



**Institut Supérieur d'Informatique, de
Modélisation et de leurs Applications.**

Complexe des Cezeaux
BP125 – 63173 Aubière CEDEX



CERN
**European Organisation for
Nuclear Research**
CH-1211 GENEVE 23
Suisse

Rapport de stage de 2^{ème} année
Filière Calcul et Modélisation Scientifiques

Visualisation de la base de données de configuration de l'expérience LHCb

Présenté par : **Walid Belballi**
Responsable CERN : **Mr Eric Van Herwijnen**
Responsable ISIMA : **Mr Emmanuel Mesnard**

Du 2 avril 2007
au 31 août 2007

Remerciements

Je tiens tout d'abord à remercier mon encadrant au CERN, Eric van Herwijnen, pour m'avoir aidé tout au long de ces cinq mois, pour ses remarques pertinentes qui m'ont permis d'améliorer mon travail et aussi pour sa gentillesse.

Je remercie aussi Niko Neufeld et Sai Suman Cherukwda pour avoir répondu à beaucoup de mes questions.

Je tiens aussi à remercier Emmanuel MESNARD pour être venu me rendre visite dans les locaux du CERN, Radu Stoica pour son aide concernant la bibliothèque Qt.

Je remercie enfin Lana Abadie pour avoir répondu à toutes mes questions concernant la base de données de configuration, mon collègue Stefan Koestner pour sa gentillesse et tous les membres de l'équipe Online pour leur accueil.

Résumé

L'Organisation européenne pour la Recherche Nucléaire (CERN) situé près de Genève sur la frontière franco-suisse accueillera, en Mai 2008, le plus grand collisionneur de particules jamais construit : le LHC (Large Hardon Collider).

Le LHC est l'un des instruments scientifiques les plus grands et les plus complexes. Configurer les différents modules de l'expérience est une opération très sensible et fastidieuse vue le nombre de composants et de technologies mis en jeux.

D'où l'utilité des bases de données de configuration qui permettent d'organiser toutes les informations nécessaires à la bonne configuration du système.

La contribution de mon stage est l'amélioration d'un éditeur de la base de données de configuration du LHCb qui est une des quatre expériences menées au CERN.

L'éditeur est appelé CDBVis et permet de visualiser la topologie des différents sous détecteurs constituant le système.

Mon travail a consisté à porter l'application qui utilise la bibliothèque graphique wxWidget associée au langage Python vers la bibliothèque Qt qui est plus facile à utiliser. Le portage de l'application s'est bien déroulé et plusieurs autres améliorations ont été apportées à l'application.

Mots clefs :

CERN, LHCb, base de données de configuration, interface graphique, Python, wxWidget, Qt

Abstract

The world's largest proton collider, LHC (Large Hardon Collider), is currently build in the **European Organization for Nuclear Research (CERN)** located on the French-Swiss border. The LHC is scheduled to begin operation in May 2008 and will be one of the most complex scientific instruments ever built.

Configuring all the modules of the experiment is a complex task because of the huge network of devices and the big number on technologies involved.

That's why configuration databases are used to store and organise the informations needed to well configure the system.

My contribution was to improve an editor of the configuration database of the LHCb wich is one the four experiments taking place at CERN.

This editor is called CDBVis and my work consist on porting the software written in Python to use the Qt library instead of the wxWidget one. I also made many enhancements to this software.

Key words :

CERN, LHCb, configuration database, Graphical User Interface, Python, wxWidget, Qt

Glossaire

- **CDBVis** : Configuration Database Visualizer ; le nom de l'application permettant de visualiser la connectivité de la base de données de configuration.
- **CIC DB** : Configuration Inventory Connectivity DataBase ; base de données de configuration contenant aussi des informations sur la connectivité du système.
- **CIC DB Lib** : Configuration Database Library ; bibliothèque écrite en C constituant une interface pour la base de données de configuration
- **GUI** : Graphical User Interface, se dit d'une application graphique qui ne tourne pas qu'en mode console.
- **SQL** : Structured Query Language, un langage informatique permettant de communiquer avec les bases de données.
- **Widget** : un composant d'une interface graphique
- **Antiparticule** : A chaque type de particule correspond une antiparticule. Lorsqu'une particule entre en collision avec son antiparticule, elles s'annihilent, ne laissant que de l'énergie
- **Hadron** : Particule non élémentaire contenant des quarks et des antiquarks
- **Meson** : Particule non élémentaire de la famille des hadrons.
- **Quark** : particule élémentaire chargée qui est sensible à l'interaction forte. Elle existe en 6 sorte différentes notées u,d,s,c,b et t.

Table des matières

REMERCIEMENTS

RESUME

ABSTRACT

GLOSSAIRE

TABLE DES MATIERES

TABLE DES FIGURES

INTRODUCTION	6
PRESENTATION GENERALE	7
1) PRESENTATION DU CERN	7
2) L'EXPERIENCE LHCB	8
a) LHCb : Le détecteur	8
b) Système « Online »	9
c) Configuration de l'expérience LHCB.....	9
d) Outils de configuration :	10
PVSS :	10
CDBVis :	11
ANALYSE DU PROBLEME.....	13
1) ETUDE DES BESOINS.....	13
2) COUCHE TRAITEMENT DE DONNEES.....	13
3) SCHEMA DE LA BASE DE DONNEES.....	13
a) Inventaire des composants	13
b) Connectivité.....	15
c) Schéma de la base de données.....	15
4) COUCHE PRESENTATION DES DONNEES.....	16
a) Présentation des composants dans CDBVis.....	17
b) Exemple : système MUON	17
c) Python.....	20
d) WxPython	21
5) MIGRATION VERS QT	22
a) Gestion des événements	23
b) L'outil Qt Designer.....	23
c) Génération de code : Le Compilateur d'Interfaces Utilisateur (uic).....	25
REALISATION DE LA SOLUTION	26
1) DEFINITION DE L'ENVIRONNEMENT DE TRAVAIL.....	26
2) ARCHITECTURE DE L'APPLICATION CDBVIS : UML, DIFFERENTES CLASSES	27
3) DEMARCHE SUIVIE POUR LE PORTAGE DE L'APPLICATION.....	28
a) Identification les classes à modifier	29
b) Utilisation d'interfaces.....	29
c) Partie visuelle : Canevas	30
Gestion des collisions:.....	32
Matrices de transformations:	33
d) Séparer le contenu de la présentation.....	33

4)	AFFICHAGE DES GRANDS COMPOSANTS.....	34
5)	MISE A DISPOSITION DES UTILISATEURS.....	36
RESULTATS ET DISCUSSIONS		38
1)	PRESENTATION GENERALE DE CDBVIS.....	38
a)	<i>Vue Principale</i>	38
b)	<i>Zone de sélection</i>	39
c)	<i>Zone d'information</i>	41
d)	<i>Barre d'état</i>	41
2)	MODE DE NAVIGATION	42
a)	<i>Visualisation des voisins</i>	42
b)	<i>Visualisation des chemins</i>	43
3)	MODE DE CREATION	45
a)	<i>Création et modification des composants</i>	45
	Création d'un nouveau Type de composant	45
	Création de composant(s)	46
	Modification de composant	47
b)	<i>Création de ports</i>	47
c)	<i>Création de connexions</i>	48
d)	<i>Exportation des données dans un fichier</i>	50
e)	<i>Création à partir d'un fichier</i>	50
CONCLUSION.....		52
REFERENCES BIBLIOGRAPHIQUES		53
ANNEXE I		I
ANNEXE II.....		V
ANNEXE III		V
ANNEXE III		VI

Table des figures

Figure 1 – Deux vue différentes du LHC.....	7
Figure 2 - Vue d'ensemble du détecteur LHCb.....	8
Figure 3 - Vue générale du système «Online» montrant le positionnement de la CIC DB parmi les autres composants du système.	10
Figure 4 - Exemple d'une interface graphique PVSS montrant le commutateur TFC.	11
Figure 5- Exemple d'une interface graphique montrant un switch du DAQ.....	12
Figure 6 - Transitions possibles entre états d'un composant.....	15
Figure 7 - relation entre l'état de la partie hardware et la partie fonctionnelle d'un composant.....	15
Figure 8 - Schéma de la base de données de configuration	16
Figure 9 - Représentation d'un composant dans CDBVis.....	17
Figure 10 - topologie du système MUON.....	18
Figure 11 - Chemin reliant la carte frontale (Front End Board) à la HVBRD_CAEN (High Voltage Board)	19
Figure 12 - chemin reliant la carte frontale aux différents composants du MUON Chamber.....	20
Figure 13 - Logo de Python.....	21
Figure 14 - Logo de wxPython	21
Figure 15 - Différence entre l'apparence de la même application sur deux OS différents (Windows XP et GNOME).....	21
Figure 16 - Logo de la librairie Qt.....	22
Figure 17 - Mécanisme de signaux/slots dans Qt	23
Figure 18 – Vue d'ensemble de l'outil Qt Designer.....	24
Figure 19 - Création de connexion signaux/slots dans Qt Designer.....	25
Figure 20 - processus de développement en utilisant les outils de Qt	25
Figure 21 – L'environnement de l'application CDBVis.....	26
Figure 22 - Diagramme UML de la l'application CDBVis.....	28
Figure 23 - Utilisation des interfaces de communication	29
Figure 24 - Exemple d'une zone de défilement	31
Figure 25 - Exemple d'une application utilisant le Canvas module de Qt.....	32
Figure 26 - Séparation entre la partie graphique du code et la partie non graphique	33
Figure 27 - Interface utilisant des Layouts.....	34
Figure 28 - Interface réalisé sans Layouts	34
Figure 29 - Représentation du routeur du DAQ avec toutes ses connexions.....	35
Figure 30 - visualisation des grands composants (ici le DAQ_ROUTER_1)	36
Figure 31 - Vue d'ensemble de l'application CDBVis.....	39

Figure 32 - Deux vues de la zone de selection montrant les deux premiers niveaux de l'arbre	40
Figure 33 - Les trois derniers niveaux (types de composants, composants, ports) de l'arbre contenant tous les composants.....	40
Figure 34 - Deux vues de la zone d'information ; dans la première vue un composant est sélectionné et dans la deuxième il s'agit d'une connexion.....	41
Figure 35 - Barre de status de l'application	42
Figure 36 - Visualisation des voisins du VELO_HYBRID_R_0	42
Figure 37 - Visualisation des voisins du VELO_SHORT_KAPTON_0	43
Figure 38 - Bouton permettant de passer au mode de visualisation de chemins .	43
Figure 39 - Visualisation des chemins passant par la carte VELO_TELL1_0	44
Figure 40 - Formulaire de création d'un nouveau type de composant	45
Figure 41 - Formulaire de creation de composant(s)	46
Figure 42 - Menu contextuel suite à un clique droit sur le VELO_REPEATER_0	47
Figure 43 - Formulaire de création de ports	48
Figure 44 - Formulaire de création de connexion.....	49

Introduction

Pour explorer l'infiniment petit, il faut voir très très grand comme en témoigne le LHC (Large Hadron Collider), cet immense collisionneur de 27 kilomètres de circonférence érigé de part et d'autre de la frontière franco-suisse par le Conseil Européen pour la Recherche Nucléaire (CERN). 4 milliards d'euros et la collaboration de quelques 6500 scientifiques et techniciens ont été nécessaires pour venir à bout de ce défi technologique qui prendra fonction en Mai 2008.

Le LHC aura pour mission, à travers ses quatre détecteurs, de voir la physique des particules sous une nouvelle lumière en permettant aux physiciens de faire des expériences à des énergies encore jamais atteintes.

Les détecteurs du LHC sont d'une très grande complexité, il est donc nécessaire de sauvegarder et d'organiser toutes les informations relatives à leur installation et à leur configuration. C'est ainsi que des bases de données de configuration ont été créées pour rendre disponible cette énorme quantité d'information.

Mon travail a porté sur un outil de visualisation de la base de données de configuration du LHCb (Large Hardon Collider beauty), l'un des quatre détecteurs du LHC qui étudiera les collisions mettant en jeu un quark b dit « beauté » ou son antiquark.

L'outil de visualisation consiste en une application graphique qui permet de visualiser la liste de tous les composants de l'expérience ainsi que leur connectivité ce qui permettra de mieux appréhender la topologie du système afin de pouvoir localiser les éventuels dysfonctionnements et leurs origines.

Cette application porte le nom de CDBVis (Configuration DataBase Visualizer) et mon travail a principalement consisté à la porter vers une nouvelle bibliothèque graphique offrant plus de possibilités tout en améliorant les fonctionnalités déjà présentes dans l'application.

Présentation générale

1) Présentation du CERN

Le 24 septembre 1954 à Genève, douze Etats ratifient la convention de l'Organisation Européenne pour la Recherche Nucléaire. Ils signent l'acte de naissance juridique de ce qui deviendra le plus grand laboratoire de physique des particules au monde.

Aujourd'hui, le Conseil Européen pour la Recherche Nucléaire compte une vingtaine de pays participants et abrite des centaines de physiciens et d'ingénieurs européens en plus du grand nombre de scientifiques internationaux qui y mènent leurs recherches. Il contiendra aussi d'ici 2008 le plus grand accélérateur de particules au monde qui permettra de recréer les conditions de températures qui prévalaient dans l'Univers à ses tout premiers instants après le Big Bang.

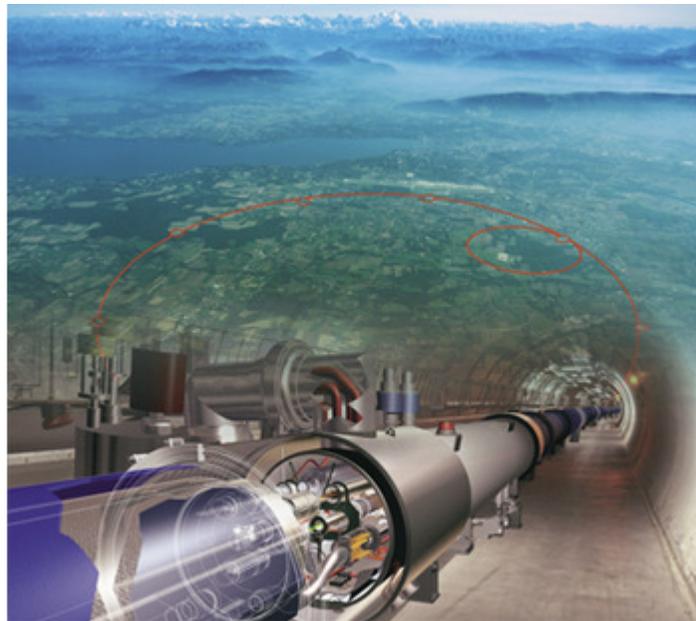


Figure 1 – Deux vue différentes du LHC

Installé à une centaine de mètres sous terre, le nouveau collisionneur de 27 kilomètres de circonférence aura pour objectif d'accélérer et de faire collisionner deux faisceaux de protons à une vitesse proche de celle de la lumière. Les collisions auront lieu en quatre points de l'anneau où se trouvent les quatre détecteurs du LHC (ATLAS, ALICE, CMS, LHCb) et devront produire un ensemble de particules secondaires dont l'étude permettra de révéler quelques un des secrets les mieux gardés de notre Univers tels que dévoiler l'existence du boson de « Higgs » soupçonné d'être à l'origine de la masse des particules ou révéler l'origine du déséquilibre matière-antimatière...

2) L'expérience LHCb

L'expérience LHCb réunit près de 600 scientifiques issus de 47 laboratoires et de 15 pays. Le détecteur LHCb traquera les particules contenant un quark b dit « beauté » et son antiquark dans l'ultime but d'expliquer comment la matière a pris le dessus sur l'antimatière alors qu'elles étaient présentes en quantités égales à la naissance de l'Univers.

a) LHCb : Le détecteur

L'expérience LHCb est composée d'un ensemble de sous détecteurs complémentaires :

- VELO : (Vertex Locator) permet de déterminer l'emplacement des collisions avec une précision de 10 microns.
- RICH 1&2 : (Ring Imaging Cherenkov) permettent d'identifier les particules chargées ainsi que leur quantité de mouvement.
- TRACKERS : fournit des mesures précises sur les quantités de mouvement des particules chargées.
- CALORIMETERS : mesurent principalement l'énergie des particules contenant des quarks. Ils sont en nombre de quatre. Le HCAL, par exemple, possède 80 km de fibres et est constitué d'un empilement de plaques en acier qui pèsent 500 tonnes.
- MUON : sert pour l'identification des muons.

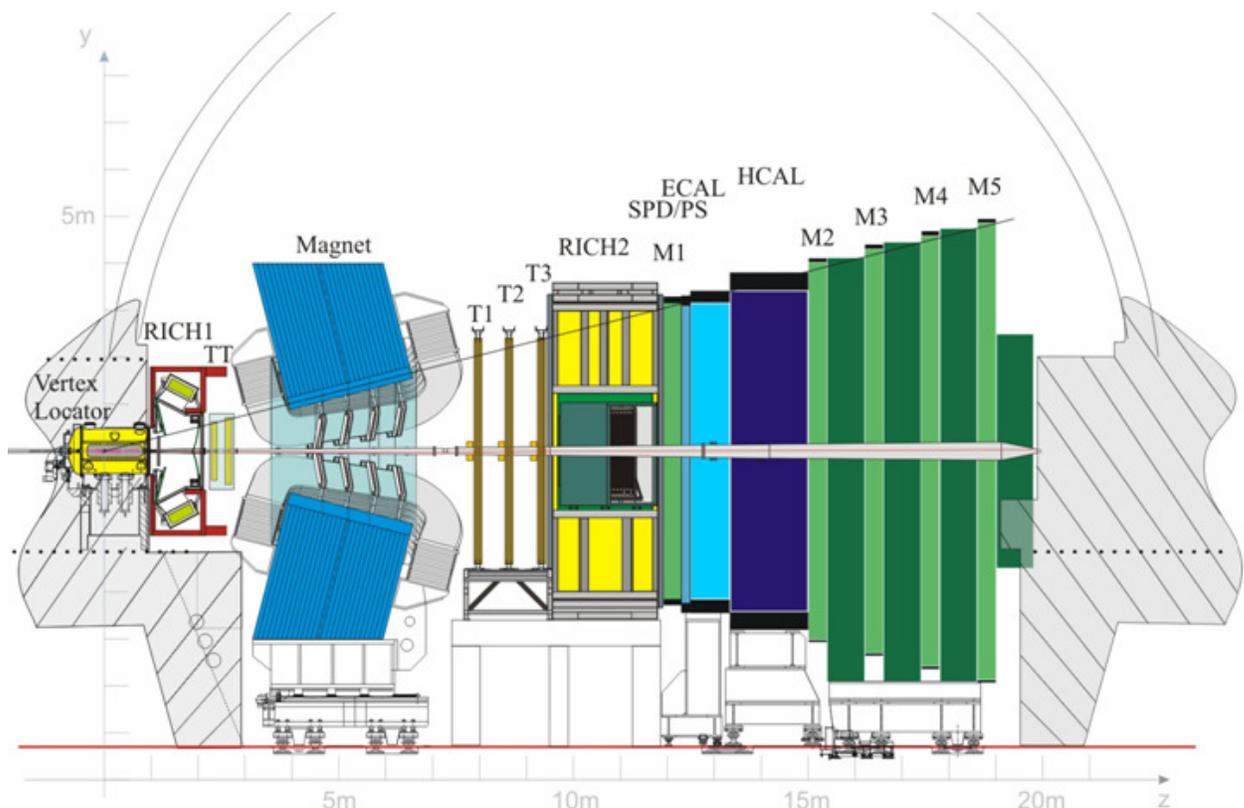


Figure 2 - Vue d'ensemble du détecteur LHCb

b) Système « Online »

En plus de l'ensemble des sous détecteurs, L'expérience LHCb s'appuie sur un système dit « Online » qui permet de superviser le déroulement de l'expérience et qui est composé de quatre éléments :

Le Trigger : Les collisions ont lieu à une fréquence de 40 MHz (une collision toute les 25 ns), ce qui représente une grande quantité de données à traiter. Le trigger a pour but de réduire cette fréquence en ne sélectionnant que les événements importants.

