=
prev )
                    {
                        _model-
>removeRows( current, 1 );
                        prev =
current;
                    }
                }
            }
            else
            {
                int row = _mapper-
>currentIndex();
                _model-
>removeRow(_mapper-
>currentIndex());
                _model-
>submitAll();
                _mapper-
>setCurrentIndex(row - 1);
            }
        }
        else if (reponse ==
QMessageBox::No)
        {
            qDebug()
<<"supprimer2" ;
        }
    }
}
void GTableDialog::newMode()
{
    _model->insertRow(_model-
>rowCount());
    _mapper->toLast();
    int max = ui->tableview-
>verticalScrollBar()->maximum();
    ui->tableview-
>verticalScrollBar()-
>setValue(max);
}
void
GTableDialog::on_pushButton_2_clic
ked()
{
    if (_tabMode)
    {
        _model->submitAll();
        _mapper->toFirst();
    }
    else
    {
        int row = _mapper-
>currentIndex();
        _model->submitAll();
        ui->tableview-
>clearSelection();
        ui->tableview-
>selectRow(row);
        _mapper-
>setCurrentIndex(row);
    }
}
void
GTableDialog::on_pushButton_3_clic
ked()
{
    _mapper->toNext();
}
void
GTableDialog::on_toolButton_clicke
d()
{
    _mapper->toPrevious();
}
void
GTableDialog::on_toolButton_2_clic
ked()
{
    _mapper->toFirst();
}
void
GTableDialog::on_toolButton_3_clic
ked()
{
    _mapper->toLast();
}
void
GTableDialog::on_tabWidget_current
Changed(int index)
{
    if(index==0)
    {
        if (_queryModel)
        {
            _tabMode = true;
            int row2 = _mapper-
>currentIndex();
```

```
            ui->tableview->clearSelection();
            ui->tableview->selectRow(row2);
        }
        else
        {
            _tabMode = true;
            int row2 = _mapper->currentIndex();


        if (_queryModel)
        {
            _tabMode = false;
            int row = ui->tableview->currentIndex().row();
            _mapper->setCurrentIndex(row);
        }
        else
        {
            _tabMode = false;
            _model->setEditStrategy(QSqlTableModel::OnManualSubmit);
            int row = ui->tableview->currentIndex().row();
            _mapper->setCurrentIndex(row);
        }
    }
}
void GTableDialog::on_btn_export_clicked()
{
    QString filter;
    QString fileName = QFileDialog::getSaveFileName(this, tr("Export"),"",tr("Fichier CSV (séparateur ;) (*.csv);;Fichier Text (*.txt);;Fichier CSV (séparateur Tabulation) (*.csv)"),&filter);
    if (_queryModel)
    {
        QSqlQueryModel* modelQuery = _modelQuery;
        emit exportModeQuery(modelQuery,filter,fileName);
    }
    else
    {
        QSqlRelationalTableModel* model = _model;
        emit exportMode(model,filter,fileName);
    }
}
```

```
            _model->setEditStrategy(QSqlTableModel::OnFieldChange);
            ui->tableview->clearSelection();
            ui->tableview->selectRow(row2);
        }
    }
    else if(index==1)
    {
```

Annexe 5 : Interface SIG

main.cpp :

```
#include <QtGui/QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    return a.exec();
}
```

gmenu.h :

```
#include "qgisplugin.h"
#include "qgisinterface.h"
#include "qgsdatasourceuri.h"
#include "qgsvectorlayer.h"
#include "qgsmaplayerregistry.h"

