Ecole Supérieure Multinationale des Télécommunications

Ecole Supérieure Polytechnique

MEMOIRE DE FIN DE CYCLE.

Pour l'obtention du
Diplôme d'Ingénieur Technologue.
Spécialité: Téléinformatique.

Sujet:

Réalisation d'une version graphique de YAO: Logiciel de modélisation et de génération de code pour l'assimilation de données variationnelles.

Lieu de stage: E.S.P/LTI

Période du stage: 06/04/09 au 06/10/09

Présenté par: MAYAKI YERIMA Abdouramane.

Professeur encadreur :

Dr. Alex CORENTHIN

Maitre de stage:

Dr. Awa NIANG

Promotion 2007-2009

Dédicaces

Je dédie ce mémoire à :

- ✓ Mes parents qui m'ont donné naissance avec la grâce de DIEU. Ils ont consacré toute leur vie pour faire de moi ce que je suis aujourd'hui dans une logique de foi, dignité, honneur et persévérance.
- ✓ Mes sœurs et frères: qu'ils retrouvent ici l'exemple et aussi qu'ils fassent du travail leur cheval de bataille.

Quoique je dise, mes mots ne pourront exprimer mon amour pour vous.

Remerciements

Je rends grâce à DIEU de m'avoir permis d'achever ce travail et de m'avoir donné la foi en ce que je fais.

Aussi, je tiens à remercier particulièrement :

- ✓ *Les responsables du laboratoire LTI de l'ESP.*
- ✓ *Mes professeurs encadreurs à savoir Mme Awa NIANG et M. Alex CORENTHIN.*
- ✓ *M. Luigi NARDI pour son assistance depuis la France.*
- ✓ *Dr. Samuel OUYA pour ses précieux conseils.*
- ✓ *Tout le corps professoral de l'ESMT.*
- ✓ *Mes très chers Amis à savoir Sokhna DIOUF, Désirée IRIE LOU, Assaghada DACHAM, Zoul-Koufle ALASSANI, Teddy DOKOUNA, Aldinga SUBA, et Zimé-Mansourou SUANON.*
- ✓ *Mes condisciples et amis du laboratoire LTI «Le peuple LTI».*
- ✓ *La promotion DIT-Téléinformatique 2007-2009.*
- ✓ *« Par leur manière de penser et d'entrer dans votre propre raisonnement, des gens possèdent le pouvoir très rare de vous rendre plus intelligent. Vous vous surprenez alors dans vos discussions avec eux, à aller plus loin dans vos idées et vous croyez que vous avez été naturellement bon. Ce n'est qu'après que vous prenez conscience de leurs capacités à vous révéler. C'est une chance de les rencontrer sur notre chemin...» A ces gens là, je dis Merci.*

Avant-propos.

Pourquoi élaborer un tel document?

Une bonne communication au sein d'une entreprise moderne qui transmet le savoir est essentielle à sa bonne marche. Savoir transmettre une information et surtout savoir bien écrire peut donc revêtir une importance considérable. C'est pourquoi la rédaction de mémoires occupe une place importante dans les deux écoles que nous fréquentons à savoir l'Ecole Supérieure Multinationale des Télécommunications de Dakar (E.S.M.T) et l'Ecole Supérieure Polytechnique (E.S.P).

Le mémoire nous apprend d'une part à recueillir, analyser, organiser et présenter de l'information sur un sujet donné autour d'une problématique posée. D'autre part, les stages en entreprise complètent les connaissances théoriques acquises durant toute l'année.

Au cours des différents stages, nous aurons l'occasion de préparer plusieurs types de rapports; ces documents deviendront une description écrite de notre travail.

C'est ainsi que le présent document tient lieu de mémoire de fin de cycle d'Ingénieur Technologue. Il a pour objectif de présenter le travail effectué au Laboratoire de Traitement de l'Information (LTI) de l'Ecole Supérieure Polytechnique de Dakar de sur le thème proposé.