Le DAQ (Système d'acquisition de données) est un réseau Gigabit Ethernet qui s'occupe du transport des données du détecteur jusqu'aux disques de stockage en passant par les PCs de la ferme où s'effectue la sélection des événements.

Le TFC (Timing and Fast Control) : permet de synchroniser tous les modules de l'expérience.

ECS (Experiment Control System) : Le système de contrôle de l'expérience LHCb est en charge de la configuration, du contrôle et de la surveillance de tous les composants du détecteur.

c) Configuration de l'expérience LHCb

Avec plus de 500 000 composants, le détecteur du LHCb est un système d'une grande complexité. La configuration de tous ses composants devient rapidement une opération fastidieuse, d'autre part il n'est pas toujours possible de se déplacer sur le lieu où se trouve le matériel, celui-ci pouvant de plus se trouver dans une zone soumise aux radiations.

Il convient donc d'automatiser ces opérations et stocker toutes les informations relatives à la configuration qui peuvent aller de quelques kB à quelques MB selon le type du composant.

C'est ici qu'intervient la base de données de configuration du LHCb, plus communément appelée Conf DB ou CIC DB (Configuration Inventaire Connectivité), qui devrait contenir toutes les informations dont a besoin le système de contrôle (ECS) pour définir les paramètres de configuration des composants de l'expérience.

La CIC DB comporte aussi :

- Une description complète de la topologie du système (connexions entre composants).
- La hiérarchie des différents éléments.
- Les caractéristiques physiques et fonctionnelles des équipements utilisés.
- Un historique et inventaire des modules de l'expérience.

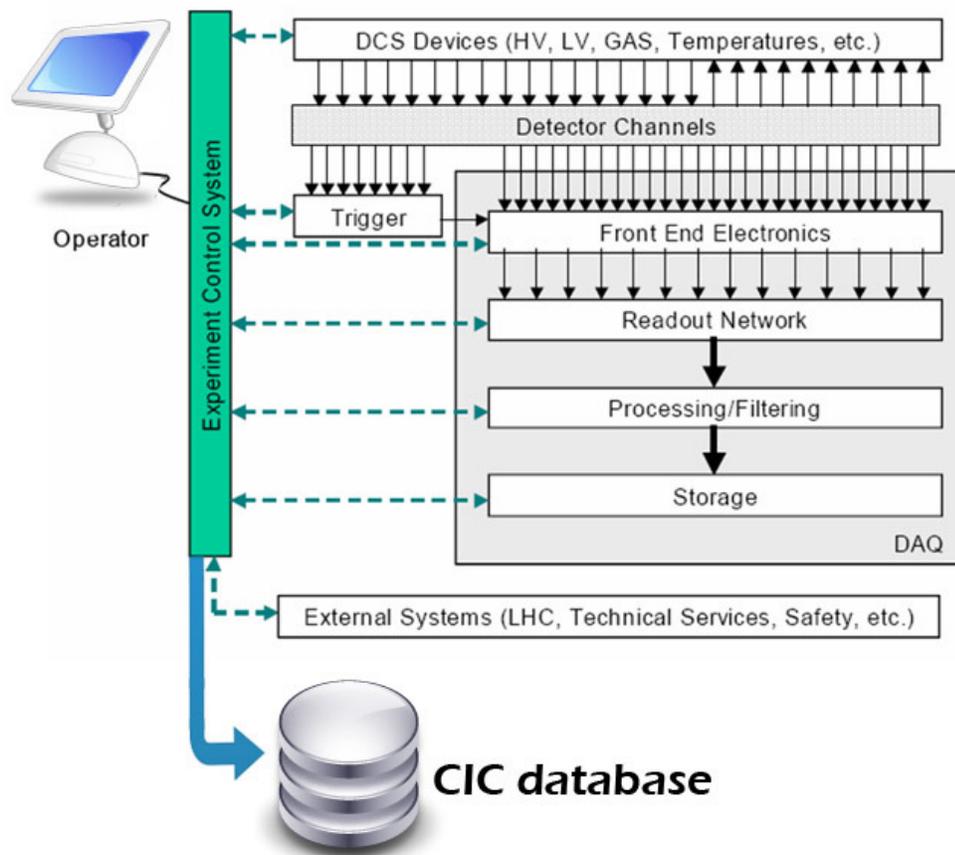


Figure 3 - Vue générale du système «Online» montrant le positionnement de la CIC DB parmi les autres composants du système.

d) Outils de configuration :

PVSS :

Le système de contrôle du LHCb repose sur PVSS qui est un logiciel de type SCADA (Supervisory Control And Data Acquisition, Superviseur de contrôle et d'acquisition de données).



Ce logiciel a été développé par la société autrichienne ETM et permet de récupérer des données à partir du matériel afin de surveiller et contrôler son fonctionnement ainsi que de configurer le matériel à distance. Tous les modules contrôlables doivent donc être représentés dans PVSS.

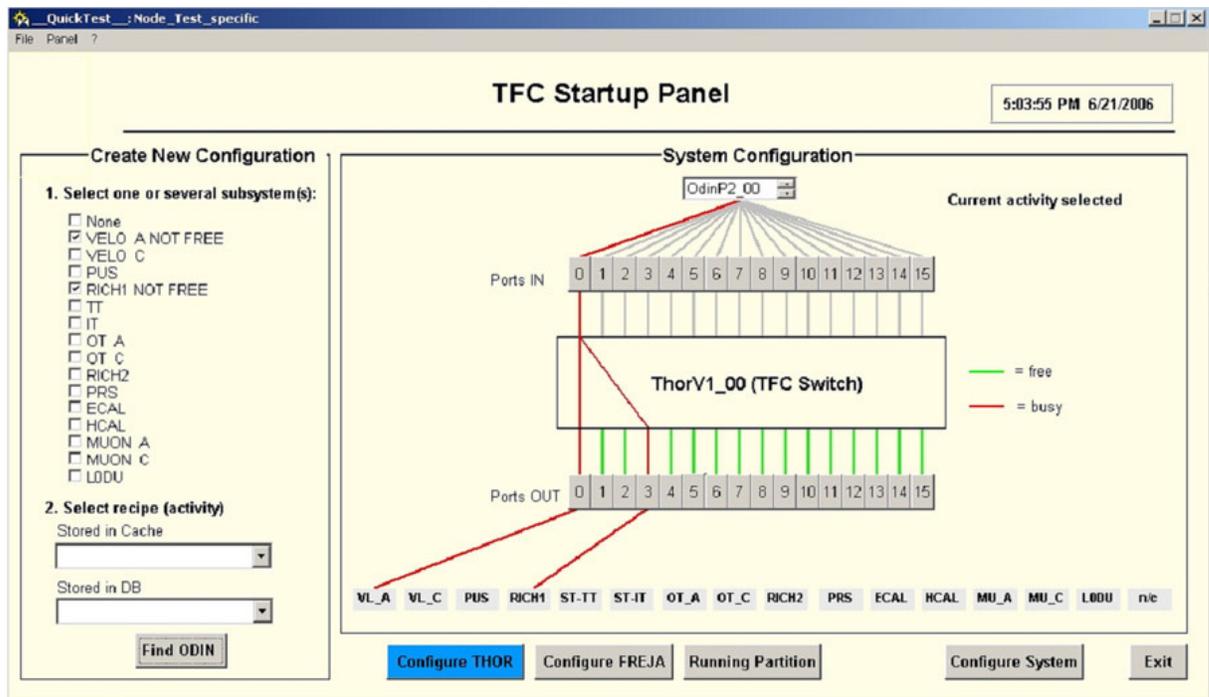


Figure 4 - Exemple d'une interface graphique PVSS montrant le commutateur TFC.

CDBVis :

Bien que PVSS permet de représenter fidèlement tous les composants du détecteur, il n'est pas très utile quand il s'agit de visualiser la topologie du système ou comment les différents modules sont reliés entre eux.

De même, la plupart des éditeurs de bases de données classiques offrent une bonne idée de la structure des tables et de leur contenu mais ils restent inadaptés pour visualiser la connectivité contenue dans la Conf DB de manière intuitive. En effet, si on considère que le détecteur est constitué de plusieurs composants reliés entre eux à travers leurs différents ports par des connexions, et que les composants, les ports et les connexions sont stockés dans la Conf DB dans des tables différentes, il devient donc difficile de déterminer quel composant est relié à quel autre. Car pour savoir que le composant A est connecté au composant B. Il va falloir consulter la table des ports pour en extraire les ports appartenant à A et à B, pour ensuite rechercher dans la table des connexions les liens reliant un des ports de A à un des ports du composant B.

Il paraît donc clair, qu'un éditeur graphique de la Conf DB, permettant de visualiser de manière intuitive comment les composants sont reliés entre eux, sera d'une grande utilité pour naviguer dans le contenu de la base de données de configuration. C'est dans cette optique qu'a été développé l'éditeur graphique CDBVis sur lequel a porté mon travail pendant toute la période de mon stage.

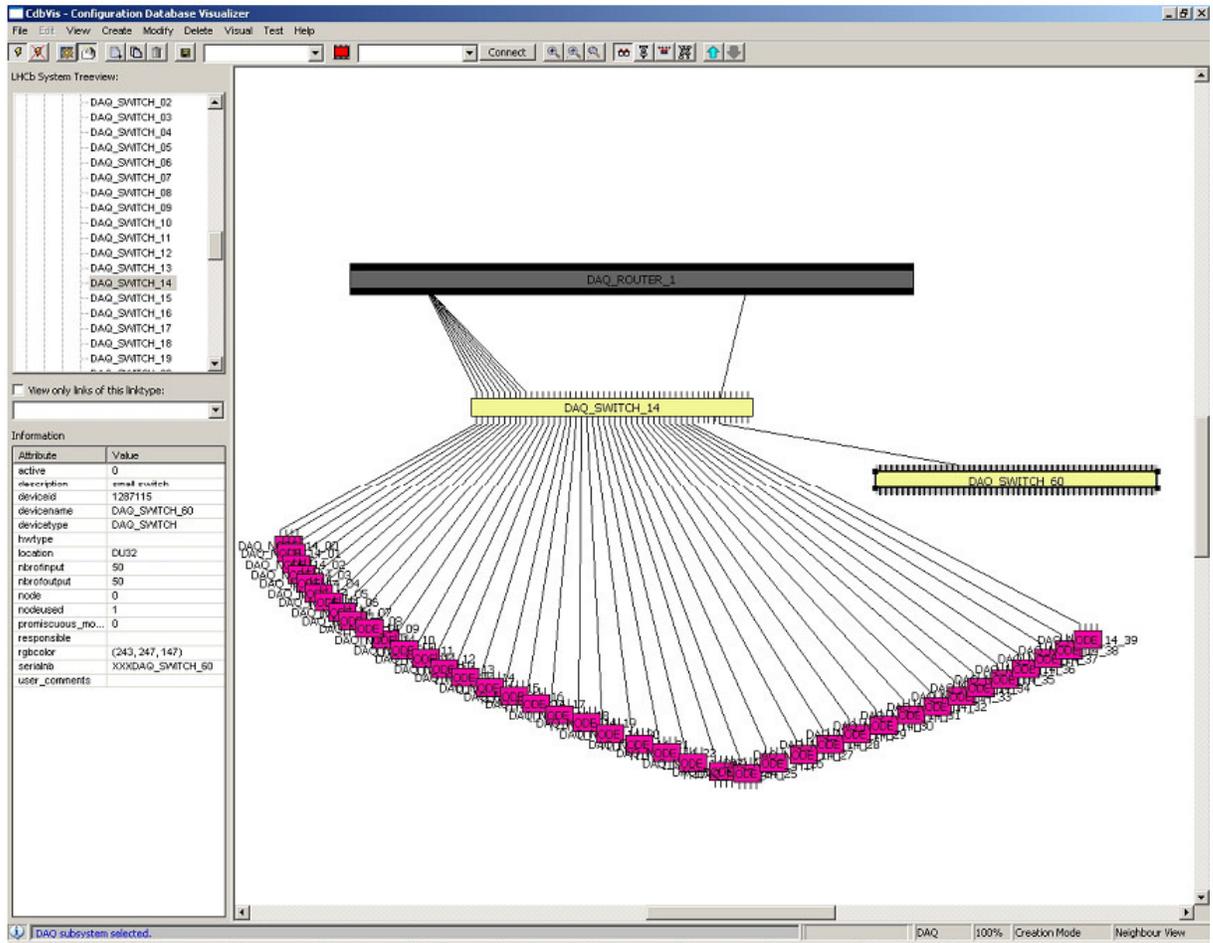


Figure 5- Exemple d'une interface graphique montrant un switch du DAQ

Analyse du problème

1) Etude des besoins

Comme expliqué précédemment, l'éditeur CDBVis devra permettre une navigation simple et intuitive dans le réseau des composants du détecteur. Ceci permettra entre autre de détecter rapidement les erreurs liées à la topologie du système. Par exemple dans le cas du dysfonctionnement d'un des composants on pourrait déterminer facilement les autres composants qui lui sont reliés et qui seront affectés ou bien en cause de ce dysfonctionnement.

De même, connaître la topologie du système permettra de suivre le flux des données d'un bout à l'autre du détecteur.

CDBVis devra aussi permettre l'ajout de nouveaux composants dans la ConfDB, la modification des paramètres d'un composant et prévoir une option pour l'insertion d'une grande quantité de données. CDBVis fait appel pour cela à des bibliothèques de fonctions intelligentes permettant de manipuler les informations contenues dans la base de données de manière sûre et cohérente sans connaissance préalable de la structure de la base de données.

2) Couche traitement de données

La CIC DB dispose d'une couche applicative permettant d'accéder aux données et leurs manipulations d'une manière simple et sûre afin de préserver la cohérence de données contenues dans la CIC DB et de permettre aux utilisateurs d'interroger la base de données sans aucune connaissance du schéma des tables ou du langage SQL.

Cette couche applicative a été mise sous forme d'une bibliothèque écrite en C, appelée **CIC Lib**, et qui regroupe un ensemble de fonctions qui peuvent être appelées par l'utilisateur ou par d'autres applications.

La CIC Lib permet aussi de protéger le schéma de la base de données dans la mesure où elle nous évite de changer toutes les applications utilisant la CIC DB dans le cas où un changement imprévu du schéma de la base de données venait à arriver. Dans un tel cas de figure on n'aura besoin que de changer notre librairie pour prendre en compte le changement du schéma de la base de données et cette opération sera dès lors transparente pour les applications utilisant la CIC DB du moment où on s'arrange à leur fournir les mêmes fonctions pour communiquer avec la base de données.

3) Schéma de la base de données

a) Inventaire des composants

Le schéma de la CIC DB a été élaboré pour répondre aux différents cas d'utilisation demandés par l'ensemble des utilisateurs. Il doit pour cela être générique pour s'appliquer à n'importe quel sous système du détecteur et extensible

pour pouvoir inclure facilement d'ultérieurs changement non prévus au départ. Ce travail a été réalisé en 2006 par Lana Abadie lors de sa thèse au CERN.

Il a ainsi été choisit de distinguer dans le schéma de la CIC DB entre le coté fonctionnel et le coté matériel ou hardware des différents composants. Ainsi, à chaque composant de l'expérience sera associées deux entrées dans la base de données dans deux tables différentes :

- La table FUNCTIONAL_DEVICES : contient les informations relatives à la fonction du composant.
- La table HARDWARE_DEVICES : contient les informations relatives à la partie purement matériel du composant (numéro de série, nom du responsable à prévenir dans le cas d'une éventuelle panne).

Une telle distinction a été adoptée car il est plus facile de retenir le nom fonctionnel d'un composant plutôt que son numéro de série. Ainsi, la carte qui porte le numéro de série XDHFFD41 (hardware) accomplit les tâches de MUON_TELL1_12 (fonctionnel) et si jamais la carte MUON_TELL1_12 tombe en panne, la carte du point de vue élément physique sera remplacée. Et donc par la suite, c'est une carte qui porte le numéro de série GT54RVSS, par exemple, qui accomplira les tâches de MUON_TELL1_12. Le nom fonctionnel reste inchangé au cours du temps.

Pour pouvoir donner une vue générale de l'état du système, plusieurs états possibles ont été définis pour tous les composants. Ainsi un composant matériel peut être:

- IN_USE : si le composant a bien été installé sur les lieux de l'expérience et est prêt à être utilisé.
- SPARE : s'il s'agit d'une pièce de rechange qui n'est pas connectée au système.
- IN_REPAIR : si le composant est en réparation.
- DESTROYED : si le composant a été endommagé et ne peut plus être utilisé.
- EXT_TEST : si le composant a été déconnecté du système pour être testé en laboratoire.
- TEST : si le composant est testé localement (sans être déconnecté du système)

Etat initial d'un composant	Statut autorisés auxquels il peut transiter
IN_USE	SPARE, IN_REPAIR, TEST, DESTROYED, EXT_TEST
SPARE	IN_USE
TEST	SPARE, IN_REPAIR, IN_USE, DESTROYED
EXT_TEST	SPARE, IN_REPAIR, IN_USE, DESTROYED

IN_REPAIR	SPARE, IN_USE, DESTROYED
DESTROYED	

Figure 6 - Transitions possibles entre états d'un composant

L'état du composant fonctionnel peut être déduit directement de celui du composant matériel suivant le tableau suivant :

Composant matériel (hardware)	Composant fonctionnel
IN_USE	IN_USE
SPARE	indéfini
TEST	indéfini
EXT_TEST	indéfini
IN_REPAIR	indéfini
DESTROYED	indéfini

Figure 7 - relation entre l'état de la partie hardware et la partie fonctionnelle d'un composant

b) Connectivité

Dans le schéma de la base de données, chaque composant fonctionnel dispose de plusieurs ports à travers lesquels il est connecté aux autres composants du système. Les liens entre composants peuvent être de plusieurs types différents et sont stockés dans la table CONNECTIVITY.