#include <QObject>

class QAction;

class GMenu : public QObject,
public QgisPlugin
{
    Q_OBJECT

public:
    GMenu(QgisInterface* iface);
    ~GMenu();
    void initGui();
    void unload();

private:
    QgisInterface*
_iface;
    QAction*
_actionAffichage;
    QAction*
_actionPage;
    QAction*
_actionEmprise;
    QAction*
_actionRepresentation;
    QAction*
_actionConnection;
    GConnectionDialog*      _d;
    QWidget*                _w;
    GAffichageDialog*       _a;
    GEmpriseDialog*         _b;
    GRepresentationDialog*  _c;
    DataBaseConnection*     _dbc;
    QString                 _host;
    QString
_userName;
    QString
_password;
    QString
_dbName;
    QString                 _port;
    bool
_openA;
    bool
_openB;
    bool
_openC;
    QgsDataSourceURI*       _uri;
    QgsVectorLayer*         _vl;
    QgsMapLayerRegistry*    _mlr;

private slots:
    void affichage();
    void page();
    void emprise();
    void representation();
    void connection();
    void createConnection(QString
host, QString userName, QString
password, QString dbName, int
port);
    void closeConnectionDialog();
    void affDonnee(bool
placette,bool scan,bool
arbreNumerique,bool
arbreTerrain,QString foret,QString
parcelle,QString placette2);

};

#endif // GMENU_H
```

gmenu.cpp :

```
#include "gmenu.h"
#include "gaffichagedialog.h"
#include "gemprisedialog.h"
#include "grepresentationdialog.h"

#include "qgsdatasourceuri.h"
#include "qgisplugin.h"
#include "qgsmaplayerregistry.h"
#include "qgsmaprenderer.h"
#include "qgscomposition.h"
#include "qgsmapcanvas.h"
#include "qgscomposermap.h"
```

```
#include "qmessagebox.h"
#include <QAction>
#include "QtSql"
#include "qdebug.h"

#ifdef WIN32
#define QGISEXTERN extern "C"
__declspec( dllexport )
#else
#define QGISEXTERN extern "C"
#endif

GMenu::GMenu(QgisInterface* iface)
: _iface(iface),
_actionAffichage(0),
_actionPage(0), _actionEmprise(0),
_actionRepresentation(0),
_actionConnection(0)
{
    _openA = false;
    _openB = false;
    _openC = false;
    _dbc = NULL;
}

GMenu::~GMenu()
{
    if (_dbc != NULL) {_dbc-
>disconnect();}
}

void GMenu::initGui()
{
    _actionAffichage = new
QAction(tr("&Affichage"),this);
    _actionPage = new
QAction(tr("&Mise en Page"),this);
    _actionEmprise = new
QAction(tr("&Emprise"),this);
    _actionRepresentation = new
QAction(tr("&Representation"),this
);
    _actionConnection = new
QAction(tr("&Connection"),this);

    _actionAffichage-
>setWhatsThis( tr( "Permet
d'afficher les données
géographique de la table BdTLidar"
) );
    _actionPage-
>setWhatsThis(tr("Mise en page
automatique des données afficher"
) );
    _actionEmprise->setWhatsThis(
tr( "Calcul des emprise temporaire
pour chaque placette afficher." )
);
    _actionRepresentation-
>setWhatsThis(tr( "Permet de
calculer une representation de la
circonference ou du diametre de
chaque arbre." ) );
    _actionConnection-
>setWhatsThis(tr("Permet de
modifier les paramètresn de
connection." ) );

    connect(_actionAffichage,
SIGNAL(activated()), this,
SLOT(affichage()));
    connect(_actionPage,
SIGNAL(activated()), this,
SLOT(page()));
    connect(_actionEmprise,
SIGNAL(activated()), this,
SLOT(emprise()));
    connect(_actionRepresentation,
SIGNAL(activated()), this,
SLOT(representation()));
    connect(_actionConnection,
SIGNAL(activated()),this,SLOT(conn
ection()));

    _iface-
>addPluginToMenu("BdTLidar
plugin", _actionAffichage);
    _iface-
>addPluginToMenu("BdTLidar
plugin", _actionPage);
    _iface-
>addPluginToMenu("BdTLidar
plugin", _actionEmprise);
    _iface-
>addPluginToMenu("BdTLidar
plugin", _actionRepresentation);
    _iface-
>addPluginToMenu("BdTLidar
plugin", _actionConnection);
}
void GMenu::unload()
{
    _iface-
>removePluginMenu("BdTLidar
plugin", _actionAffichage);
```