TABLE DES MATIERES

Table de figures

Introduction

Dans le domaine scientifique, une chose est de découvrir une technologie, une autre est de suivre son évolution afin de pouvoir répondre aux besoins des Hommes. Besoins qui ne cessent d'évoluer avec le temps.

Les premiers ordinateurs occupaient de grandes salles, leurs capacités de stockage étaient très réduites (de l'ordre du kilo-octet). Les développeurs de l'époque «codaient»[1] presque en binaire. L'évolution générale et parfaite dans ce monde concurrentiel ne sera certes jamais atteinte mais constitue la raison principale qui pousse l'Homme à faire toujours des recherches. C'est ainsi que les ordinateurs de nos jours sont de taille très petite, avec une capacité de stockage qui atteint le tera-Octet, intégrant des processeurs de plus en plus «rapides» et implémentant des programmes en «langage humain»[2].

Le thème de ce mémoire intitulé **«Réalisation d'une version graphique de Yao: logiciel de modélisation et de génération de code pour l'assimilation de données variationnelles»** entre dans la même logique scientifique.

Dans les lignes qui suivront, nous allons tenter de trouver la réponse à chacune des questions suivantes:

- ✓ Quel est le projet?
- ✓ Quel est le résultat attendu?
- ✓ Comment fait-on pour créer des interfaces graphiques?
- ✓ En matière de conception d'une telle application, que doit-on utiliser, y'a-t-il une démarche?
- ✓ S'il y'a plusieurs outils (démarches), lequel (laquelle) devons-nous utiliser et pourquoi?
- ✓ L'interface, ne dépend-t-elle pas de l'outil existant?
- ✓ Comment se fait l'implémentation?

Cette étude trouve son importance dans le souci d'élaborer une solution souple et adéquate pour le logiciel.

---

[1]: Écrivaient les programmes .

[2]: Langage facilement compréhensible par l'homme tel le C++.

L'approche que nous proposons de découvrir ici sera axée sur une logique structurée en trois (3) chapitres :

- ✓ Dans le premier chapitre, nous ferons une présentation générale du cadre de travail et du thème proposé. Nous présenterons également quelques méthodes qui permettent de concevoir des objets graphiques.
- ✓ Au second chapitre, nous évoquerons l'existant à travers une étude en fonction des éléments qui importent pour notre projet.
- ✓ Dans le troisième et dernier chapitre nous proposerons des solutions qui, d'une part pourront servir à l'entreprise qui a bien voulu nous soumettre ce travail et d'autre part, nous permettront d'être mieux armé pour affronter le monde de la programmation.

Chapitre 1

Présentation générale.

1.1 Présentation de la structure d'accueil.

Dans le cadre de notre projet, deux (2) structures d'accueil entrent en jeu. Il s'agit du Laboratoire de Traitement de l'Information de l'Ecole Supérieure Polytechnique de Dakar (LTI), notre environnement de travail, et du laboratoire LOCEAN basé en France. En effet, c'est principalement au niveau de ce dernier que se trouvent les utilisateurs de notre application. Nous travaillons donc à distance à partir du LTI. Pour des questions d'ordre technique entrant dans l'amélioration ou la bonne compréhension des besoins de ces utilisateurs, il nous est arrivé à plusieurs reprises de contacter le LOCEAN.

1.1.1 Présentation du LTI.

Créé en janvier 2004 le Laboratoire de Traitement d'Information est devenu l'un des plus importants laboratoires de l'Ecole Supérieure Polytechnique, unité de recherche transversale sur les Sciences de l'Ingénieur. Les activités du LTI couvrent un large spectre qui part de l'océanographie pour aboutir à des systèmes complexes de transport, en passant par tous les composants de la chaîne informatique : réseaux, systèmes répartis, calcul numérique et calcul formel, logiciels de la recherche d'information et d'aide à la décision, codage des langues nationales sur Internet.
Le LTI se veut centre africain de référence et d'excellence au cœur de la recherche et de ses applications. C'est dans cette optique que des collaborations avec les grandes écoles de la sous région ont été créées, unissant leurs compétences à celles du LTI. Le laboratoire est également largement impliqué dans les formations par la recherche, à travers le Mastère Recherche SPI.