Les liens peuvent être des connexions physiques (conduites de gaz, files de transport de données...) ou bien des connexions logiques qui servent à regrouper les composants qui sont traités de la même manière ou qu'on a l'habitude de ranger dans la même catégorie.

c) Schéma de la base de données

Après l'étude des différents cas d'utilisation possible, le schéma suivant a été élaboré pour la base de données de configuration :

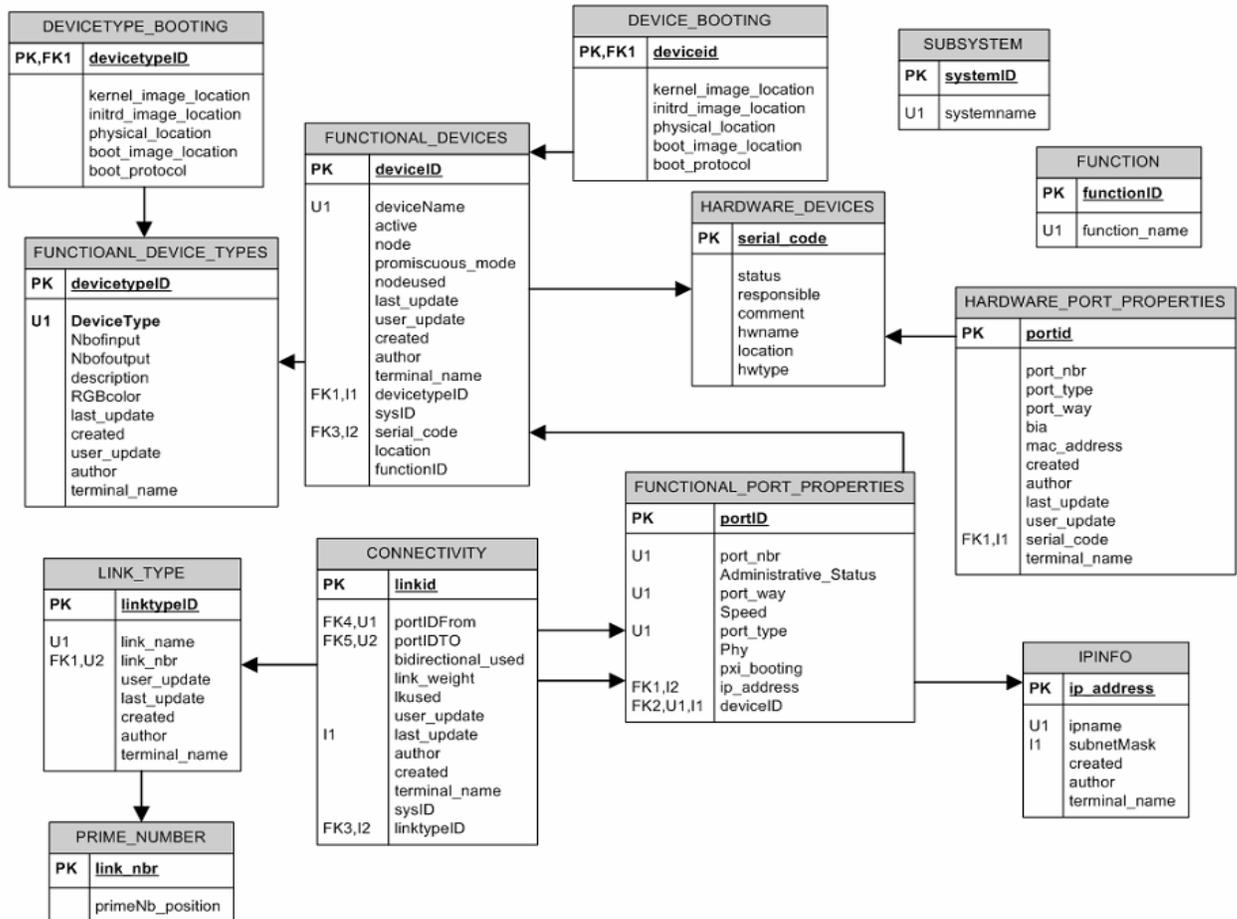


Figure 8 - Schéma de la base de données de configuration

Dans le schéma présenté ci-dessus, on voit que les composants fonctionnels sont regroupés selon différents types (chaque composant fonctionnel est associé à un type donné dans la table FUNCTIONAL_DEVICE_TYPES). Ceci permet de regrouper les propriétés communes à plusieurs composants et d'éviter la duplication inutile d'information.

On remarque aussi que la table contenant les types de connexions est reliée à une table contenant une liste de nombres premiers. En effet, vu le nombre réduit de type de liens dans le système (14 au total), à chaque type de lien a été associé un nombre premier qui lui sert d'identifiant. Ceci facilite le stockage et la gestion des liens car il est beaucoup plus aisé de manipuler des nombres que de d'utiliser des chaînes de caractères.

4) Couche présentation des données

La couche présentation des données de la CIC DB repose sur des interfaces graphiques PVSS et sur CDBVis sur lequel s'est concentré mon travail pour toute la durée de mon stage.

a) Présentation des composants dans CDBVis

Les composants du système sont représentés dans CDBVis par des rectangles avec les ports d'entrée (input) sur la partie supérieure et les ports de sortie sur la partie inférieure comme le montre la figure ci-dessous. Les deux rangées de ports sont numérotées à partir de zéro.

Les composants sont affichés de différentes couleurs selon leur type. De même, plusieurs couleurs sont utilisées pour distinguer les 14 types de connexions présentes dans le système.

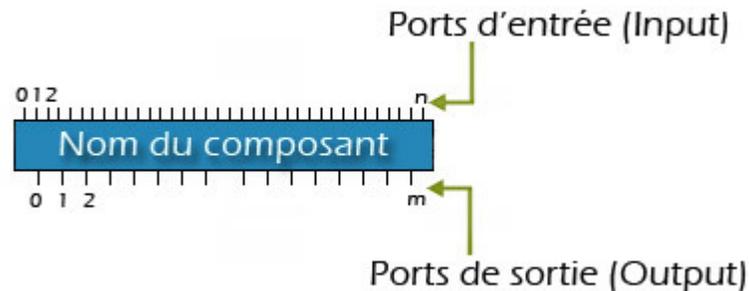


Figure 9 - Représentation d'un composant dans CDBVis

b) Exemple : système MUON

Pour illustrer de quelle manière l'application CDBVis permet de représenter la topologie des différents systèmes considérons le système MUON comme exemple.

La figure suivante montre une représentation du système MUON donnée par le groupe travaillant dessus. On distingue la carte frontale (*FE_M3C17C*) en couleur verte qui dispose d'une connectivité interne et qui est reliée aux autres composants spécifiques au système : les cartes *PAD* suivies des cartes *GAP* et pour finir le *Chamber* et les *HV Ch*.

On note aussi que trois différents types de connexions sont mises en jeu ici : les connexions de transport de données (data link), les files de hautes tensions (High Voltage Link) et des liens d'association (part Link) dont le rôle est d'indiquer que les cartes *Gap* sont associées au *Chamber* même s'il n'y a aucune connexion physique entre les deux.

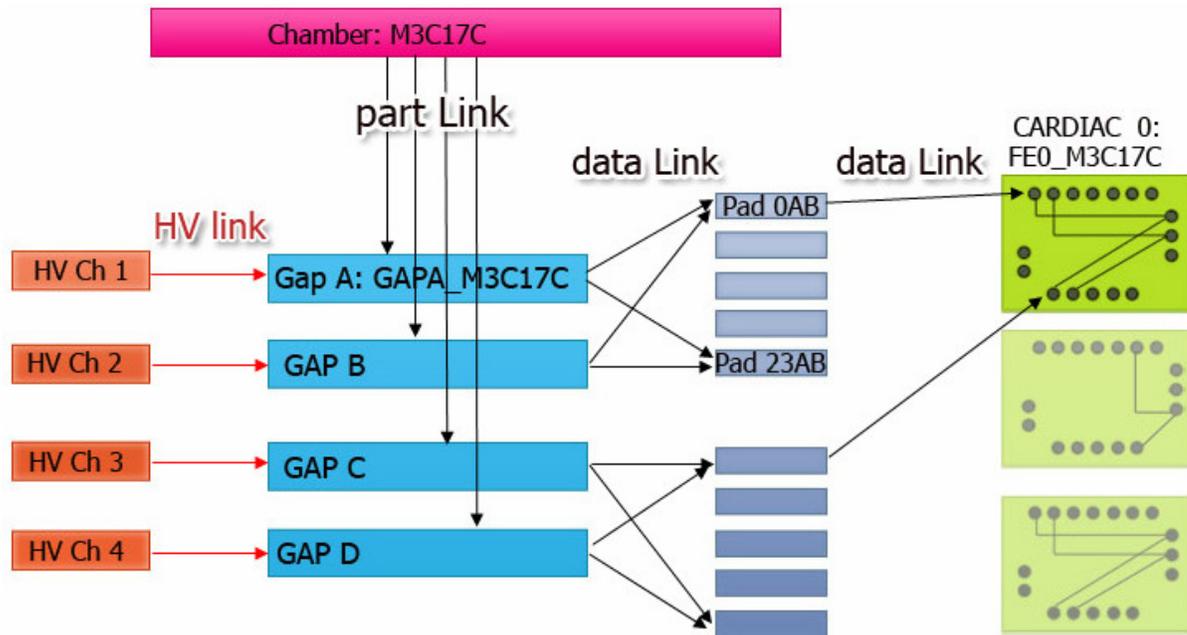


Figure 10 - topologie du système MUON

Bien que la gestion de la connectivité microscopique soit incluse dans la CIC DB Lib, l'application CDBVis ne permet pas encore de l'afficher, la connectivité interne des cartes frontales ne sera pas accessible à partir de CDBVis, par contre toute la connectivité macroscopique est bien représenté comme le montre les deux figures ci-dessous.

Sur le premier figure on visualise le premier chemin allant de la carte frontale jusqu'aux *HV Ch* et sur la deuxième figure le chemin reliant la carte frontale au *MUON Chamber*.

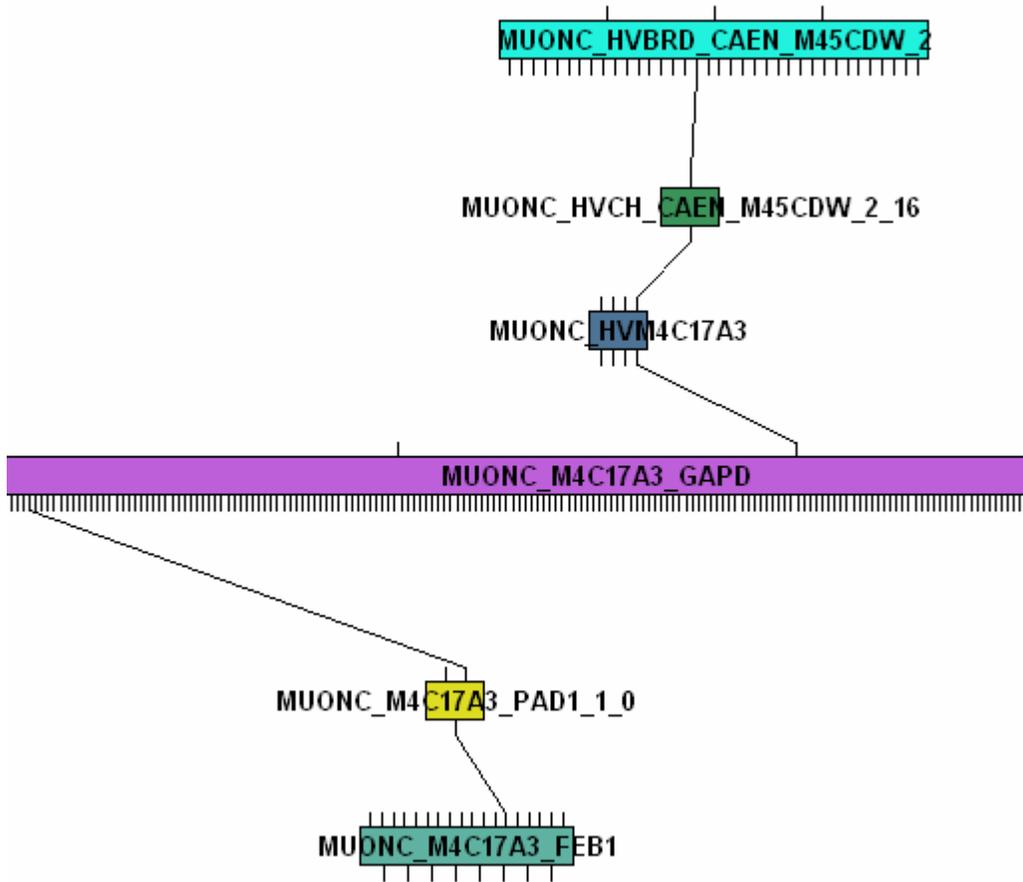


Figure 11 - Chemin reliant la carte frontale (Front End Board) à la HVBRD_CAEN (High Voltage Board)

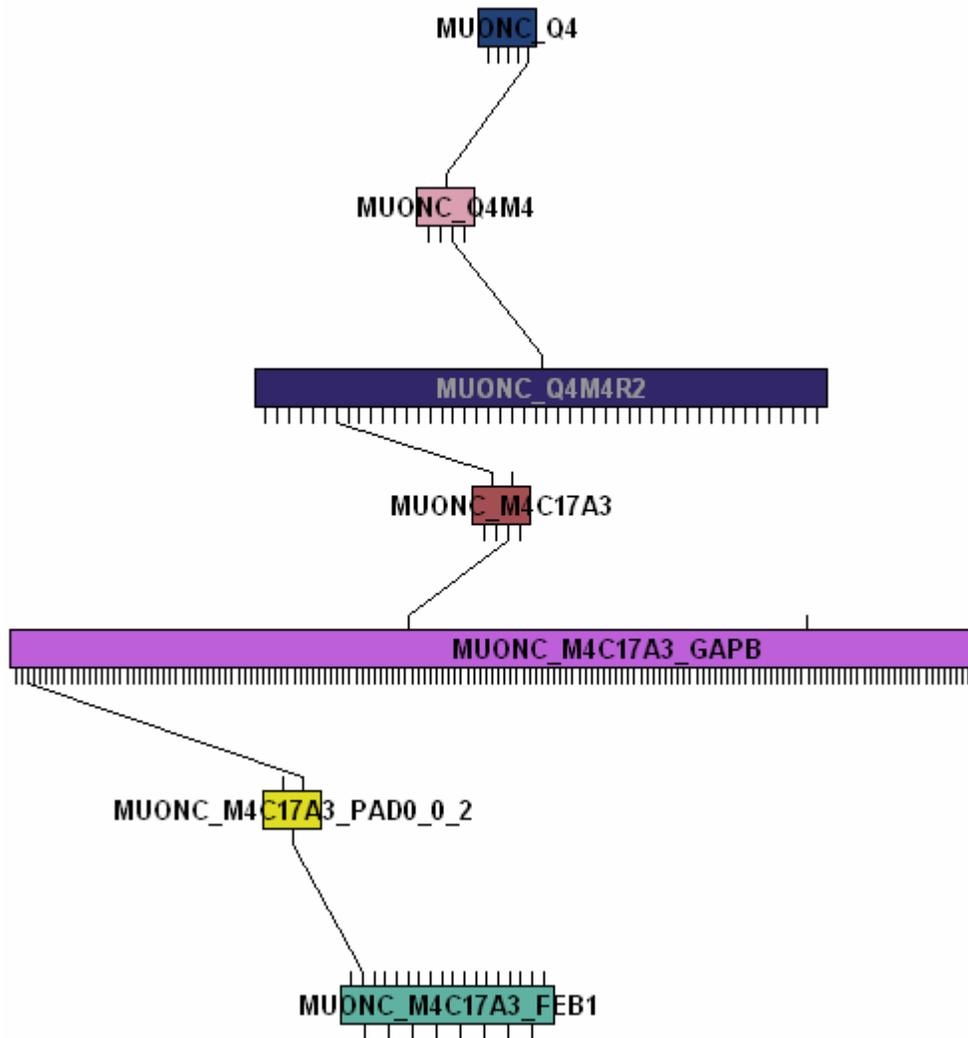


Figure 12 - chemin reliant la carte frontale aux différents composants du MUON Chamber

Une première version minimal de CDBVis a été développée durant l'été 2004, par l'étudiant Felix Schmidt-Eisenlohr, le travail a été ensuite poursuivi par un autre étudiant, Thomas Johansen, qui a ajouté plusieurs fonctionnalités aux programmes.

L'application CDBVis a été développée en utilisant le langage Python ainsi que la bibliothèque graphique wxPython pour les premières versions.

c) Python

Le langage Python a été choisi car il fait partie, avec le C et le C++, des langages communément utilisés au CERN. Python a été préféré au C et C++ pour sa simplicité qui rend le processus de développement beaucoup plus rapide. En effet en combinant une syntaxe très simple à un ensemble de structures de données évoluées (listes, dictionnaires,...), Python permet de produire des programmes à la fois très compacts et très lisibles. A fonctionnalités égales, un programme Python abondamment commenté est souvent de 3 à 5 fois plus court qu'un programme C ou

C++ équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.



Figure 13 - Logo de Python

De plus, Python est gratuit, portable sur la majorité des systèmes d'exploitation existants et intègre toute la puissance du modèle objet. Il dispose aussi d'un grand nombre de bibliothèques standards qui donnent accès à beaucoup de services et est connu pour son extensibilité qui permet de l'interfacer facilement avec des bibliothèques en C comme sera le cas pour l'application CDBVis avec la couche applicative de la base de données de configuration, la CIC DB Lib, qui est une bibliothèque de fonctions en C.

d) WxPython

wxPython est une implémentation libre en Python de l'interface de programmation wxWidgets qui est écrite en C++. wxPython met à disposition des développeurs un grand nombre de classes permettant de réaliser des interfaces homme machine (IHM) complètement indépendantes du système d'exploitation (OS) utilisé.