```
    _iface->removePluginMenu("BdTLidar plugin", _actionPage);
    _iface->removePluginMenu("BdTLidar plugin", _actionEmprise);
    _iface->removePluginMenu("BdTLidar plugin", _actionRepresentation);
    _iface->removePluginMenu("BdTLidar plugin", _actionConnection);

    delete _actionAffichage;
    delete _actionPage;
    delete _actionEmprise;
    delete _actionRepresentation;
    delete _actionConnection;
}

void GMenu::affichage()
{
    _a = new GAffichageDialog;

connect(_a,SIGNAL(affichage(bool,bool,bool,bool,QString,QString,QString)),this,SLOT(affDonnee(bool,bool,bool,bool,QString,QString,QString)));
    _a->move(800,400);
    _a->show();

    _openA = true;

    connection();
}
void GMenu::page()
{
    connection();

    _iface->actionPrintComposer()->trigger();
    QList<QgsComposerView*> composerList;
    composerList = _iface->activeComposers();

    QgsMapRenderer* mapRenderer = new QgsMapRenderer();
    QgsMapCanvas* mapCanvas = new QgsMapCanvas();
    mapRenderer = _iface->mapCanvas()->mapRenderer();
    QgsComposition* compo = new QgsComposition(mapRenderer);
    compo->setPlotStyle(QgsComposition::Print);

    double x = 0;
    double y = 0;
    double w = compo->paperWidth();
    double h = compo->paperHeight();
    QgsComposerMap* composerMap = new QgsComposerMap(compo, x, y, w, h);
    compo->addItem(composerMap);
}

void GMenu::emprise()
{
    _b = new GEmpriseDialog;
    _b->move(800,400);
    _b->show();

    _openB = true;

    connection();
}

void GMenu::representation()
{
    _c = new GRepresentationDialog;
    _c->move(800,450);
    _c->show();

    _openC = true;

    connection();
}

void GMenu::connection()
{
    if (_dbc == NULL)
    {
        _w = new QWidget;
        _d = new GConnectionDialog(_w,"localhost","postgres","Lidar159","Lidar_test",5432);
        connect(_d, SIGNAL(connectionData(QString,QString,QString,QString,int)), this,
```

```
SLOT(createConnection(QString,QStr
ing,QString,QString,int)));
        connect(_d,
SIGNAL(cancel()), this,
SLOT(closeConnectionDialog()));
        _d->setModal(true);
        _d->move(850,400);
        _d->show();
    }
    else
    {
        _a->remplirComboBox();
    }
}

void
GMenu::createConnection(QString
host, QString userName, QString
password, QString dbName, int
port)
{
    _host = host;
    _userName = userName;
    _password = password;
    _dbName = dbName;
    _port =
QString("%1").arg(port);

     if (_dbc != NULL) {delete
_dbc;}

     _dbc = new
DataBaseConnection(_host,
_userName, _password, _dbName,
port, "QPSQL");
     bool ok = _dbc->connect();

     if (ok)
     {

QMessageBox::information(0,
"Connection", "Connection reussie
!");

         if(_openA)
         {
             _a-
>remplirComboBox();
         }
     }
     else
     {
         QMessageBox::warning(0,
"Connection", "Parametre de
connexion invalide !");
         connection();
     }
}

void
GMenu::closeConnectionDialog()
{
    _d->reject();

    if (_openA)
    {
        _a->reject();
        _openA = false;
    }
    else if (_openB)
    {
        _b->reject();
        _openB = false;
    }
    else if (_openC)
    {
        _c->reject();
        _openC = false;
    }
}

void GMenu::affDonnee(bool
placette, bool scan, bool
arbreNumerique, bool arbreTerrain,
QString foret, QString parcelle,
QString placette2)
{
    QString where = "" ;

    if(foret != "")
    {
        if(parcelle != "")
        {
            if(placette2 != "")
            {
                where =
QString("AND (placette.pla_id =
'%1') ").arg(placette2);
            }
            else
            {
                where =
QString("AND
(localisation.loc_nom_foret =
'%1') AND
```