Les chercheurs du LTI participent à de nombreux programmes de recherche nationaux ou internationaux (AUF, CRDI, CNRS, etc...). Ils sont les acteurs majeurs de l'activité scientifique internationale par le biais de l'animation de comités éditoriaux de revues, de l'organisation de très nombreuses conférences, l'accueil des chercheurs étrangers etc... Enfin, le LTI entretient un tissu de relations industrielles et des collaborations académiques avec des universités ou des centres de recherche comme l'IRD à travers des projets communs.

1.1.2 Présentation du LOCEAN.

C'est un laboratoire dont l'activité est dévolue à l'étude des processus de la variabilité océanique et de leurs interactions avec la variabilité climatique.

En ce qui concerne le premier volet, le LOCEAN étudie un large spectre d'échelles qui va de celles du mélange vertical et des ondes internes à celles des mouvements planétaires.

Dans le second volet, le laboratoire s'intéresse plus spécifiquement au rôle de l'océan sur la variabilité du climat, en particulier aux échelles qui vont de l'inter-annuel au siècle, aussi bien pour le climat passé, que présent et futur. L'évolution du laboratoire au cours des dernières années a montré un clair élargissement du spectre des études réalisées, avec

- ✓ des ouvertures sur les études d'impact, un prolongement naturel d'études sur la variabilité climatique, en interaction avec d'autres communautés, soit atmosphériciennes, soit continentalistes et les milieux applicatifs (santé, agriculture, etc…), mais aussi
- ✓ des ouvertures sur des échelles de temps et d'espace plus variées et des chantiers plus diversifiés que par le passé, en particulier en océanographie physique (en Arctique, par exemple, sur les interactions entre circulation, glace de mer, ondes internes et mélange, ou encore sur les sub-méso échelles de la variabilité horizontale, des chantiers sur des régions de talus continental (upwellings tropicaux, Golfe de Lion), ou la calibration de proxies, parmi tant d'autres).

Par ailleurs, le LOCEAN accueille deux services nationaux de l'INSU:

- ✓ NEMO, plate-forme de modélisation numérique de l'océan utilisée pour la recherche fondamentale, l'océanographie opérationnelle, les prévisions saisonnières et les études climatiques, dont la coordination et le développement sont assurés au LOCEAN;
- ✓ CARAUS qui assure la mesure du carbone inorganique et de paramètres associés dans l'océan austral au cours de campagnes répétées. Le LOCEAN héberge aussi le service d'analyse SNAPO-CO2 du carbone inorganique et de l'alcalinité de l'eau de mer.

Ce vaste champ de recherche est mené dans le cadre de projets s'appuyant fortement sur des analyses numériques et d'observations, tant in situ que par satellite, mais inclut aussi du développement instrumental et méthodologique, la réalisation de campagnes expérimentales ainsi

que la participation à des missions satellitaires. Il sera réalisé au sein de sept équipes (SURF, PHYBIOCAR, AUSTRAL-BOREAL, VARCLIM-TG, RICLES, PALEOPROXUS, MMSA).

Les projets ainsi que le champ d'activité des différentes équipes sont présentés sous les rubriques des équipes.