Figure 14 - Logo de wxPython

Pour la création et gestion des widgets, wxPython se base sur les routines natives du système d'exploitation sur lequel tourne l'application ce qui fait que les interfaces développées avec wxPython sont semblables à celles du système comme le montre les deux figure suivantes :



Figure 15 - Différence entre l'apparence de la même application sur deux OS différents (Windows XP et GNOME)

Toutefois, la version obtenue de CDBVis en utilisant wxPython souffrait d'un ensemble de problèmes qui ont rendu l'exploitation de l'éditeur et sa maintenance difficiles.

En effet, wxPython souffre de quelques fuites de mémoire (du moins sur la version 2.4.2 de la distribution linux utilisée au CERN).

De plus, wxPython ne fournit pas beaucoup de fonctions facilitant la manipulation des objets graphiques (dans notre cas les composants qu'on dessine) ; il fallait que l'application puisse gérer le déplacement et le redimensionnement des objets dessinés ce qui était à l'origine d'une grande perte en performances.

Le but de mon stage a donc été de porter l'application CDBVis sur une autre bibliothèque disposant de plus de fonctionnalités afin de venir à bout des problèmes apparus dans les versions précédentes.

5) Migration vers Qt

Pour remplacer wxWidget le choix s'est porté sur Qt qui est une bibliothèque C++ d'interfaces graphiques qui offre également beaucoup d'autres modules non graphiques : des composants d'accès aux données, de connexions réseaux, d'analyse XML, etc. Qt a été développée par la société Trolltech et est disponible en version libre pour Linux, Mac et pour WINDOWS à partir de sa version 4.



Figure 16 - Logo de la librairie Qt

Les principales raisons qui ont guidé vers le choix de Qt peuvent être résumés dans la liste suivante :

- A la différence de wxWidget qui est un projet communautaire, Qt semble beaucoup plus professionnelle car il y a une société derrière, ce qui assure un support et une évolution continues pour la bibliothèque. Ceci se reflète aussi au niveau de la documentation ; Qt dispose d'une documentation très bien fournie par rapport à celle de wxWidget.
- Qt fournit aussi des outils puissants permettant de dessiner rapidement ses interfaces et de générer aussi le code correspondant.
- Les modules de dessin de Qt sont très complets et faciles à manipuler.
- Qt dispose d'un puissant système de gestion d'événements et a ses propres routines d'affichage ; elle ne s'appuie pas donc sur ce qui est fourni par le système d'exploitation comme est le cas pour wxWidget ce qui représente un grand gain en performances.

a) Gestion des événements

Le système de gestion des événements de Qt est basé sur le mécanisme de signaux /slots.

Un signal est émis par un composant de l'application lorsqu'un événement se produit.

Un événement est généralement le fait d'un utilisateur qui clique par exemple sur un bouton ou qui remplit un champ. Les slots sont des fonctions appelées en réponse à un signal particulier.

La figure suivante illustre la puissance de ce mécanisme. En effet, en définissant pour chaque objet un ensemble de signaux et de slots on peut ensuite connecter les signaux aux slots des différents objets. L'avantage est qu'on peut connecter un seul signal à plusieurs slots, ce qui fait qu'un seul événement reçu de l'utilisateur peut déclencher plusieurs actions ce qui est impossible à réaliser avec les autres bibliothèques.

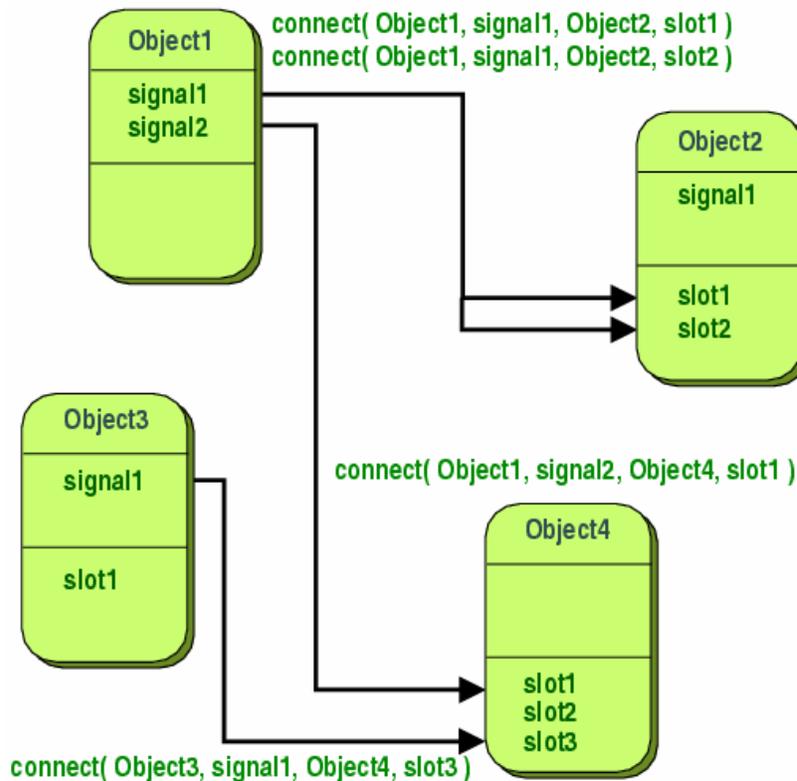


Figure 17 - Mécanisme de signaux/slots dans Qt

b) L'outil Qt Designer

Qt Designer est un Environnement de Développement Intégré à Qt qui permet de concevoir graphiquement une interface.

Comme le montre la figure suivante, le Qt Designer est composé d'un ensemble de boîtes à outils faciles à utiliser et très pratique :

- La barre vertical de gauche permet de sélectionner un composant graphique (bouton, ligne de texte, label de texte, ...), puis de le placer sur la fenêtre central qui représente l'interface qu'on veut créer.
- Le panel « Object Explorer » donne une vision hiérarchique des composants et de leurs classes.
- Le panel « Property » permet d'éditer les propriétés de tous les composants de notre interface.
- « Project overview » présente l'arborescence des fichiers utilisés dans le projet.

Une fois l'interface statique créée, le Designer permet d'enregistrer le résultat sous forme d'un fichier qui porte l'extension « .ui » et qui contient toute les informations sur l'interface dans un format XML.

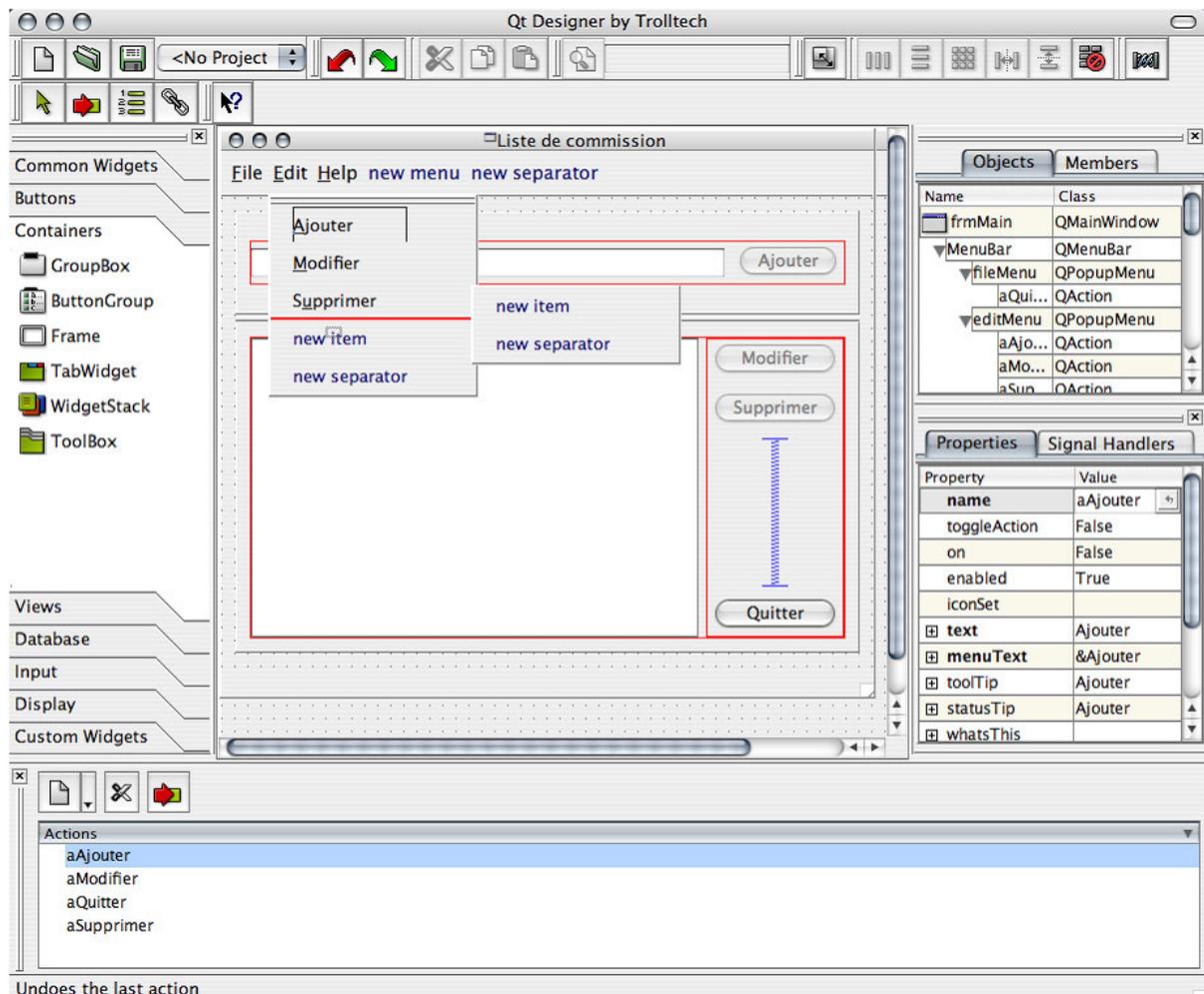


Figure 18 – Vue d'ensemble de l'outil Qt Designer

Le Qt Designer permet aussi de gérer les signaux slots comme le montre la figure suivante où l'on peut définir l'objet envoyant le signal et celui qui le reçoit (colonnes « sender » et « receiver ») ainsi que le signal et le slot mis en jeu.

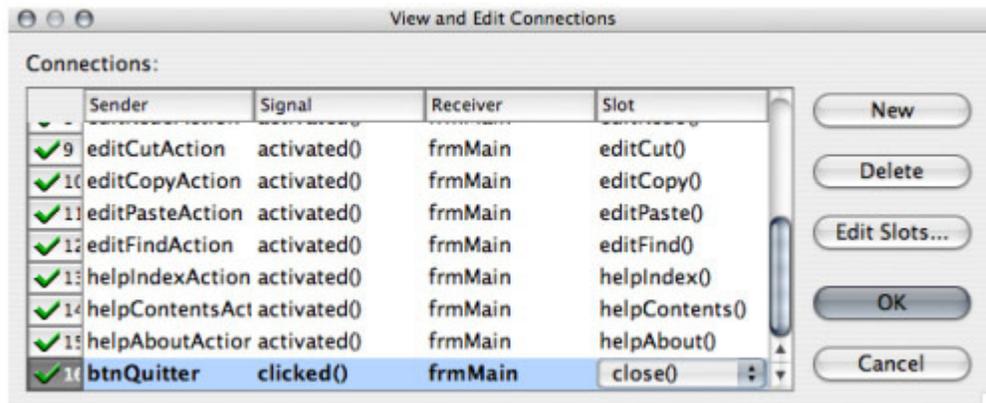


Figure 19 - Création de connexion signaux/slots dans Qt Designer

c) Génération de code : Le Compilateur d'Interfaces Utilisateur (uic).

Qt dispose aussi d'un outil appelé uic (*User Interface Compiler*) qui permet de traduire le fichier XML obtenu à partir du Designer en code C++.

Comme l'application CDBVis est développée en Python, nous utiliserons pyuic qui est un utilitaire équivalent à uic mais qui permet de générer du code Python.

Le processus de développement adopté peut donc se résumer dans le schéma suivant.

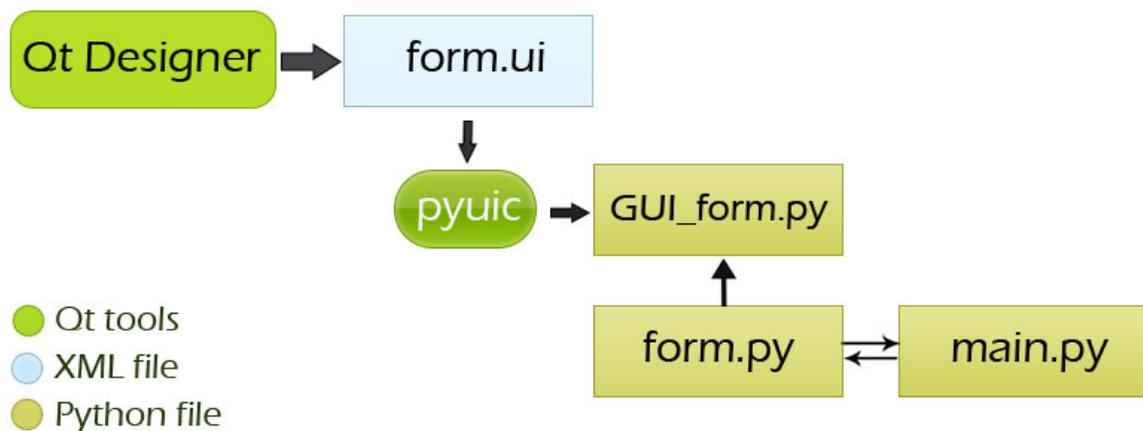


Figure 20 - processus de développement en utilisant les outils de Qt

Tout d'abord nous utilisons le Qt Designer pour définir l'apparence de l'interface, ensuite à l'aide de pyuic nous obtenons une classe Python que nous allons dériver pour y ajouter nos propres méthodes avant de l'utiliser dans l'application principale.

Réalisation de la solution

1) Définition de l'environnement de travail

La première tâche que j'ai eu à été de définir un environnement de travail avant de pouvoir commencer le portage de l'application.

La figure suivante illustre bien les différents composants mis en jeu pour le développement de l'application.

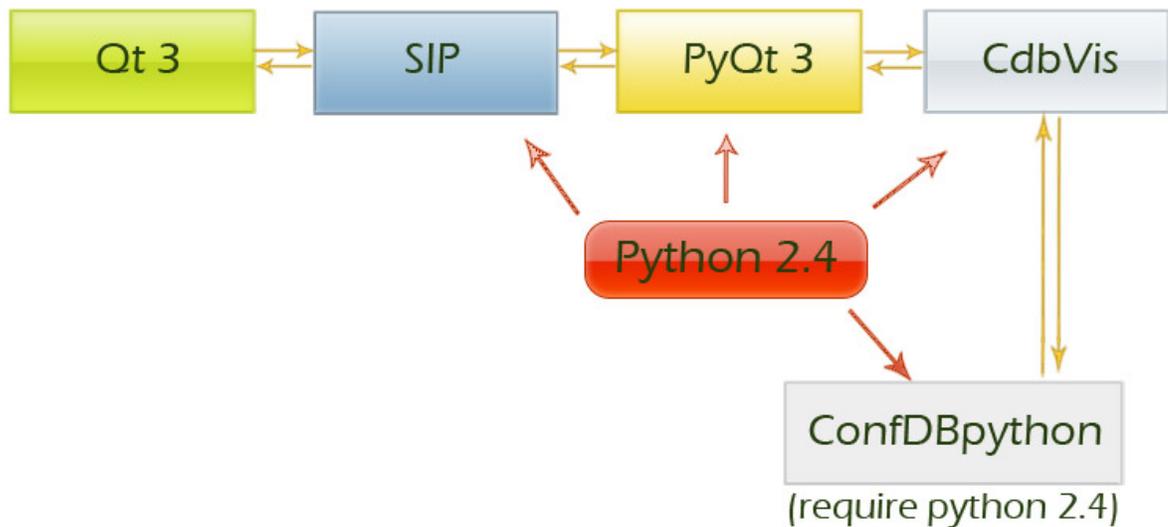


Figure 21 – L'environnement de l'application CDBVis

Qt étant une bibliothèque C++, nous ne pouvons l'utiliser directement à partir de CDBVis, il faut donc installer PyQt qui est une adaptation des classes C++ de Qt en classes Python.

PyQt a recours à l'outil SIP qui permet d'importer des classes C et C++ dans Python.

Le mieux aurait été d'utiliser la version 4 de Qt, mais l'utilisation de Qt 4, nécessite d'avoir PyQt 4 qui n'est compatible qu'avec la version 2.5 de Python. Or CDBVis fait aussi appel à la bibliothèque ConfDBpython qui contient la couche applicative permettant de se connecter à la base de données et qui n'est pas encore compatible avec Python 2.5.

L'application CDBVis doit donc tourner, pour le moment, avec la version 2.4 de python. Dans ce cas on ne pourra plus utiliser la version 4 de Qt est on doit se contenter de Qt v3.

Le problème avec Qt 3 est que cette version n'est pas disponible en version libre sous WINDOWS. Il restait alors la solution de se procurer une licence payante pour pouvoir utiliser Qt 3 sur WINDOWS. Cette solution n'a pas été adoptée car je suis parvenu, moyennant quelques recherches, à trouver une adaptation non officielle pour WINDOWS de la version libre de Qt sous linux. Cette version non officielle de

Qt a été réalisée au sein du projet QTWIN qui visait à mettre à disposition Qt en version libre sous WINDOWS. A l'heure actuelle, ce projet a été abandonné car Trolltech a décidé, à partir de la version 4 de Qt, de fournir une version libre sous WINDOWS.

L'inconvénient avec l'utilisation de la version non officielle de Qt vient lors de l'installation de celle-ci. Alors que la version commerciale de Qt est fourni avec un installateur qui permet d'installer l'environnement en quelques minutes, la version non officielle nécessite de recompiler Qt sur sa machine ce qui prend facilement une demi heure sur une machine disposant de 2 GHz de processeur et 1 Go de RAM.

Un descriptif des instructions d'installation figure en annexe I.

2) Architecture de l'application CDBVis : UML, différentes classes

Une fois l'environnement de travail défini (Python 2.4 avec Qt 3), vient la deuxième étape de mon travail qui est de porter l'application CDBVis pour n'utiliser que la librairie Qt.

Pour avoir une idée de la quantité de travail que ce portage constitue, il est utile de rappeler comment CDBVis est architecturée.

Les sources de l'application sont constituées d'un ensemble de classe réparties en 16 fichiers différents. Le plus petit de ces fichiers fait une centaine de lignes alors que le plus grands fait 5000 miles lignes de codes. L'application fait au total un peu plus de 16 500 lignes de code où sont entremêlées les parties graphiques qui font appel à wxPython et les parties de traitement de données qui sont non graphiques. Il va falloir donc identifier tous les appels à des fonctions de la librairie wxPython pour les remplacer par des fonctions équivalentes de Qt si équivalence il y a.