```
(localisation.loc_num_parcelle =
'%2') ").arg(foret).arg(parcelle);
            }
        }
        else
        {
            where = QString("AND
(localisation.loc_nom_foret =
'%1') ").arg(foret);
        }
    }

    if (arbreTerrain == true)
    {
        QSqlQueryModel*
modelTerrain = new QSqlQueryModel;
        modelTerrain-
>setQuery(QString("SELECT arb_id
FROM arbre_terrain, placette,
localisation WHERE
(arbre_terrain.pla_id =
placette.pla_id) AND
(placette.loc_id =
localisation.loc_id) %1 ORDER BY
arb_id ASC").arg(where));

        int countTerrain =
modelTerrain->rowCount();
        QString requeteTerrain;

        for (int  j = 0; j <
countTerrain; j++)
        {
            if (j == 0)
            {
                QString idTerrain
= modelTerrain-
>record(j).value("arb_id").toStrin
g();
                requeteTerrain =
"(arb_id = " + idTerrain +") ";
            }
            else
            {
                QString idTerrain
= modelTerrain-
>record(j).value("arb_id").toStrin
g();
                requeteTerrain =
requeteTerrain + "OR (arb_id = " +
idTerrain + ") ";
            }
        }
        _uri = new
QgsDataSourceURI();
        _uri-
>setConnection(_host,_port,_dbName
,_userName,_password);
        _uri-
>setDataSource("public","arbre_ter
rain","the_geom");
        _uri-
>setSql(requeteTerrain);
        _vl = new
QgsVectorLayer(_uri-
>uri(),"arbre_terrain",
"postgres");
        QString way =
"/home/administrateur/cedric/BdTLi
dar_Qgis/arbre.qml";
        bool flag = false;
        _vl-
>loadNamedStyle(way,flag);

QgsMapLayerRegistry::instance()-
>addMapLayer(_vl, true);
        _iface-
>refreshLegend(_vl);
    }

    if(placette == true)
    {
        QSqlQueryModel*
modelPlacette = new
QSqlQueryModel();
        modelPlacette-
>setQuery(QString("SELECT pla_id
FROM placette, localisation WHERE
(placette.loc_id =
localisation.loc_id) %1 ORDER BY
pla_id ASC").arg(where));

        int countPlacette =
modelPlacette->rowCount();
        QString requetePlacette;

        for (int  i = 0; i <
countPlacette; i++)
        {
            if (i == 0)
            {
                QString idPlacette
= modelPlacette-
>record(i).value("pla_id").toStrin
g();
```

```
                requetePlacette =
"(pla_id = " + idPlacette + ") ";
            }
            else
            {
                QString idPlacette
= modelPlacette-
>record(i).value("pla_id").toStrin
g();
                requetePlacette =
requetePlacette + "OR (pla_id = "
+ idPlacette + ") ";
            }
        }

        _uri = new
QgsDataSourceURI();
        _uri-
>setConnection(_host,_port,_dbName
,_userName,_password);
        _uri-
>setDataSource("public","placette"
,"the_geom");
        _uri-
>setSql(requetePlacette);
        _vl = new
QgsVectorLayer(_uri-
>uri(),"placette", "postgres");
        QString way =
"/home/administrateur/cedric/BdTLi
dar_Qgis/placette.qml";
        bool flag = false;
        _vl-
>loadNamedStyle(way,flag);

QgsMapLayerRegistry::instance()-
>addMapLayer(_vl, true);
        _vl-
>toggleScaleBasedVisibility(flag);
        _iface-
>refreshLegend(_vl);