- **Organigramme de LOCEAN.**

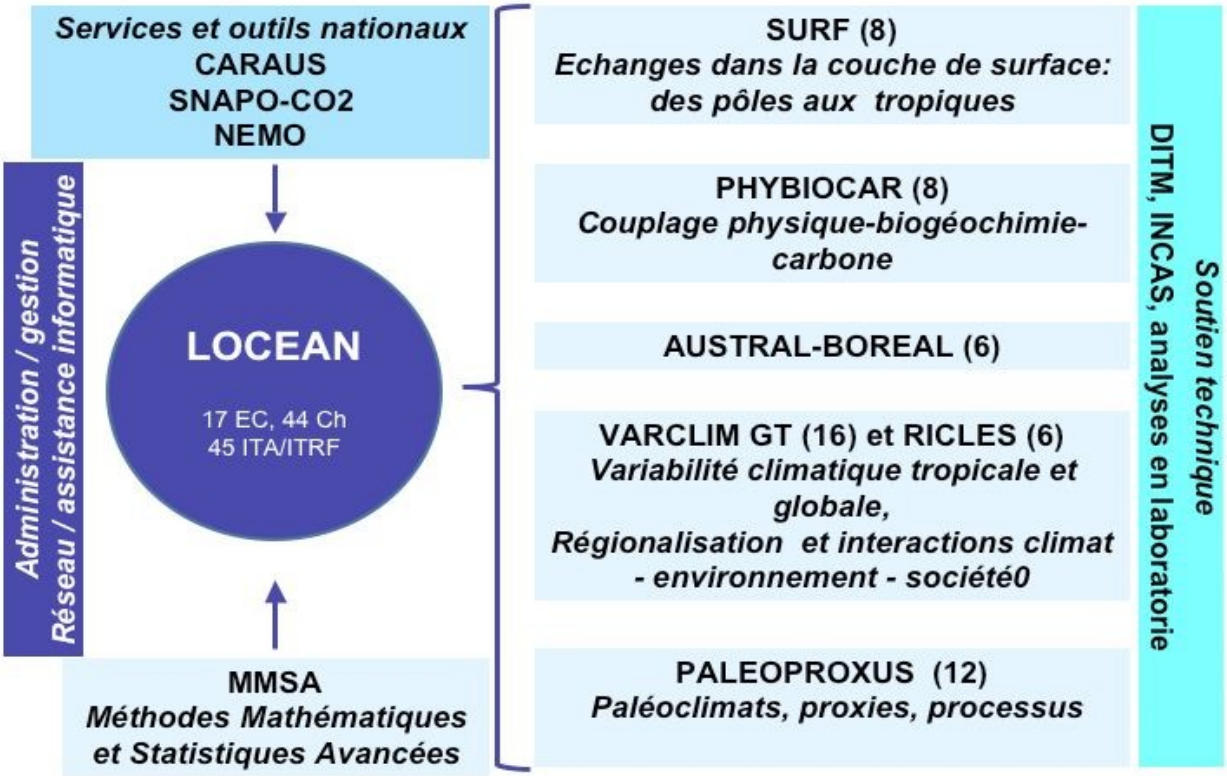

Figure 1: Organigramme de LOCEAN

1.2 Contexte du projet.

Après avoir présenté nos structures d'accueil, nous allons essayer à ce niveau d'énoncer le cadre d'application du projet et la problématique.

1.2.1 Cadre d'application du projet.

Nous travaillons sur une application qui est très utilisée à LOCEAN. Notre tâche consistait à améliorer les conditions de travail des utilisateurs de ladite application en créant une version graphique qui découlera de l'amélioration de l'ancienne version. Précisons qu'actuellement l'application ne peut s'installer que sur le système linux mandriva.

1.2.2 Problématique.

Yao, c'est le nom de l'application que nous étudions. Yao est un logiciel de modélisation et de génération de code pour l'assimilation de données variationnelles. Dans son état d'origine, son appréhension s'avère très délicate (moyennant quelques notions en programmation). Depuis son installation jusqu'à sa prise en main, tout se faisait en «lignes de commandes». Pour faciliter son installation et son utilisation, nous nous sommes fixés des objectifs. Est-ce que le thème «Réalisation d'une version graphique de YAO» que nous nous proposons d'étudier permet d'atteindre cet objectif?