La figure suivante montre un diagramme UML simplifié de l'architecture de l'application en ne prenant en compte que les principales classes.

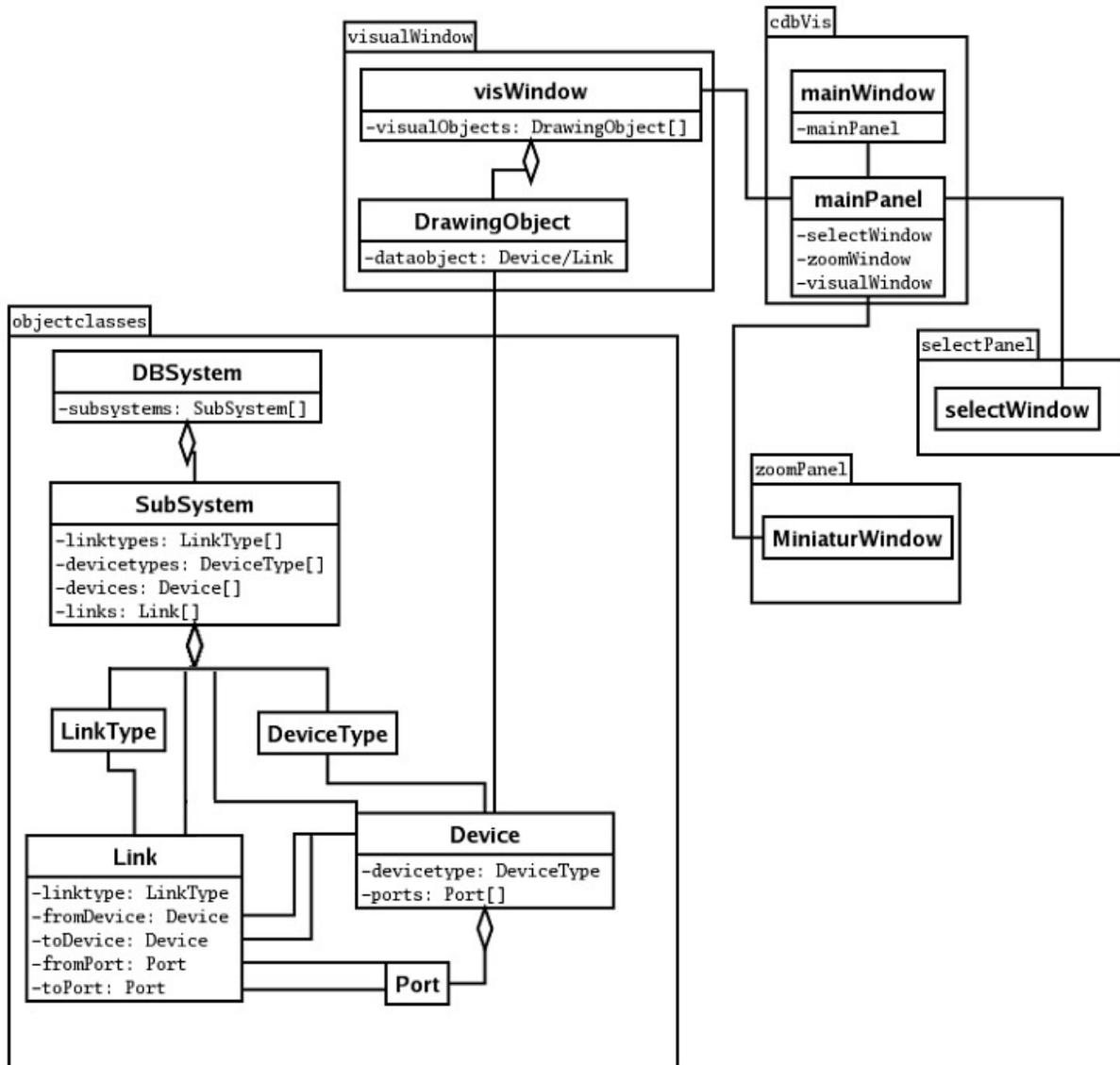


Figure 22 - Diagramme UML de la l'application CDBVis

Dans le diagramme ci-dessus on peut distinguer les trois fichiers principaux qui constituent l'application :

objectclasses.py : contient le modèle objet permettant d'encapsuler les données récupérées de la base de données.

visWindow.py : contient les classes relatives à la partie centrale de l'application ou seront affichés les composants du système qui sont représentés par la classe : DrawindObject.

cdbVis.py : contient la fenêtre principale de l'application qui contient les différents menus et barres d'outils.

3) Démarche suivie pour le portage de l'application

a) Identification des classes à modifier

La première étape avant de commencer le portage de l'application a consisté à déterminer les principaux fichiers qui devraient être modifiés. Car l'idée a été de porter l'application petit à petit en gardant une version qui marche de l'application tout au long du portage.

Une première analyse du code source de l'application permet d'éliminer les deux fichiers `objectclasses.py` et `cdbVisCore.py` car ils ne font pas appel à la librairie wxPython. Aucune modification ne sera donc apportée à ces deux fichiers.

Le fichier `cdbVisCore.py` contient les paramètres intrinsèques de l'application (version de l'application, épaisseur par défaut des traits représentés, codes associés aux différentes variables globales...).

b) Utilisation d'interfaces

Une des méthodes utilisées au début pour porter l'application sur le nouvel environnement est d'avoir recours à des interfaces de communication qui se chargent de traduire les appels vers des fonctions de wxPython en des appels vers des fonctions de Qt.

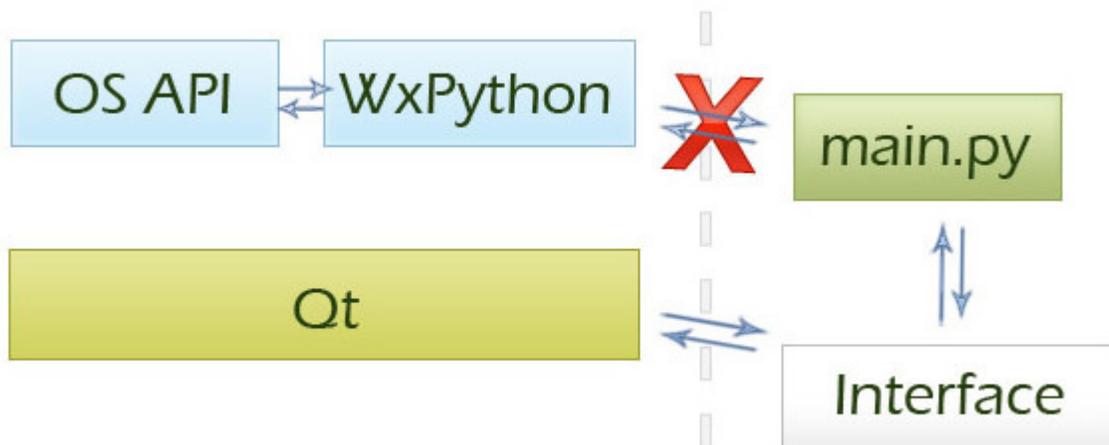


Figure 23 - Utilisation des interfaces de communication

L'idée est de fournir à notre application des objets semblables à ceux qu'elle a l'habitude d'utiliser dans wxPython mais qui s'appuieront cette fois sur les méthodes de la nouvelle bibliothèque. Prenons un exemple très simple pour comprendre le principe de l'utilisation de telles interfaces de communication. La librairie wxPython utilise la classe `wxSize` pour représenter un couple formé d'une longueur et d'une largeur. Un objet de type `wxSize` dispose de deux méthodes : `wxSize::GetWidth` et `wxSize::GetHeight` qui retournent respectivement la largeur et la hauteur de l'objet. La classe équivalente à `wxSize` dans Qt s'appelle `QSize` et les méthodes dont elle dispose sont : `QSize::width` et `QSize::height`.

L'idée est donc de créer une classe qu'on appellera `wxSize` mais qui s'appuie sur la classe `QSize` de Qt. Cette nouvelle classe devra fournir les méthodes `GetWidth` et `GetHeight` qui ne feront qu'appeler les fonctions équivalentes dans Qt à savoir : `QSize::width` et `QSize::height`.

Ainsi, par l'ajout d'une simple classe, on arrive à simuler une classe de wxPython et on peut s'épargner la fastidieuse tâche de rechercher toutes les fois où on utilise la classe spécifique à wxPython pour les remplacer par la classe équivalente dans Qt.

Bien qu'elle donne des résultats rapides et sans grands efforts, cette méthode est à proscrire pour le portage d'une application car on se retrouve rapidement avec beaucoup de nouvelles classes ce qui rend la détection des erreurs plus difficile car on confond rapidement les classes de la première bibliothèque avec celles que nous avons nous même créé.

Toutefois l'utilisation de cette technique s'est avérée très utile pour déterminer si les deux classes qu'on considère comme équivalentes ont le même comportement, on peut donc rapidement créer une telle interface de communication pour faire des tests avant de remplacer proprement les classes d'une bibliothèque par leurs équivalents dans la deuxième bibliothèque.

c) Partie visuelle : Canevas

Dans bien des cas, les deux bibliothèques ont des conceptions différentes des composants graphiques, il n'est pas toujours aisé de trouver de parfaites équivalences d'où la nécessité de reprendre entièrement certaines classes pour les adapter à la vision de telle ou telle bibliothèque.

Ceci a été le cas par exemple pour la classe visWindow qui représente la partie centrale de CDBVis où sont représentés les différents composants ainsi que leur connectivité.

En effet, dans la version de CDBVis utilisant la librairie wxPython, la partie de visualisation a été conçue comme indiqué dans la figure suivante.

Comme le nombre et la taille des composants à afficher peut considérablement varier, il est nécessaire de disposer d'une assez grande zone pour le dessin des composants. L'utilisateur a donc une petite vue sur cette grande zone et il peut naviguer dedans en utilisant les barres de défilement.

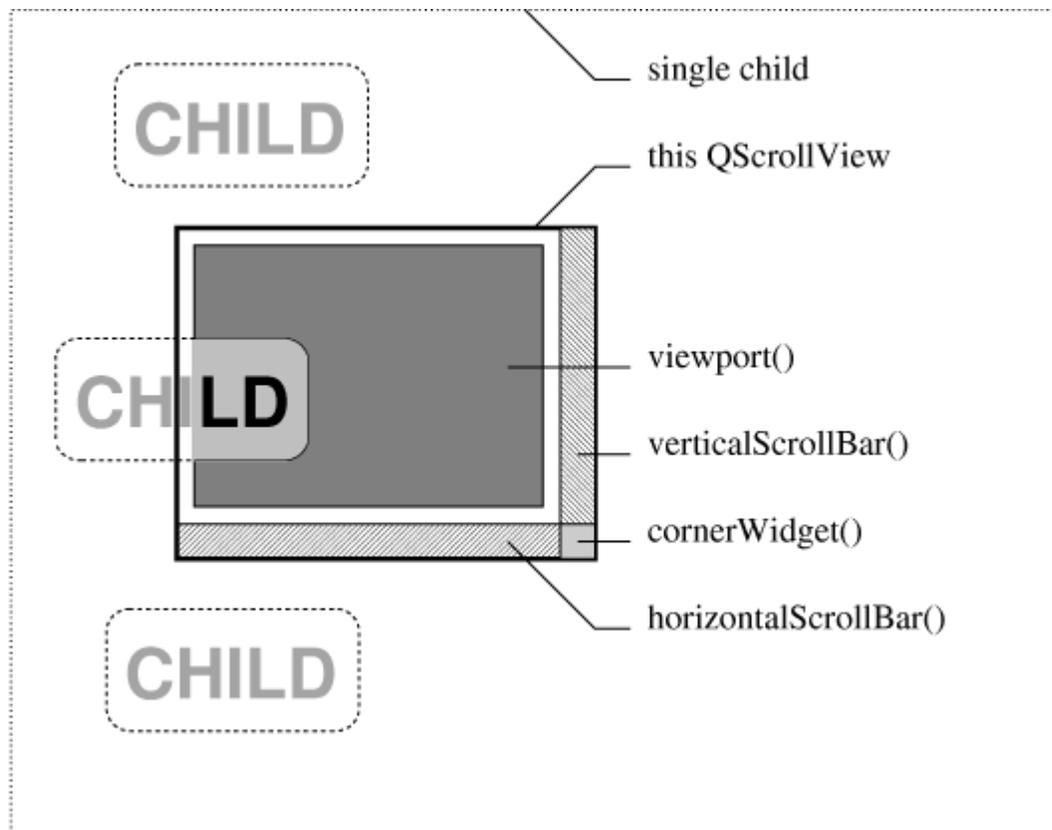


Figure 24 - Exemple d'une zone de défilement

Dans la figure ci-dessus, la grande zone de dessin correspond au grand rectangle en pointillés (*single child*). Les composants (*child*) viennent ensuite s'ajouter sur ce grand cadre et on a une vue de l'ensemble à travers le petit cadre grisé car la taille de l'écran ne peut pas visualiser la totalité de la zone de dessin en même temps. Il est possible de naviguer dans le grand cadre pour voir les différents composants en utilisant les barres de défilements (*horizontal ScrollBar* et *vertical ScrollBar*).

De son côté, Qt dispose d'un module spécial qui s'adapte parfaitement à notre cas. Ce module s'appelle le *Canvas* et il permet de visualiser un grand nombre de composants graphiques sur un grand cadre comme le montre la figure suivante :

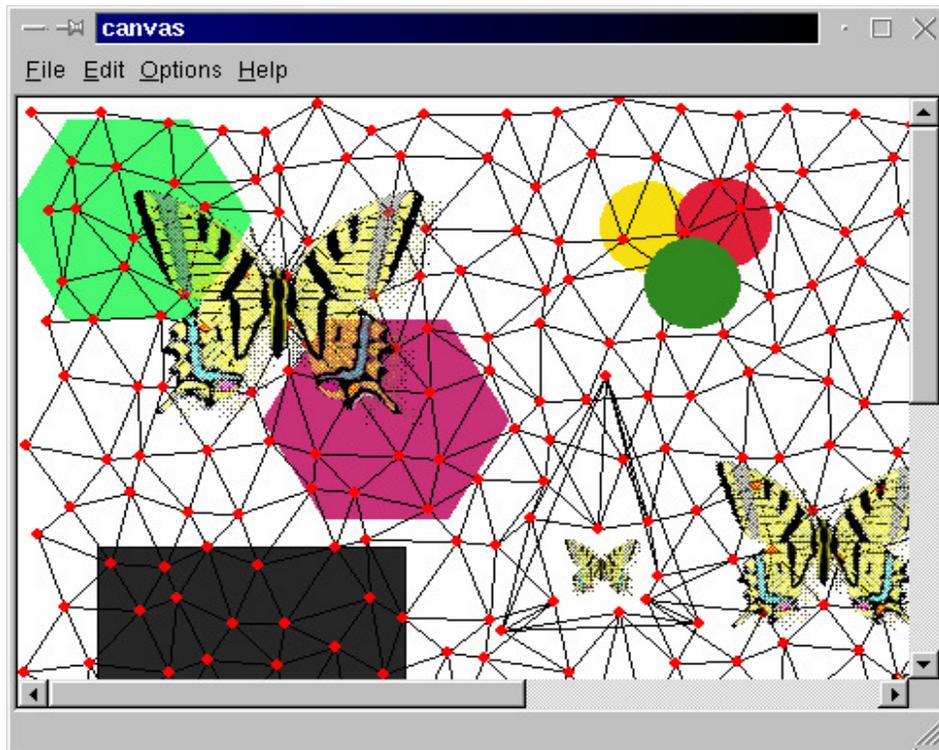


Figure 25 - Exemple d'une application utilisant le Canvas module de Qt

Le module *Canvas* de Qt dispose de suffisamment de classes et de méthodes pour rendre l'utilisation des composants graphiques et leur manipulation chose facile. En effet, sur le Canevas on peut ajouter des éléments (*Canvas items*) qui peuvent prendre différentes formes (ellipse, rectangle, ligne, polygone...). On peut aussi créer des formes composées.

Chacun de ces éléments dispose d'un ensemble de méthode facilitant sa manipulation. Par exemple pour déplacer un élément il suffit d'appeler la méthode **moveBy(int x, int y)** pour déplacer notre élément d'une distance *x* horizontalement et *y* verticalement alors que sans le module *Canvas* il aurait fallu calculer la nouvelle position de notre élément, effacer la première occurrence de l'élément et le redessiner à sa nouvelle position.

Gestion des collisions:

Le module Canvas de Qt permet aussi de gérer les collisions entre composants. Par exemple dans le cas où on se retrouve avec plusieurs éléments qui se chevauchent, un simple appel à la fonction `QCanvas::collisions(QPoint& Pt)` permet de retrouver la liste de tous les éléments se trouvant au point **Pt** même si ils sont masqués par d'autres éléments.

En plus des deux coordonnées, *x* et *y*, permettant de positionner le composant dans notre cadre, Qt ajoute une troisième coordonnée, *z*, permettant de déterminer le plan sur lequel se trouve le composant. Ainsi les éléments qui ont le *z* le plus grand sont ceux du premier plan et ceux avec un petit *z* sont ceux au fond de la scène.

Avec ce mécanisme il devient facile de déterminer l'élément à sélectionner après un clic de l'utilisateur car il suffit de rechercher dans la liste des éléments retournés par la fonction `QCanvas::collisions` celui qui a le `z` le plus grand.

Matrices de transformations:

Le module `Canvas` dispose aussi d'un puissant système de transformations 2D. En effet, il suffit de choisir la matrice 2D correspondant à la transformation souhaitée (rotation, translation, agrandissement...) et le module `Canvas` se charge d'appliquer cette transformation à tous les éléments constituant la partie visuelle.

J'ai utilisé ces transformations 2D lors de la gestion du zoom, car la version wxPython de `CDBVis` devait pour effectuer un zoom, redimensionner tous les éléments un par un alors qu'avec `Qt` il est possible de leur appliquer une même transformation en même temps.

d) Séparer le contenu de la présentation

Dans un souci de mieux organiser les sources de l'application, j'ai essayé tant que possible de séparer les parties du code responsables de tout ce qui est purement graphique (création des boutons, placements des éléments sur l'interface...) de la partie qui s'occupe du traitement des données. Bien qu'une séparation parfaite ne soit pas possible j'ai essayé tout du moins de séparer la partie du code générée par le `Qt Designer` du reste du code source de l'application.

Pour cela il a fallu reprendre toutes les interfaces et formulaires utilisant wxPython pour les redessiner en utilisant le `Qt Designer`. J'ai choisi ensuite de préfixer les noms de tous les fichiers sources générés à partir du `Designer` par le mot `GUI` pour (Graphical User Interface). Il suffit ensuite de dériver la classe obtenue pour y ajouter nos propres fonctions comme le montre la figure suivante.