    }

    if (arbreNumerique == true)
    {
        QSqlQueryModel* modelNumer
= new QSqlQueryModel;
        modelNumer-
>setQuery(QString("SELECT arn_id
FROM arbre_numerique ,scan
,fichier ,simulation , placette,
localisation WHERE
(arbre_numerique.sim_id =
simulation.sim_id) AND
(simulation.fic_id =
fichier.fic_id) AND
(fichier.sca_id = scan.sca_id) AND
(scan.pla_id = placette.pla_id)
AND (placette.loc_id =
localisation.loc_id) %1 ORDER BY
arn_id ASC").arg(where));

        int countNumer =
modelNumer->rowCount();
        QString requeteNumer;

        for (int  k = 0; k <
countNumer; k++)
        {
            if (k == 0)
            {
                QString idNumer =
modelNumer-
>record(k).value("arn_id").toStrin
g();
                requeteNumer =
"(arn_id = " + idNumer +") ";
            }
            else
            {
                QString idNumer =
modelNumer-
>record(k).value("arn_id").toStrin
g();
                requeteNumer =
requeteNumer + "OR (arn_id = " +
idNumer + ") ";
            }
        }

        _uri = new
QgsDataSourceURI();
        _uri-
>setConnection(_host,_port,_dbName
,_userName,_password);
        _uri-
>setDataSource("public","arbre_num
erique","the_geom");
        _uri-
>setSql(requeteNumer);
        _vl = new
QgsVectorLayer(_uri-
>uri(),"arbre_numerique",
"postgres");
```

```
        QString way =
"/home/administrateur/cedric/BdTLi
dar_Qgis/arbre.qml";
        bool flag = false;
        _vl-
>loadNamedStyle(way,flag);

QgsMapLayerRegistry::instance()-
>addMapLayer(_vl, true);
        _vl-
>toggleScaleBasedVisibility(flag);
        _iface-
>refreshLegend(_vl);
    }

    if(scan == true)
    {
        QSqlQueryModel* modelScan
= new QSqlQueryModel;
        modelScan-
>setQuery(QString("SELECT sca_id
FROM scan, placette, localisation
WHERE (scan.pla_id =
placette.pla_id) AND
(placette.loc_id =
localisation.loc_id) %1 ORDER BY
sca_id ASC").arg(where));

        int countScan = modelScan-
>rowCount();
        QString requeteScan;

        for (int  l = 0; l <
countScan; l++)
        {
            if (l == 0)
            {
                QString idScan =
modelScan-
>record(l).value("sca_id").toStrin
g();
                requeteScan =
"(sca_id = " + idScan +") ";
            }
            else
            {
                QString idScan =
modelScan-
>record(l).value("sca_id").toStrin
g();
                requeteScan =
requeteScan + "OR (sca_id = " +
idScan + ") ";
            }
        }

        _uri = new
QgsDataSourceURI();
        _uri-
>setConnection(_host,_port,_dbName
,_userName,_password);
        _uri-
>setDataSource("public","scan","th
e_geom");
        _uri->setSql(requeteScan);
        _vl = new
QgsVectorLayer(_uri->uri(),"scan",
"postgres");
        QString way =
"/home/administrateur/cedric/BdTLi
dar_Qgis/scan.qml";
        bool flag = false;
        _vl-
>loadNamedStyle(way,flag);

QgsMapLayerRegistry::instance()-
>addMapLayer(_vl, true);
        _vl-
>toggleScaleBasedVisibility(flag);
        _iface-
>refreshLegend(_vl);
    }
}

QGISEXTERN QgisPlugin*
classFactory(QgisInterface* iface)
{
    return new GMenu(iface);
}

QGISEXTERN QString name()
{
    return "BdTLidar plugin";
}

QGISEXTERN QString description()
{
    return "Un plugin qui permet
d'interagir avec la base de donnée
TLidar";
}

QGISEXTERN QString version()
{
    return "0.00001";
}
```

```
QGISEXTERN int type()
{
    return QgisPlugin::UI;
}

QGISEXTERN void unload(QgisPlugin*
theQgsBdTLidar)
{
    delete theQgsBdTLidar;
}
```

databaseconnection.h :