1.2.3 Objectifs.

Ce projet a pour but la conception de la partie vue (du modèle M-V-C) de YAO qui faciliterait ou masquerait certaines étapes de l'installation et de l'utilisation du logiciel. D'une part cette version graphique que nous projetons de réaliser au LTI en collaboration avec l'équipe MMSA, créateur du YAO textuel améliorerait les conditions de travail des utilisateurs actuels de Yao. D'autre part nous pensons que sa diffusion pourrait intéresser un grand nombre de scientifiques désirant se mettre à l'assimilation des données. Ce travail aboutira à un produit diffusable, encore améliorable, mais final pour une première version.

1.3 Méthodes d'analyse et de conception.

Nous allons présenter dans cette partie quelques méthodes permettant de créer et manipuler des objets graphiques. En plus des bibliothèques qui sont spécifiques à chaque système d'exploitation, il en existe beaucoup d'autres qui sont multiplates-formes. Ci-après, quelques exemples:

- ✓ **Sous Windows** : Windows API, la MFC.
- ✓ **Sous Mac OS X** : Cocoa.
- ✓ **Sous Linux** : Xlib.
- ✓ **Sous Windows, Mac OS X et/ou linux**: Java (AWT et SWING), GTK+, WxWidget, Qt

1.3.1 Windows API.

Windows API ou **WinAPI** est le nom donné par Microsoft à l'interface de programmation (API) sur les systèmes d'exploitation Microsoft Windows[w1]. Elle est conçue pour les langages de programmation C et C++ et est la manière la plus directe pour une application d'interagir avec le système d'exploitation Windows.

Description de WinAPI.

Comme services de bases, elle donne accès aux ressources fondamentales du système disponibles sous Windows. Exemples: Système de fichiers, périphériques, processus, processus léger, accès au registre système et au Système de gestion d'exceptions.

Cette interface de programmation permet d'accéder aux ressources pour l'affichage sur les moniteurs, imprimantes, etc.... Elle permet aussi d'afficher et de gérer les contrôles de base comme les boutons et barres de défilement, de recevoir les informations du clavier et de la souris et des fonctionnalités associées comme l'environnement graphique. Elle affiche les «boites de dialogue» pour ouvrir et enregistrer des fichiers, choisir la couleur et la police. Elle donne accès à des fonctions avancées du système d'exploitation comme des barres de statut (situées au bas des fenêtres), barres de progression, barres d'outils et onglets. Diverses possibilités de gestion de réseau du système d'exploitation. Ses sous-composants incluent NetBIOS, Winsock, RPC etc... Sur les anciennes versions de Windows, cela incluait NetDDE.

Chaque version de Windows a apporté son lot de modifications de l'API. Malgré cela, le nom de

l'API reste le même sauf pour les modifications majeures. Microsoft a quand même changé le nom Win32 en Windows API pour les contenir dans une seule famille même pour une prochaine modification de celle-ci.

Win16 était la première API pour les versions 16-bits du système. Elle était au départ simplement nommée Windows API mais a été renommée en Win16 pour permettre le passage au 16-bits de Windows API. Les fonctionnalités étaient principalement dans le noyau du système: kernel.exe (ou krnl286.exe ou krnl386.exe), user.exe et gdi.exe. Malgré l'extension exe, ces fichiers sont des bibliothèques de liens dynamiques.

Win32 est la version 32-bits de l'API pour les systèmes plus récents. L'API se compose d'implémentations, comme avec Win16, dans le système bibliothèques de liens dynamiques. Le noyau de Win32 réside dans les fichiers kernel32.dll, user32.dll, et gdi32.dll. Win32 a été présentée avec Windows NT. La version embarquée avec Windows 95 était appelée Win32c, avec "c" pour compatibilité (compatibility en anglais) mais ce nom a été abandonné pour Win32. Dans la version Windows NT 4.0 et suivantes, les appels de Win32 sont exécutés par deux modules, csrss.exe (Client/Server Runtime Server Subsystem) en mode utilisateur et win32k.sys en mode noyau.