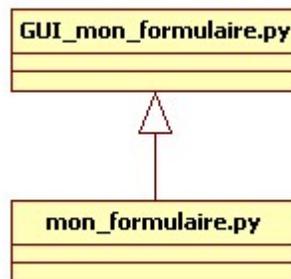


Figure 26 - Séparation entre la partie graphique du code et la partie non graphique

Certes, avec cette méthode nous avons deux fois plus de fichiers et de classes dans nos sources mais elle permet une meilleure organisation du code ; on sait quel fichier il faut modifier selon qu'on veuille modifier l'apparence de notre interface ou modifier nos propres fonctions de traitement des données. De plus, la modification de l'apparence de notre interface devient une tâche facile car on peut à tout moment

reprendre l'interface avec le Designer, y apporter les modifications qu'on veut avant de générer un nouveau fichier *GUI_mon_formulaire.py* qui écrasera le premier fichier.

En reprenant les différentes interfaces de l'application je leurs ai associé des *Layouts* qui permettent de repositionner les éléments de l'interface dans le cas ou la fenêtre principale est redimensionné de sorte à ce qu'ils prennent la totalité de l'espace disponible. Le deux figures suivante montrent la même interface avec et sans *layouts*.

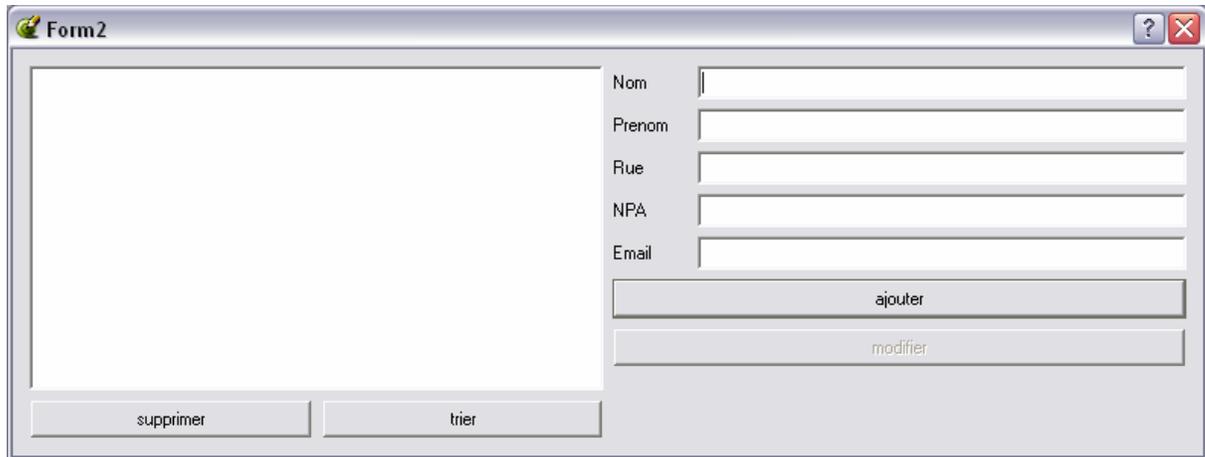


Figure 27 - Interface utilisant des Layouts

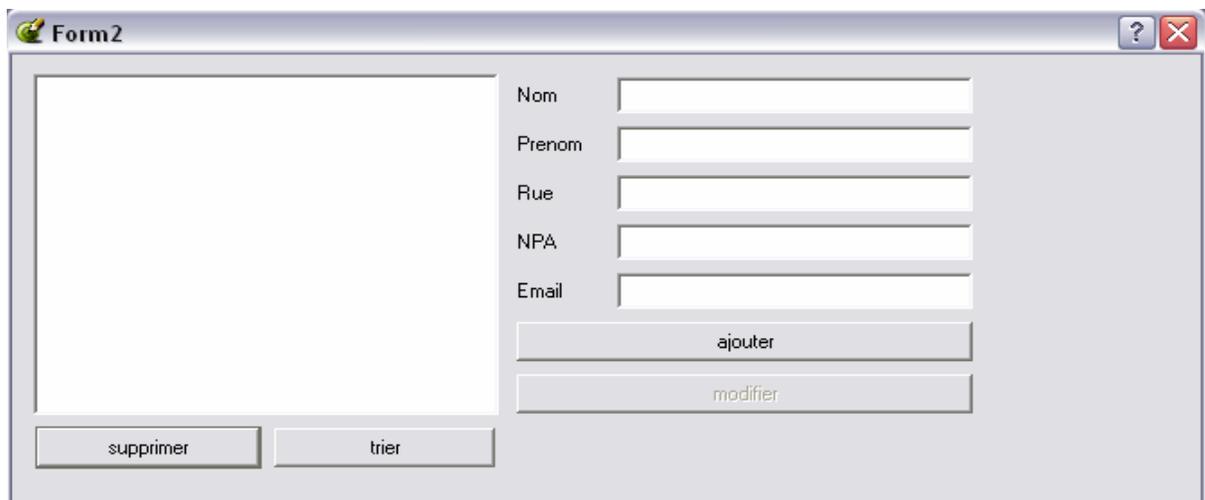


Figure 28 - Interface réalisé sans Layouts

4) Affichage des grands composants

Certains composants de l'expérience possèdent un grand nombre de connexions ce qui rend leur visualisation très difficile. Par exemple dans le cas d'un routeur du système d'acquisition de données il faut un peu plus d'une minute pour que l'application puisse afficher tous les composants qui lui sont reliés.

Et même après un aussi long temps d'attente, le résultat obtenu n'est pas très satisfaisant car, comme on peut le voir sur la figure suivante, on se retrouve avec un

grand nombre de composants avec des connexions dans tous les sens ce qui détériore la lisibilité et réduit notablement les performances de l'application.

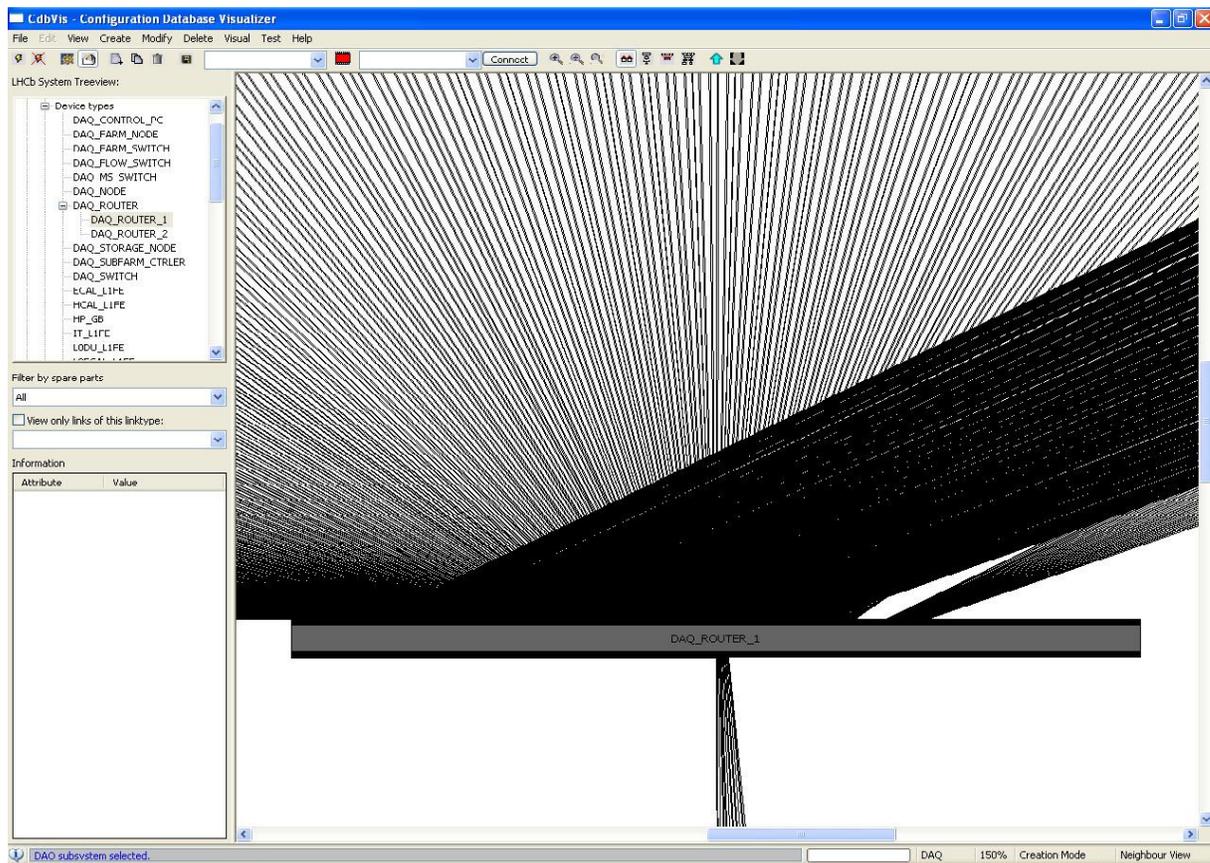


Figure 29 - Représentation du routeur du DAQ avec toutes ses connexions

J'ai donc choisi de fixer un nombre maximum de connexions au delà duquel l'application affiche une nouvelle fenêtre contenant la liste de tous les composants reliés au composant central. L'utilisateur peut ensuite choisir le composant qu'il veut visualiser en cliquant sur son nom comme le montre la figure suivante ou on a choisit de visualiser le *DAQ_SWITCH_01* relié à l'entrée du routeur du DAQ. Ainsi on laisse à l'utilisateur le choix de visualiser les informations qu'il souhaite sans surcharger l'application.

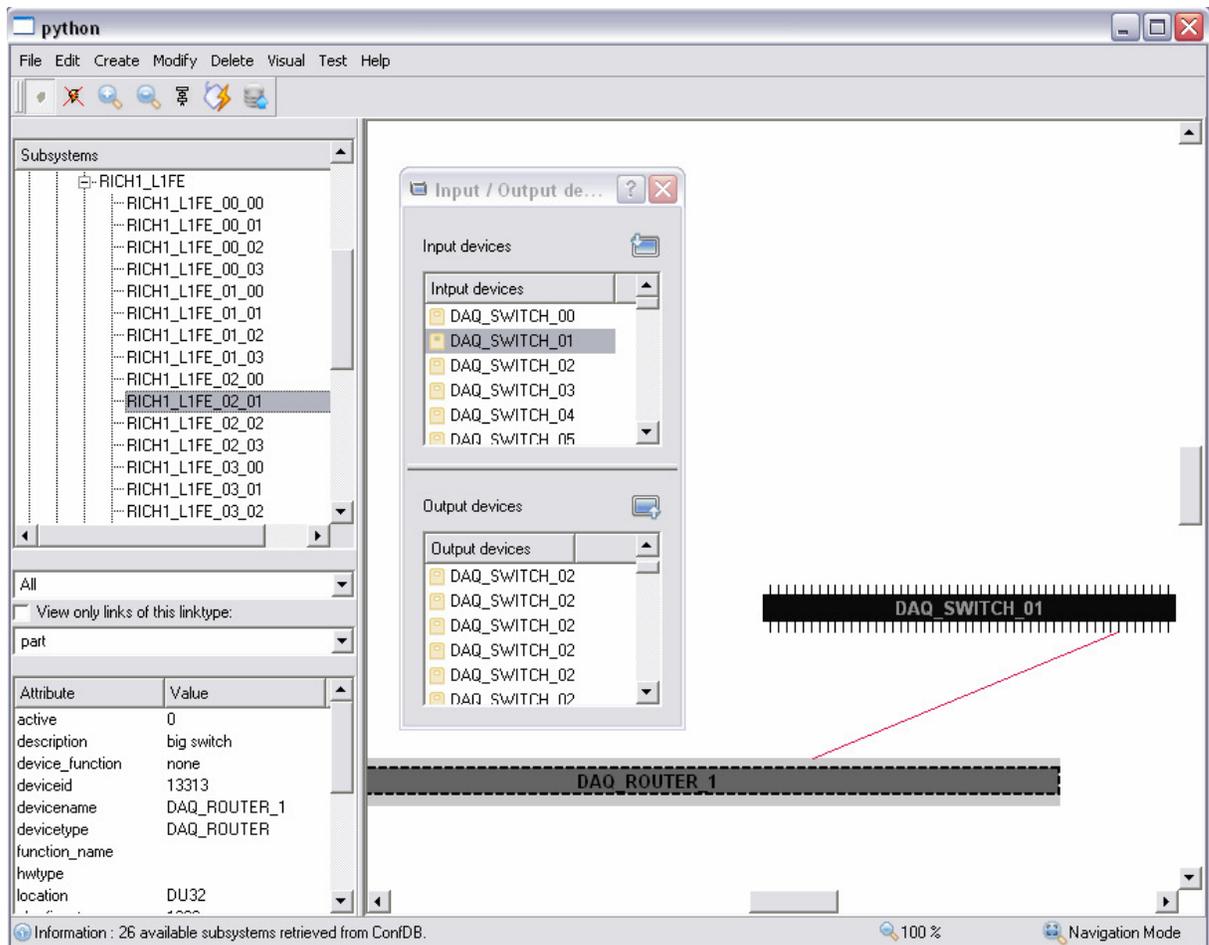


Figure 30 - visualisation des grands composants (ici le DAQ_ROUTER_1)

5) Mise à disposition des utilisateurs

Pour faire fonctionner l'application CDBVis, il faut disposer de python 2.4, Qt v3, SIP, PyQt v3 en plus d'un accès au système de fichiers distribué AFS (Andrew File System) qui permet à un ensemble de machines réparties en réseau de partager des fichiers de façon cohérente.

Il serait contraignant pour l'utilisateur de devoir installer tout l'environnement comme indiqué dans les parties précédentes. C'est pourquoi j'ai utilisé l'extension **py2exe** de Python qui convertit les scripts Python (.py) en exécutables Windows (.exe).

Py2exe permet aussi de rassembler dans un même dossier toutes les bibliothèques dont a besoin l'application pour fonctionner correctement. Ainsi on arrive à obtenir un dossier comportant l'exécutable de l'application accompagné de plusieurs bibliothèques dynamiques (aussi appelée dll pour *Dynamic Link Library*) représentant les différentes bibliothèques appelées.

L'utilisateur peut ainsi utiliser l'application CDBVis sans avoir à installer Qt ni même Python. Tout ce dont il a besoin c'est d'avoir un accès à AFS.

L'archive réalisé fait 7 Mo environ et est téléchargeable sur la page réservée à l'application sur le site du LHCb : <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/cdbVis.htm>

Résultats et discussions

1) Présentation générale de CDBVis

Dans cette partie je vais présenter l'application et ses différentes fonctionnalités après son portage vers Qt.

L'application CDBVis peut fonctionner dans deux modes différents : un mode de navigation ou on peut afficher les données de la CIC DB et un mode de création ou on peut modifier les propriétés des composants existants et en créer des nouveaux.

Il n'y a pas de séparation entre les deux modes et on passe du mode de navigation à celui de la création automatiquement ; il suffit pour cela de créer un nouveau composant ou d'en modifier un déjà existant.

L'application se comporte toutefois de manière différente selon qu'on est dans l'un des deux modes ; dans le mode de navigation toutes les informations qu'on visualise sont celle de la CIC DB alors que dans le mode de création on voit les composants avec les modifications qu'on leur a apporté. Par exemple si on modifie la localisation d'un composant on passe automatiquement en mode de création. Et toute les prochaines fois ou on demandera à l'application d'afficher la localisation de ce composant nous verrons apparaitre la nouvelle localisation et non celle actuellement présente dans la base de données. Pour revenir au mode de navigation il faut propager les changements effectués localement à la base de données.

a) Vue Principale

L'application CDBVis est composée de trois zones principales : la plus grande sert pour visualiser les composants et leur connectivité. Une zone de sélection affiche la liste de tous les composants de la CIC DB sous forme d'un arbre. La dernière zone sert à afficher les informations relatives à l'objet actif.

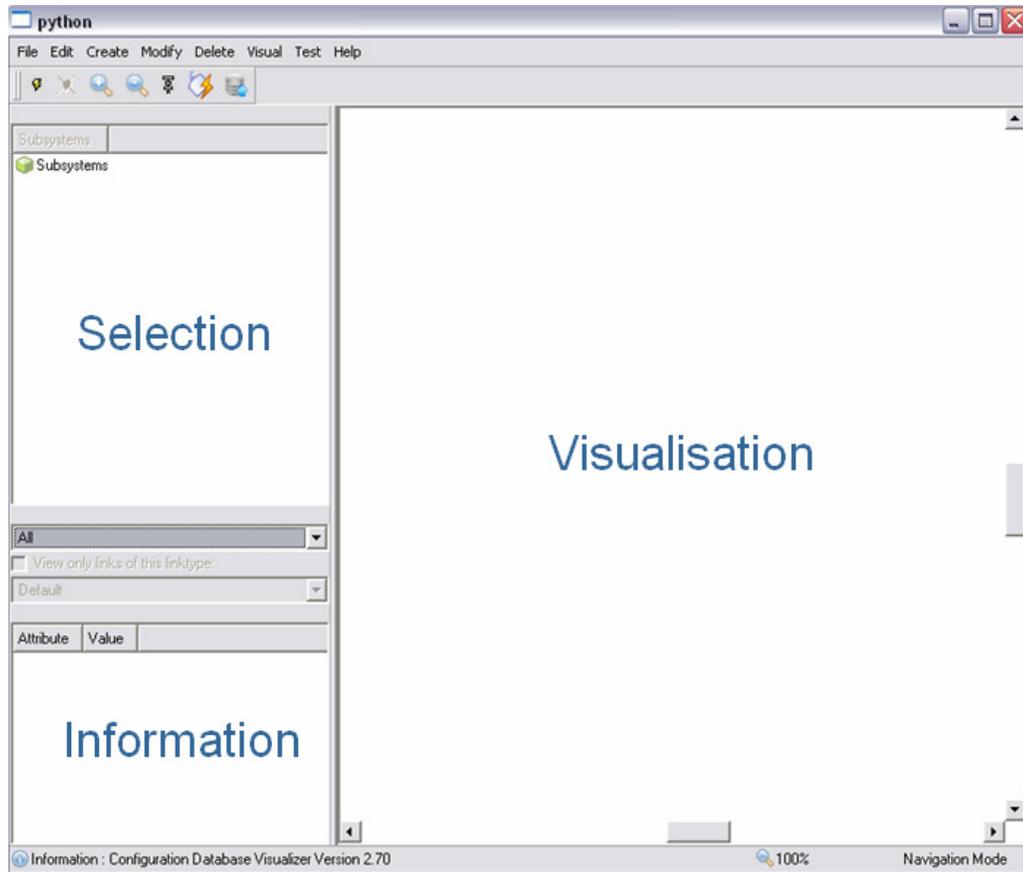


Figure 31 - Vue d'ensemble de l'application CDBVis

b) Zone de sélection

La zone de sélection contient un arbre où sont listés tous les composants des différents systèmes. Les données sont organisées en niveau ; chaque niveau de l'arbre contient un type de données différent. Le premier niveau contient la liste des systèmes, pour chaque système on définit une liste des types de composants et une autre liste pour les types de connexions sur le deuxième niveau (voir deuxième vue de la figure suivante).