```
#ifndef DATABASECONNECTION_H
#define DATABASECONNECTION_H

#include "qstring.h"
#include <QtSql>


class DataBaseConnection
{
public:
    DataBaseConnection(QString
host = "localhost", QString
userName = "postgres", QString
password = "Lidar159", QString
dbName = "Lidar_test", int port =
5432, QString driver = "QPSQL");

    bool connect();
    void disconnect();

private:
    QString     _host;
    QString     _password;
    QString     _userName;
    QString     _dbName;
    int         _port;
    QString     _driver;

    QSqlDatabase _db;

public slots:


};



#endif // DATABASECONNECTION_H
```

databaseconnection.cpp :

```
#include "databaseconnection.h"

#include "qdebug.h"
#include "QtSql"
#include <QString>

#include "qtableview.h"
#include
"qsqlrelationaldelegate.h"

DataBaseConnection::DataBaseConnec
tion(QString host, QString
userName, QString password,
QString dbName, int port, QString
driver)
{
    _host = host;
    _password = password;
    _userName = userName;
    _dbName = dbName;
    _port = port;
    _driver = driver;

}

bool DataBaseConnection::connect()
{
   _db =
QSqlDatabase::addDatabase(_driver)
;
   _db.setHostName(_host);
   _db.setDatabaseName(_dbName);
   _db.setUserName(_userName);
   _db.setPassword(_password);
   _db.setPort(_port);

   return _db.open();
}

void
DataBaseConnection::disconnect()
{
    _db.close();
}
```

gaffichagedialog.h :

```
#ifndef GAFFICHAGEDIALOG_H
#define GAFFICHAGEDIALOG_H

#include <QDialog>
```

```
#include <QtSql>

namespace Ui {
class GAffichageDialog;
}

class GAffichageDialog : public
QDialog
{
    Q_OBJECT

public:
    explicit
GAffichageDialog(QWidget *parent =
0);
    ~GAffichageDialog();

public slots:
    void remplirComboBox();

private slots:
    void
on_btn_affichage_clicked();
    void
on_btn_retour_clicked(){reject();}
    void
on_cmb_foret_activated(const
QString &arg1);
    void
on_cmb_parcelle_activated(const
QString &arg1);
    void
on_cmb_placette_activated(const
QString &arg1);

private:
    Ui::GAffichageDialog *ui;
    QString         _foret;
    QString         _parcelle;
    QString         _placette;
    QSqlQueryModel*
_modelQueryForet;
    QSqlQueryModel*
_modelQueryParcelle;
    QSqlQueryModel*
_modelQueryPlacette;

signals:
    void
affichage(bool,bool,bool,bool,QStr
ing,QString,QString);
};
#endif // GAFFICHAGEDIALOG_H
```

gaffichagedialog.cpp :

```
#include "gaffichagedialog.h"
#include "ui_gaffichagedialog.h"
#include "gmenu.h"

#include "qmessagebox.h"
#include "qdebug.h"

GAffichageDialog::GAffichageDialog
(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::GAffichageDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);

    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);

    _foret = "";
    _parcelle = "";
    _placette = "";
}

GAffichageDialog::~GAffichageDialo
g()
{
    delete ui;
}

void
GAffichageDialog::on_btn_affichage
_clicked()
{
    bool placette = false;
    bool scan = false;
    bool arbreNumerique = false;
    bool arbreTerrain = false;

    if(ui->chk_placette-
>isChecked()){placette = true;}
    if(ui->chk_scan-
>isChecked()){scan = true;}
```

```
    if(ui->chk_numerique-
>isChecked()){arbreNumerique =
true;}
    if(ui->chk_terrain-
>isChecked()){arbreTerrain =
true;}

    emit
affichage(placette,scan,arbreNumer
ique,arbreTerrain,_foret,_parcelle
,_placette);
}

void
GAffichageDialog::remplirComboBox(
)
{
    ui->cmb_foret->clear();
    ui->cmb_parcelle->clear();
    ui->cmb_placette->clear();

    _modelQueryForet = new
QSqlQueryModel();
    _modelQueryForet-
>setQuery("SELECT loc_nom_foret
FROM localisation GROUP BY
loc_nom_foret");

    int count = _modelQueryForet-
>rowCount();

    for (int  i = 0; i < count;
i++)
    {
        ui->cmb_foret-
>addItem(_modelQueryForet-
>record(i).value("loc_nom_foret").
toString());
    }
}

void
GAffichageDialog::on_cmb_foret_act
ivated(const QString &arg1)
{
    ui->cmb_parcelle->clear();

    _modelQueryParcelle = new
QSqlQueryModel();
    _modelQueryParcelle-
>setQuery(QString("SELECT
loc_num_parcelle FROM localisation
WHERE loc_nom_foret =
'%1'").arg(arg1));

    int count =
_modelQueryParcelle->rowCount();

    for (int  i = 0; i < count;
i++)
    {
        ui->cmb_parcelle-
>addItem(_modelQueryParcelle-
>record(i).value("loc_num_parcelle
").toString());
    }
    _foret = arg1;
    _parcelle = "";
    _placette = "";
}

void
GAffichageDialog::on_cmb_parcelle_
activated(const QString &arg1)
{
    ui->cmb_placette->clear();

    _modelQueryPlacette = new
QSqlQueryModel();
    _modelQueryPlacette-
>setQuery(QString("SELECT pla_id
FROM placette, localisation Where
placette.loc_id =
localisation.loc_id AND
loc_nom_foret = '%1' AND
loc_num_parcelle =
'%2'").arg(_foret).arg(arg1));

    int count =
_modelQueryPlacette->rowCount();

    for (int  i = 0; i < count;
i++)
    {
        ui->cmb_placette-
>addItem(_modelQueryPlacette-
>record(i).value("pla_id").toStrin
g());
    }
    _parcelle = arg1;
    _placette = "";
}
void
GAffichageDialog::on_cmb_placette_
activated(const QString &arg1)
```

```
{
    _placette = arg1;
}
```

gconnectiondialog.h :

```
#ifndef GCONNECTIONDIALOG_H
#define GCONNECTIONDIALOG_H

#include <QDialog>

namespace Ui {
class GConnectionDialog;
}

class GConnectionDialog : public
QDialog
{
    Q_OBJECT

public:
    explicit
GConnectionDialog(QWidget *parent,
QString host, QString
defaultUserName, QString
defaultPassword, QString dbName,
int port);
    ~GConnectionDialog();

private:
    Ui::GConnectionDialog *ui;
    QString
_defaultUserName;
    QString
_defaultPassword;

public slots:
    void validate();
    void invite(bool checked);

signals:
    void connectionData(QString
host, QString userName, QString
password, QString dbName, int
port);
    void cancel();
};

#endif // GCONNECTIONDIALOG_H
```

gconnectiondialog.cpp :

```
#include "gconnectiondialog.h"
#include "ui_gconnectiondialog.h"

GConnectionDialog::GConnectionDial
og(QWidget *parent, QString host,
QString defaultUserName, QString
defaultPassword, QString dbName,
int port) :
    QDialog(parent),
    ui(new Ui::GConnectionDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);

    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);

    connect(this->ui->btn_ok,
SIGNAL(clicked()), this,
SLOT(validate()));
    connect(this->ui->btn_retour,
SIGNAL(clicked()), this,
SIGNAL(cancel()));
    connect(this->ui-
>chb_parametre,
SIGNAL(toggled(bool)), this,
SLOT(invite(bool)));

    ui->lie_hostname-
>setText(host);
    ui->lie_nom->setText(dbName);
    ui->spb_port->setValue(port);