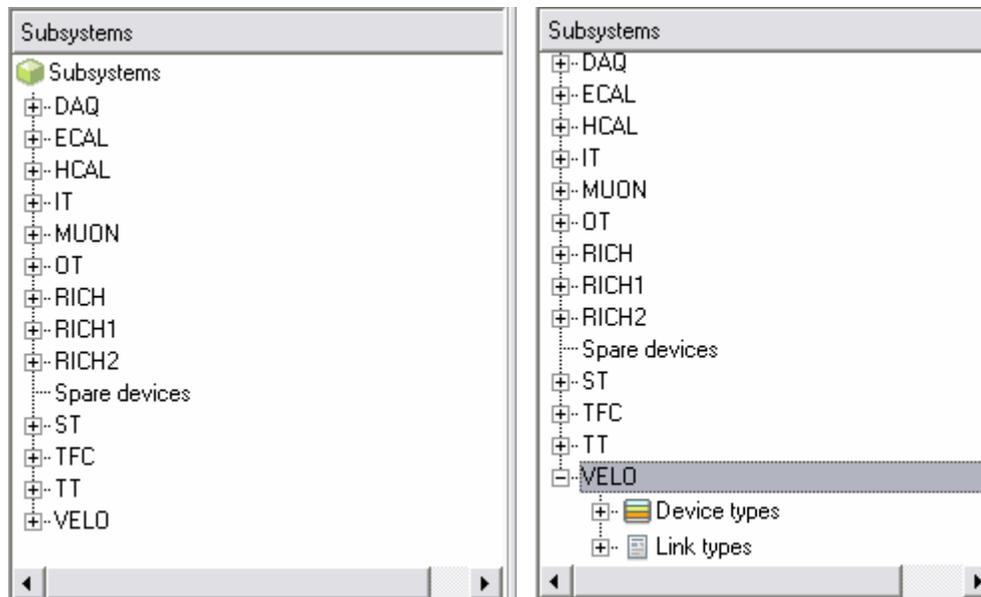


Figure 32 - Deux vues de la zone de sélection montrant les deux premiers niveaux de l'arbre

Au troisième niveau on retrouve la liste des types de composants (ou types de connexions), et dans chaque type sont listés les composants en faisant partie. Au dernier niveau figure la liste des ports se rattachant aux différents composants comme on peut le voir sur la figure suivante.

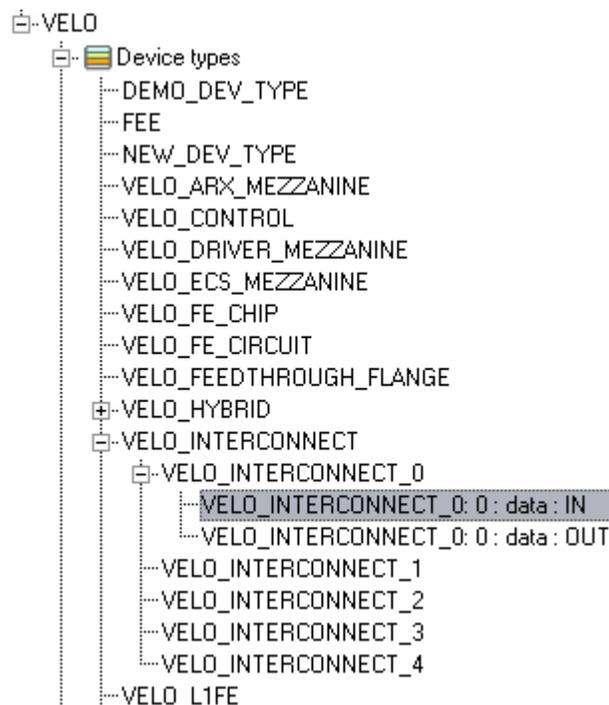


Figure 33 - Les trois derniers niveaux (types de composants, composants, ports) de l'arbre contenant tous les composants

c) Zone d'information

La zone d'information montre toutes les informations concernant l'élément actif. Le type de ces informations dépend du type de l'objet actif (composant ou connexion).

Attribute	Value
description	VELO_HYBRID
device_function	none
deviceid	1287487
devicename	VELO_HYBRID_R_0
devicetype	VELO_HYBRID
function_name	
hwtype	
location	VL01L_R_UPSTREAM
nbrofinput	0
nbrofoutput	4
node	1
nodeused	1
promiscuous_mode	0
responsible	
rgbcolor	(0, 128, 0)
serialnb	4TVLIVHYB00001
url	http://velodb.ph.liv.ac.uk/lhcb/moduledata
user_comments	

Attribute	Value
bidirectional_link_used	0
link_info	none
link_type	mixed_data
linkid	44443
lkused	1
node_from	VELO_HYBRID_R_0
node_to	VELO_SHORT_KAPTON_3
port_nbrfrom	3
port_nbrto	0
port_typefrom	data
port_typeto	data
rgbcolor	(104, 236, 27)

Figure 34 - Deux vues de la zone d'information ; dans la première vue un composant est sélectionné et dans la deuxième il s'agit d'une connexion.

d) Barre d'état

La barre d'état est divisée en trois parties, la première sert pour l'affichage de tout type de message visant à refléter l'état de l'application (Informations, Erreurs, avertissement, succès ou échec d'une opération...). Les messages importants (erreurs critiques, erreurs d'insertion dans la base de données...) sont affichés en plus sous forme d'une boîte de dialogue pour être plus apparents.

Les deux autres parties de la barre d'état montrent le zoom appliqué sur la zone de visualisation et le mode dans lequel se trouve l'application (navigation ou création)



Figure 35 - Barre de status de l'application

2) Mode de Navigation

L'application CDBVis permet d'aborder la CIC DB de deux manières différents ; pour chaque composant on peut choisir de s'intéresser à ses voisins ou bien aux chemins qui y passent. Il est possible de passer d'un mode de visualisation à l'autre par un simple clique.

a) Visualisation des voisins

Ce mode de visualisation permet de voir le composant central et ses voisins qui lui sont directement connectés.

Dans la figure suivante, le composant central est le *VELO_HYBRID_R_0*, on ne peut donc voir que les composants qui lui sont directement liés (ici les quatre *SHORT_KAPTONs*).

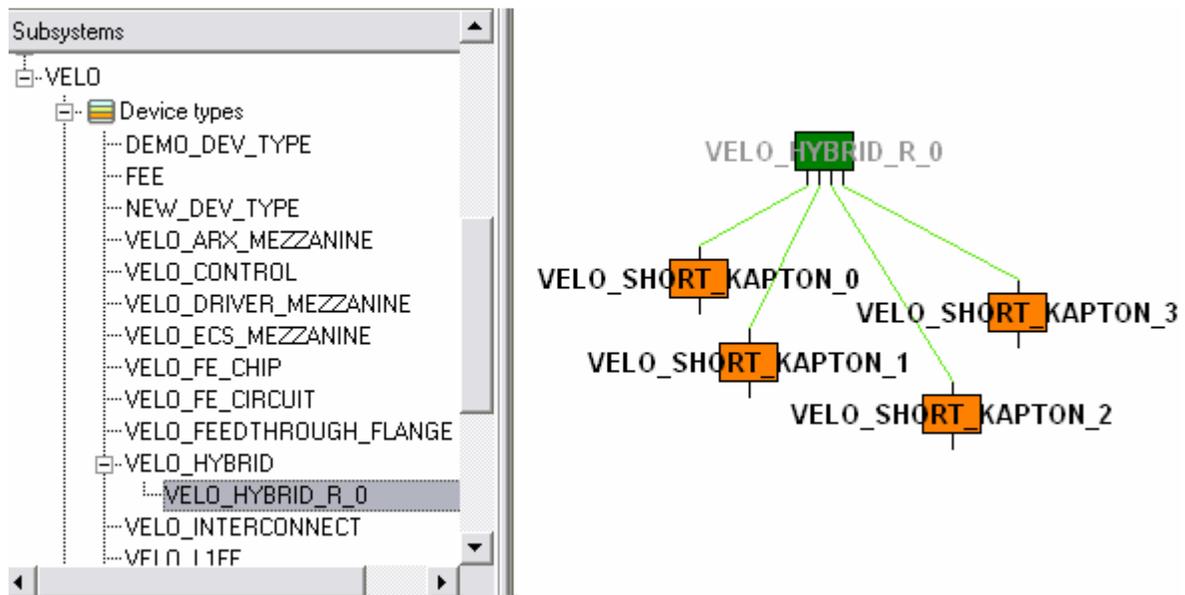


Figure 36 - Visualisation des voisins du *VELO_HYBRID_R_0*

Cependant, en choisissant un des quatre *SHORT_KAPTONs* comme composant central, nous verront le *HYBRID_R_0* avec les trois ports restants comme libres bien que ce ne soit pas cas. Il faut donc bien faire attention au composant central qu'on est entrain de visualiser pour ne pas mal interpréter les informations représentées.

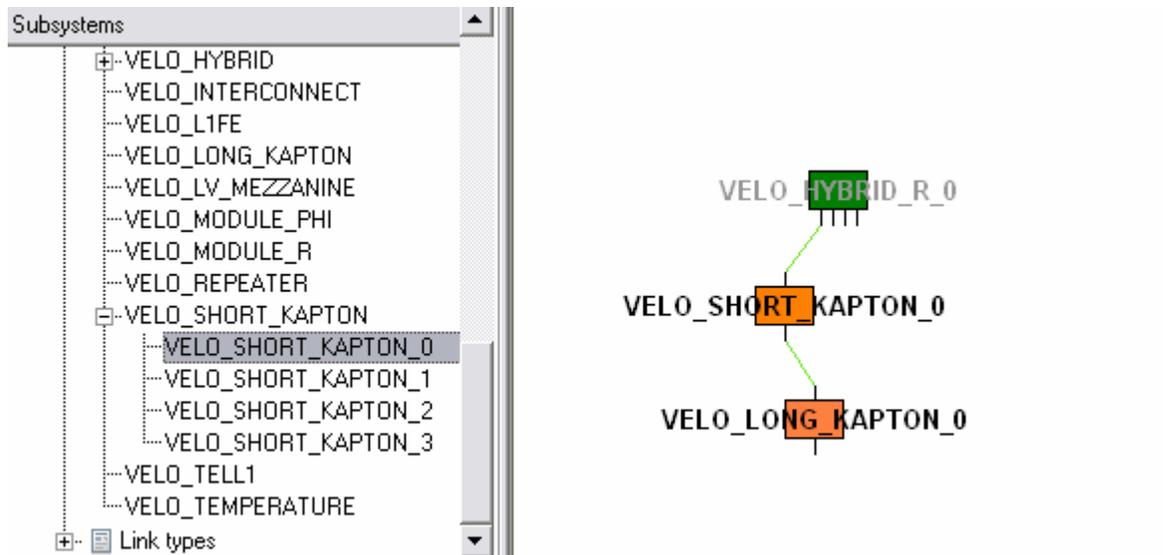


Figure 37 - Visualisation des voisins du VELO_SHORT_KAPTON_0

b) Visualisation des chemins

Le mode de visualisation des voisins et le mode par défaut après le démarrage de l'application. Le passage au mode de visualisation des chemins se fait en actionnant le bouton de la figure suivante qui se trouve sur la barre d'outils.



Figure 38 - Bouton permettant de passer au mode de visualisation de chemins

Après la sélection du mode de visualisation des chemins, l'application affichera les chemins passant par le composant central et non ses voisins. La liste de tous ces chemins sera affichée dans une nouvelle fenêtre qui montrera aussi la longueur de chaque chemin.

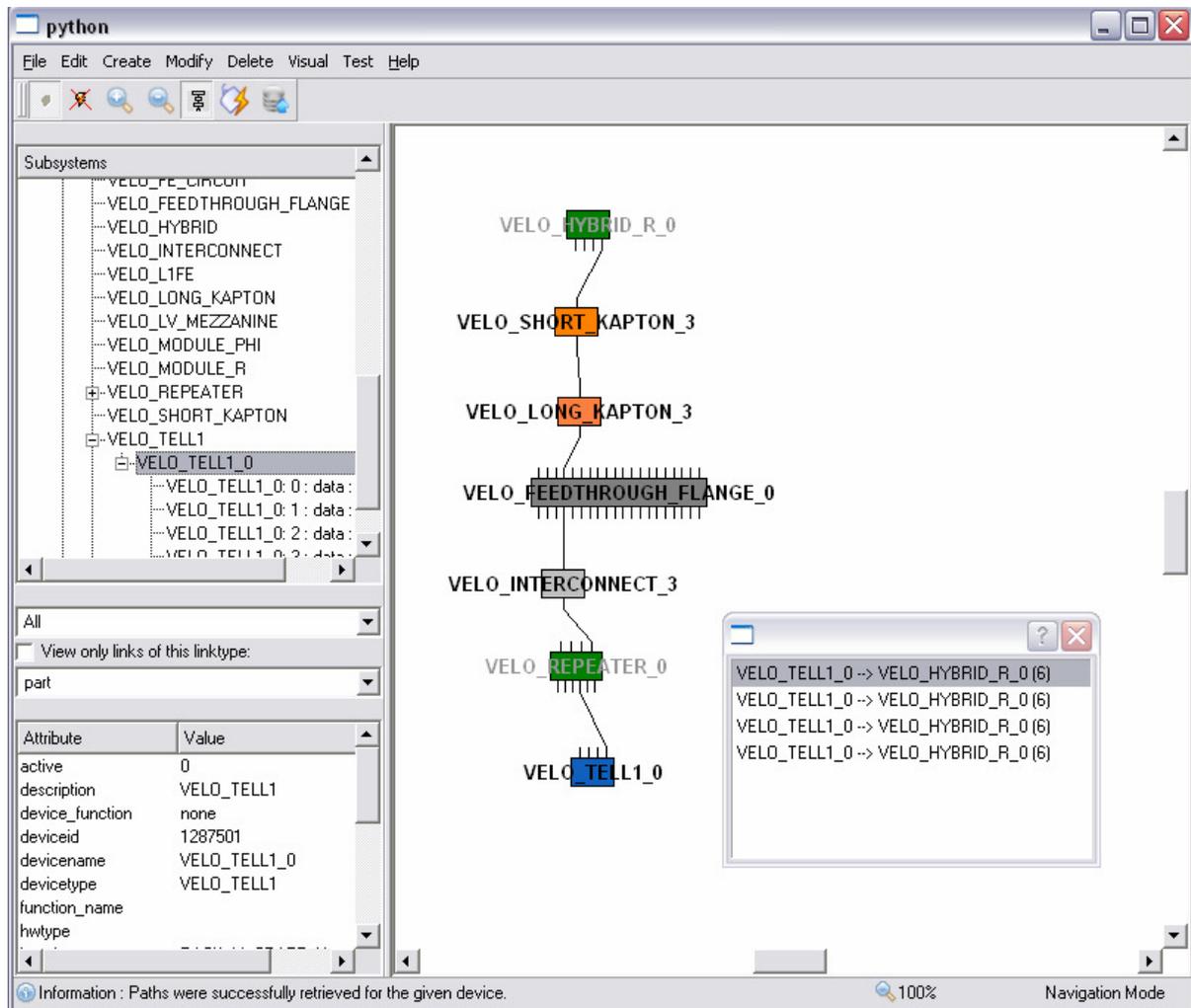


Figure 39 - Visualisation des chemins passant par la carte VELO_TELL1_0

Dans la figure ci-dessus, on peut voir que l'application a trouvé quatre chemins passants par la carte *VELO_TELL1_0* et que chacun de ces chemins fait une longueur de six connexions.

Dans la version actuelle de l'application, l'opération de recherche de chemin prend beaucoup de temps avant d'aboutir (une minute et trente secondes pour la carte *VELO_TELL1_0*).

En fait, la recherche de chemins se fait par la librairie CIC DB Lib qui est écrite en C. Le long temps d'attente nécessaire pour la recherche des chemins ne peut donc être imputé à Python et il est principalement dû au fait qu'il faut faire beaucoup de parcours dans la base de données avant de trouver les chemins recherchés. Je rappelle que ce temps a été obtenu en utilisant la version 3.5 de la CIC DB Lib, et il paraît que la nouvelle version de cette librairie arrive à réduire ce temps d'attente à quelques secondes en tirant profit du puissant système de cache d'Oracle.

3) Mode de Création

L'application CDBVis permet d'ajouter de nouveaux composants à travers un ensemble de formulaires ou bien à partir d'un fichier texte contenant les données mis dans un format spécifique. L'utilisateur a ensuite le choix entre valider ses changements directement à la base de données en actionnant le bouton suivant qui se trouve sur la barre d'outils :  ou exporter les changements vers un script python qu'il peut être par la suite exécuté en dehors de l'application.

a) Création et modification des composants

Création d'un nouveau Type de composant

Une fois connecté à la base de données et qu'un système ait été sélectionné, l'utilisateur peut créer un nouveau type de composant en utilisant l'action *Create Devices Types* du menu *Create*. L'utilisateur devra ensuite spécifier, à travers un formulaire, le nom du nouveau type, la couleur avec laquelle seront représentés ses composants dans CDBVis, les systèmes dans lesquels il apparaîtra ainsi que le nombre d'entrées/sorties dont il dispose.