    _defaultUserName =
defaultUserName;
    _defaultPassword =
defaultPassword;
}

GConnectionDialog::~GConnectionDia
log()
{
    delete ui;
}


void GConnectionDialog::validate()
```

```
{
    emit connectionData(ui-
>lie_hostname->text(),
                        ui-
>lie_utilisateur->text(),
                        ui-
>lie_passe->text(),
                        ui-
>lie_nom->text(),
                        ui-
>spb_port->value());
    close();
}

void
GConnectionDialog::invite(bool
checked)
{
    if (checked) {
        ui->lie_utilisateur-
>setText(_defaultUserName);
        ui->lie_passe-
>setText(_defaultPassword);
        ui->lie_utilisateur-
>setEnabled(false);
        ui->lie_passe-
>setEnabled(false);
    } else {
        ui->lie_utilisateur-
>setText("");
        ui->lie_passe-
>setText("");
        ui->lie_utilisateur-
>setEnabled(true);
        ui->lie_passe-
>setEnabled(true);
    }
}
```

grepresentationdialog.h :

```
#ifndef GREPRESENTATIONDIALOG_H
#define GREPRESENTATIONDIALOG_H

#include <QDialog>

namespace Ui {
class GRepresentationDialog;
}

class GRepresentationDialog :
public QDialog
{
    Q_OBJECT

public:
    explicit
GRepresentationDialog(QWidget
*parent = 0);
    ~GRepresentationDialog();

private slots:
    void on_btn_calcul_clicked();
    void
on_btn_retour_clicked(){reject();}

private:
    Ui::GRepresentationDialog *ui;
    double _coeff;
};

#endif // GREPRESENTATIONDIALOG_H
```

grepresentationdialog.cpp :

```
#include "grepresentationdialog.h"
#include
"ui_grepresentationdialog.h"

#include "QtSql"
#include "qmessagebox.h"

GRepresentationDialog::GRepresenta
tionDialog(QWidget *parent) :
    QDialog(parent),
    ui(new
Ui::GRepresentationDialog)
{
    Qt::WindowFlags flags = 0;
        flags = Qt::Window;
           flags |=
Qt::WindowTitleHint;
           flags |=
Qt::CustomizeWindowHint;
    setWindowFlags(flags);

    ui->setupUi(this);
    this-
>setAttribute(Qt::WA_DeleteOnClose
);

    _coeff = ui->dsb_coefficient-
>value();
}
```

```
GRepresentationDialog::~GRepresentationDialog()
{
    delete ui;
}

void GRepresentationDialog::on_btn_calcul_clicked()
{
    double coeff = ui->dsb_coefficient->value();

    QSqlQuery(QString("UPDATE arbre_terrain SET arb_representation = (arb_circonference / 100)*'%1' WHERE (arb_circonference IS NOT NULL)").arg(coeff));
    QSqlQuery(QString("UPDATE arbre_terrain SET arb_representation = ((arb_diam1 * PI()) / 100)*'%1' WHERE (arb_circonference IS NULL) AND (arb_diam1 IS NOT NULL) AND (arb_diam2 IS NULL)").arg(coeff));
    QSqlQuery(QString("UPDATE arbre_terrain SET arb_representation = ((((arb_diam1 * arb_diam2)/2) * PI()) / 100)*'%1' WHERE (arb_circonference IS NULL) AND (arb_diam1 IS NOT NULL) AND (arb_diam2 IS NOT NULL)").arg(coeff));

    QSqlQuery(QString("UPDATE arbre_numerique SET arn_representation = (arn_circonference / 100)*'%1' WHERE (arn_circonference IS NOT NULL)").arg(coeff));
    QSqlQuery(QString("UPDATE arbre_numerique SET arn_representation = ((arn_diam1 * PI()) / 100)*'%1' WHERE (arn_circonference IS NULL) AND (arn_diam1 IS NOT NULL) AND (arn_diam2 IS NULL)").arg(coeff));
    QSqlQuery(QString("UPDATE arbre_numerique SET arn_representation = ((((arn_diam1 * arn_diam2)/2) * PI()) / 100)*'%1' WHERE (arn_circonference IS NULL) AND (arn_diam1 IS NOT NULL) AND (arn_diam2 IS NOT NULL)").arg(coeff));

    QMessageBox::information(0, "Calcul Representation", "Calcul finie");
}
```