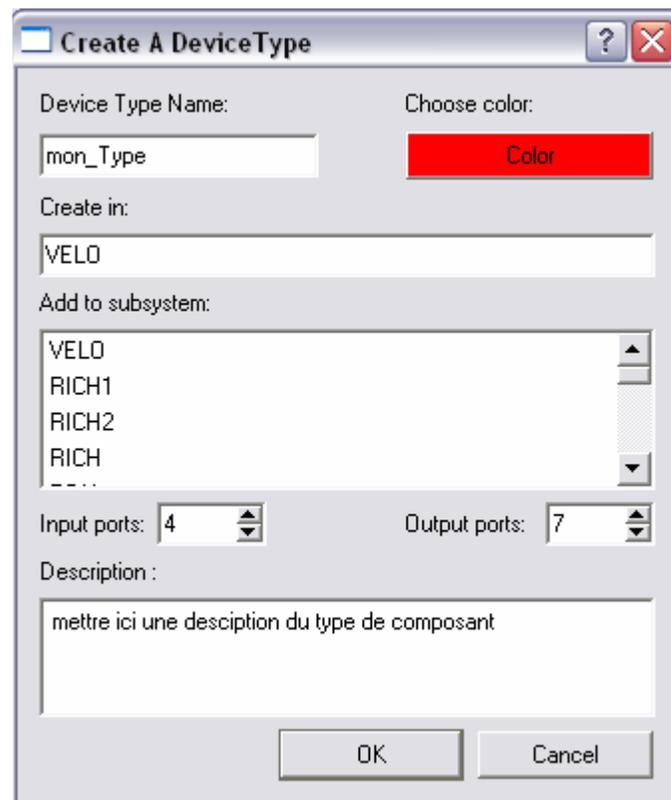


Figure 40 - Formulaire de création d'un nouveau type de composant

Création de composant(s)

Toujours dans le menu *Create*, il est possible d'ajouter de nouveaux composants avec l'action *Create Device(s)*. L'utilisateur doit ensuite remplir les informations concernant le(s) composant(s) à créer. Le nom et le numéro de série sont obligatoires et doivent être différents pour chacun des composants.

Il est possible de spécifier le nombre de composants à créer dans le champ *Total*. Dans ce cas l'utilisateur devra mettre le signe **%d** dans le nom et le numéro de série à l'endroit où il veut faire figurer le numéro du composant.

Le champ *Start number* indique le numéro à partir duquel on commencera l'indexation des composants dans le cas où on en crée plusieurs.

Il est aussi possible de spécifier le nombre de chiffres à prendre en considération pour l'indexation des composants. Dans la figure qui suit on a choisi d'afficher deux chiffres. Le nom du premier composant créé sera donc *mon_composant_01* et non *mon_composant_1*.

Une fois le bouton *Ok* activé, les différents composants seront créés et les changements se répercuteront aussi sur la zone de sélection où on verra s'ajouter les nouveaux composants dans l'arbre de sélection.

Figure 41 - Formulaire de creation de composant(s)

Modification de composant

Dans la partie visuelle, il est possible de modifier les informations d'un composant. Pour cela il faut faire un clic droit sur le composant à modifier et choisir ensuite dans le menu contextuel qui apparait l'action *Modify*. Cette action affichera le même formulaire que pour la création de composants sauf que certains champs seront grisés car non modifiables tel que le type du composant ou son numéro de série.

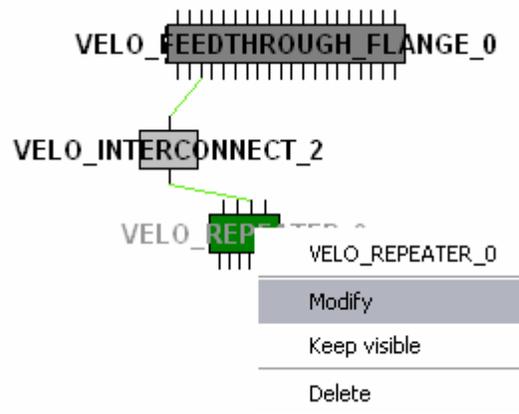


Figure 42 - Menu contextuel suite à un clic droit sur le VELO_REPEATER_0

b) Création de ports

Sur le formulaire de création de composant(s) figure un bouton *Create/Modify ports* qui permet d'afficher un second formulaire pour créer de nouveaux ports.

Les ports créés ainsi seront dupliqués autant de fois qu'il y aura de composants à créer.

	Device Name	Port Nr	Port Type	Port Way	Port Speed	FXI?	OK?	Phy	MAC Address
1	VELO_REPEATER_0	0	data	1	0	0	0	0	none
2	VELO_REPEATER_0	0	data	2	0	0	0	0	none
3	VELO_REPEATER_0	1	data	1	0	0	0	0	none
4	VELO_REPEATER_0	1	data	2	0	0	0	0	none
5	VELO_REPEATER_0	2	data	1	0	0	0	0	none
6	VELO_REPEATER_0	2	data	2	0	0	0	0	none
7	VELO_REPEATER_0	3	data	1	0	0	0	0	none
8	VELO_REPEATER_0	3	data	2	0	0	0	0	none
9	VELO_REPEATER_0	4	data	2	0	0	0	0	none

Figure 43 - Formulaire de création de ports

Le formulaire de création de ports comporte un tableau avec tous les ports déjà disponibles pour le composant sélectionné. Il est ensuite possible d'ajouter ou de retirer des entrées à cette liste.

Pour chaque port il est obligatoire de définir un numéro de port qui soit inférieur au nombre maximum défini pour le composant ainsi qu'un sens pour le port (port de sortie ou d'entrée). Les autres champs sont optionnels sauf dans certain cas bien précis. Par exemple les champs relatifs au côté réseaux (adresse MAC, IP, Masque du réseau...) deviennent obligatoires dans le cas où nos composants appartiennent au système d'acquisition de données (DAQ).

c) Création de connexions

Pour connecter deux composants différents, il faut d'abord s'arranger pour avoir les deux composants sur la zone de visualisation en même temps. Pour cela l'action *Keep Visible* du menu contextuel peut être utilisée. En effet, une fois appliquée à un composant cette action permet de le réafficher la prochaine qu'on choisira un nouveau composant qui devra être celui avec lequel on veut créer la connexion.

L'utilisateur doit ensuite activer le mode de création de lien qui se trouve dans la barre d'outils et qui est représenté par l'icône suivante :

Dans ce mode l'utilisateur ne peut plus déplacer les composants de la zone de visualisation, mais il lui est possible de tracer une ligne entre deux composants. Quand cela est fait, une nouvelle fenêtre apparaît contenant les informations concernant la connexion à créer entre les deux composants sélectionnés.

Figure 44 - Formulaire de création de connexion

La figure ci-dessus représente le formulaire pour créer une connexion entre les deux composants : *RICH1_L1FE_01_01* et *mon_composant_23*. L'application détermine pour chaque composant la liste de ses ports libres. Si un des composants ne dispose d'aucun port libre il suffit donc de cliquer sur le signe plus se trouvant à coté de la liste des ports pour réafficher le formulaire de création de ports décrit précédemment.

Le bouton *Swap input and Output* permet d'inverser la position des deux composants à connecter. L'utilisateur devra aussi définir le type de connexion à créer (simple association, conduite de gaz, transport de données ...).

d) Exportation des données dans un fichier

L'utilisateur a aussi la possibilité d'exporter les modifications à apporter à la base de données sous forme d'un script python qu'il pourra exécuter en dehors de l'application.

Cette option peut s'avérer utile pour dupliquer plusieurs fois la même structure. En effet, CDBVis ne permet de créer que des connexions individuelles, c'est-à-dire relier deux composants seulement à chaque fois. On pourrait donc créer la structure voulue une première fois en utilisant les outils de dessins et ensuite exporter la structure obtenue sous forme d'un script python. L'utilisateur pourra ensuite copier/coller les lignes de codes correspondants à sa structures autant de fois qu'il veut avant d'exécuter le script.

En annexe figure un exemple de code généré par CDBVis.

e) Création à partir d'un fichier

Pour la l'insertion d'une grande quantité de données il peut être utile de pouvoir importer les données à partir d'un fichier texte ou XML. En effet, insérer les données d'un système en utilisant l'interface graphique de CDBVis peut prendre beaucoup de temps, ceci n'est pas d'ailleurs l'objectif de l'application dont le but principale est de visualiser les composants et permettre d'apporter de petites modifications à leurs informations. Toutefois, l'application permet d'importer des fichiers de données formatées d'une manière spécifique pour les transformer en objets qui pourront ensuite être exportés dans un script Python qui permettra de valider les changements vers la base de données.

Dans le cas des fichiers texte, chaque ligne du fichier importé correspond à une entrée dans la base de données. Une ligne du fichier est composée par un ensemble d'attributs séparés par une barre verticale. Chaque attributs contient trois valeurs séparées par des «:», la première valeur correspond au nom de l'attribut, la deuxième représente la valeur que va prendre cette attribut et la dernière valeur correspond au type de données (« s » pour chaine de caractères, « i » pour un entier et « w » s'il s'agit d'une couleur).

La figure suivante représente une ligne du fichier à importer qui correspond à l'ajout d'un nouveau composant de type *FEE* et qui porte le nom de *FEE_MUON_7*.

```
Datatype:Device:s|system_name:MUON,DAQ:s|devicename:FEE_M
UON_7:s|responsible::s|node:False:i|save_status:CREATE:i|hwt
ype::s|devicetype:FEE:s|deviceid::s|serialnb::s|modify_statu
s:NO_STATUS:i|user_comments::s|location::s|
```

Figure - Ligne d'un fichier à importer qui correspond à la création d'un nouveau composant

Chacune des lignes doit commencer par l'attribut *Datatype* qui permet de spécifier le type d'entrée à ajouter parmi les trois types possibles (composant, port, connexion).

En annexe figure un descriptif plus détaillé des différents attributs possibles pour chaque type de données pour l'ajout d'un type de composant.

Conclusion

L'objectif principal de mon stage a été atteint. Le portage vers Qt s'est bien opéré et une version de l'application tournant entièrement sur Qt est disponible en téléchargement sur la page web dédiée au projet sur le site du LHCb.

La réalisation de ce projet n'a pas été sans difficultés surtout au début, car il fallait avoir une bonne connaissance des deux bibliothèques wxPython et Qt pour pouvoir passer de l'une à l'autre.

De plus la, la version de l'application qu'il y avait en place et sur laquelle je devais me baser pour le portage connaissait pas mal de problèmes. En effet, plusieurs modifications y ont été apportées et ont été à l'origine d'un grand nombre de bugs qu'il a fallu corriger tout en effectuant le portage vers Qt.

Toutefois, et avec l'aide des personnes de l'équipe Online, j'ai pu mener à bien ce projet qui m'a permis entre autres d'acquérir de nouvelles connaissances et de me confronter à une situation réelle ou il fallait prendre en compte pas mal de contraintes, ce qui a été très formateur pour moi.

Cependant le travail n'est toujours pas fini et il reste pas mal de choses à faire pour améliorer l'outil CDBVis. Les prochaines étapes importantes pour l'application seraient d'améliorer la librairie CIC DB Lib pour diminuer les temps d'attente ainsi que d'intégrer la connectivité microscopique des composants pour qu'elle puisse être visualisée dans CDBVis.

Il faut aussi se mettre en contact directe avec les utilisateurs de l'application pour pouvoir mieux l'adapter à leur besoins. A cet effet, Eric van Herwijnen et moi avons rencontré à plusieurs reprises des personnes du détecteur VELO pendant la dernière période de mon stage. Une présentation de la CIC DB et de l'outil CDBVis a aussi été faite pendant la réunion du groupe du VELO du 10 aout 2007. Cette collaboration a permis de faire connaître un peu plus l'éditeur auprès de ses utilisateurs et de répondre aussi à certains de leur besoins comme ça a été le cas pour l'ajout d'une adresse web pour chaque composant ce qui a permis de relier la base de données de tests située a Liverpool à la CIC DB.

Ce stage m'a aussi beaucoup apporté au niveau personnel. En effet, j'ai eu l'occasion de d'améliorer mes compétences de communication ainsi que de développer ma culture scientifique au travers des conférences auxquelles j'ai pu assister.

Références bibliographiques

Thomas Johansen, *LHCB Configuration Database Visualizer, Technical Note*, septembre 2006.

Lana Abadie, *An autonomic approach to configure HEP (High Energy Physics) experiments, applied to LHCb (Large Hadron Collider beauty)*, Octobre 2006.

Robert_Shade, *Populating the VELO Connectivity Database*, 2006.

Annexes

Annexe I

Installation

1. Program requirements

The program was developed and tested using the following versions of external software/libraries:

Requirements

The program should in theory run smoothly with:

- Python 2.4
- Qt 3
- ConfDB Library version >3.5
- (AFS Client so that the Boost library can be found on the CERN network.)

Python 2.4 is used due to compatibility issues with the ConfDB Library Python wrapper.

2. Installing on Windows (NICE)

2.1 Installing Python

You can find the recommended stable versions of Python here:

<http://www.python.org/download/>.

To check the version of Python you are running, run the following command on the command line:

```
python -V
```

If you get command not found, Python is not installed on your system. For Windows you will need Python version 2.4, due to compatibility issues with ConfDBLibrary.

It is possible to run two or several versions of Python in parallel, but not recommended unless you know what you're doing. You then download the Python executable from the website, and execute the *.msi or *.exe file to install it. Choose a directory to install it in, and remember this directory. You will have to add the path to the directory to the Path environment variable. This is done by right-click My Computer on your desktop, choose properties, choose the advanced tab and click on the button that says "Environment Variables". In the system variable window, you look for the Path variable, choose to edit, and add the path to the directory where you installed Python at the back of the string. If the string that was already set for the path did not end with a ';' (semicolon), you will have to add that to the string before you add (append) your new path. Click Ok all the way back, to close the windows. You will have to close all open

command line windows to make the change take effect (but in some cases you will have to log off and on again).

To check if Python was successfully installed, open a new command line window, and run `python -V` again to see that your python executable echoes the new version number. If the version number is incorrect it is most likely because you have another version of Python installed on your system that is executed instead. To deal with this; change the name of one of the Python executables (found in the Python root directory), and execute the command:

```
<name_of_my_python_executable> -V
```

To see that it is the correct version. Now you will have to use this name instead of Python when you are running python scripts.

2.2. Installing Qt 3

Due to compatibility issues with ConfDBLibrary we cannot use Python 2.5. While Qt 4 can only be used with python 2.5 we are bounded to use Qt v3.

Qt is available in a free edition for WINDOWS only since the version 4. So we will use a native win32 port of the Qt/x11 (under GPL) sources which use native win32 api and does not require cygwin.

The installation instructions for Qt are available on this page:

<http://qtwin.sourceforge.net/qt3-win32/index.php>

In this page you will have to choose how you would compile the Qt sources (using MinGW or MS Visual Studio ...) then you will find the corresponding instructions.

We have installed Qt using MinGW and we have not encountered any problem compiling the sources.

You are also recommended to download the documentation and Qt demos from the same webpage.

2.3. Installing PyQt

Qt is a C++ library and we cannot use it directly with python. You will need to Install PyQt which is a set of Python bindings for [Trolltech's](#) Qt application.

Before you can build PyQt from source you must have already built and installed [SIP](#) which is a tool that makes it easy to create Python bindings for C++.

Instructions to install SIP then PyQt are available on this page:

<http://kscraft.sourceforge.net/pyqt-windows-install.xhtml>

You can find documentation about SIP and PyQt on:

<http://www.riverbankcomputing.co.uk>

2.4. Installing CdbVis

This will be downloadable from the web in a zip file to be extracted to an arbitrary directory on the user's hard drive.

2.5. Installing Oracle Instant Client 10g

This library is needed for the Oracle database connection used by ConfDB library. Installation instructions for Windows can be found here: <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/APIusage.htm>

2.5 Installing ConfDB library

This is the library used by CdbVis to communicate with the ConfDB. Download the Python Interface library for Windows from: <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/APIusage.htm>. The two *.dll and the two *.lib files should be extracted to the installation directory of CdbVis.

2.6 Installing Python modules

You will need to install the cx_Oracle module that allows access to Oracle databases.

The confDB library already access to the database but we need the cx_Oracle module to do some actions on the database which are not yet included in the API (dealing with the URL table).

Installation instructions are available on this page:

http://www.python.net/crew/atuning/cx_Oracle/

For WINDOWS you just have to download and execute the latest release of the installer for Python 2.4.

3 Testing Installation

If CdbVis is “not working” after you have installed all of its dependencies, here are some tests you should do to see what that fails. To make it easier to debug possible errors, we take one step at a time.

3.1 Test Python

Run the following command:

```
Python
```

You'll enter Python Shell, type:

```
print "hello world"
```

and press enter. If it echoes “hello world” back to you, then it works. Press Ctrl-Z to exit the shell.

3.2 Test Qt:

Run Python then enter:

```
Import qt
```

If this command succeeds without any error you have installed successfully Qt. otherwise try to install qt again.

3.3 Test CdbVis

Go to the directory where you installed CdbVis and run the command:

```
python main.py
```

If it runs, you have done things correct so far, the ConfDB library files have been found as well. If it fails, it is probably because the ConfDB library files were not found, see through the installation of the ConfDB library as described above to see if you missed something.

If *.py files is associated with Python you can actually just double-click on main.py and it will start. (If the *.py files have a icon of a python snake).

3.4 Test ConfDB Connection

Once you have started CdbVis, click on the button which shows a lightning, or choose from the menu: File > Connect, to connect to the ConfDB. If it fails, the error will probably be reported either through CdbVis, or in the command line shell window. Errors here should be directed to Lana Abadie, who can most likely tell you why the database connection failed, and help you out.

Annexe II

Exemple de code généré par CDBVis pour insérer un composant dans la base de données :

```
#####  
#          AUTOGENERATED PYTHON CODE          #  
#          BY CDBVIS          #  
#####  
  
# Edit the following :  
databasename="Your_DB_Name"  
username="Username"  
password="Password"  
  
from confDBpython import *  
import cx_Oracle  
  
cfDB = CONFDB(databasename,username,password)  
cfDB.DBConnexion()  
  
oracle = cx_Oracle.connect( username + '/' + password + '@' +  
databasename)  
cfDB2 = oracle.cursor()  
  
cfDB.InsertFunctionalDevice("RICH1","newdevice","switchs",0,1,"sd  
fgfgdf","","","","","none",1)  
  
cfDB.DBDeconnexion()  
oracle.commit()
```

Annexe III

Description des attributs à renseigner pour le type de composants dans le fichier d'insertion de masse :

Nom de l'attribut	Exemple de valeur	Type de données	Optionnel / Obligatoire	Commentaire
Datatype	DeviceType	s	REQUIRED	Identifie les types de composant
system_name	MUON,VELO	s	REQUIRED	Noms des sous systèmes sépaés par des virgules.
save_status	CREATE	i	REQUIRED	CREATE, MODIFY or RENAME. Pour dire si le l'objet doit être créé, modifié ou supprimé.
description	...	s	OPTIONAL	Description du type de composant
rgbcolor	(255, 0, 0)	w	REQUIRED	Couleur du type de composant
devicetypeid	1	i	OPT/RE	Obligatoire si on veut modifier ou renommer un composant déjà existant dans la base de données
devicetype	FEE	s	REQUIRED	Nom du type de composant
old_name	FE	s	OPTIONAL	Ancien nom du type de composant si celui on veut le renommer
nbrofoutput	3	i	REQUIRED	Nombre de ports de sortie
nbrofinput	4	i	REQUIRED	Nombre de ports d'entrée