Win32s est une extension de Win32 pour les systèmes Windows 3.x qui a été introduite comme sous-ensemble de Win32. Le "s" est pour sous-ensemble (subset en anglais).

Win32 pour les éditions 64bits précédemment appelée **Win64** est la version pour les ordinateurs 64-bits, avec les versions *Windows XP Professional x64 Edition* pour les processeurs x86-64 ainsi que Windows XP 64-bits Edition et Windows Server 2003 pour les processeurs Itanium. Cette version apporte juste le support pour ces deux nouvelles plateformes.

Avantages et inconvénients de la WinAPI.

Cette bibliothèque permet de créer des fenêtres sous Windows. Elle est toutefois assez complexe et il faut beaucoup de lignes de code pour arriver à ouvrir une simple fenêtre. L'API Win32 est un ensemble de fonctions. Cette bibliothèque soumise aux droits d'auteurs n'utilise pas la POO. Pour palier à ce problème, Microsoft a créé une autre bibliothèque appelée MFC.

1.3.2 La MFC.

Les Microsoft Foundation Class (MFC) proposent un ensemble de classes couvrant l'ensemble des besoins pour développer une application Windows[w2].

Description de la MFC.

C'est une bibliothèque de classes C++ encapsulant l'API Win32 (pour rappel, écrite en C) de Windows[w]. Leur première apparition date de 1992. Elle offre également un framework de développement de type Document/Vue inspiré du motif de conception Modèle-Vue-Contrôleur (MVC). Les MFC ont atteint leur maturité avec la version 7.0 de Visual Studio. Certains programmeurs apprécient la possibilité d'accéder à la plus grande partie des fonctionnalités de cette API.

Elles fournissent des modèles d'architecture sur lesquels l'application va être construite, notamment le modèle document - vue déclinée en trois options : applications SDI (single document interface : une seule fenêtre), MDI (multiple document interface : plusieurs fenêtres), et une application de type boite de dialogue.

L'ensemble des classes fenêtres est une encapsulation des contrôles Windows de base. Les MFC utilisent l'héritage simple (pas d'héritage multiple), l'ensemble des classes forme une hiérarchie.

Avantages et inconvénients de la MFC.

Apparue après WinAPI, la MFC a aussi des désavantages: documentation mal structurée, problème de gestion de l'unicode, internationalisation dans le code (au lieu d'être une simple option à indiquer), emploi de templates figés qui créent souvent les vues (document – vue), utilisation de boucle d'évènements (messages) et pas de callback/listener, surcouche permettant d'accéder à l'API windows est en C, utilisation exclusive de Visual Studio.

Bibliothèque qui se contente d'appeler les fonctions de l'API Win32 (on dit que c'est une surcouche) tend aujourd'hui à disparaître sous Windows, progressivement remplacée par la bibliothèque .NET.

1.3.3 Cocoa.

Cocoa est une API native d' Apple pour le développement orienté objet sur son système d'exploitation Mac OS X[w3]. C'est l'une des cinq API majeures disponibles pour Mac OS X, les autres étant: Carbon, la boite à outils Macintosh (pour l'environnement obsolète classic), POSIX (pour l'environnement BSD), et Java. Certains environnements, comme Perl et Ruby sont considérés

comme mineurs, car ils n'ont pas accès à toutes les fonctionnalités et ne sont généralement pas utilisés pour le développement d'applications à part entière.

Description de Cocoa.

Les applications Cocoa sont typiquement construites en utilisant les outils de développement fournis par Apple, Xcode (anciennement *Project Builder*) et Interface (utilisant le langage de programmation Objective-C). De plus, l'environnement de programmation Cocoa peut être accessible en utilisant d'autres outils, comme le Pascal, le Python, le Perl et le Ruby, avec l'aide de mécanismes passerelles tels que PasCocoa, PyObjC, CamelBones ou encore RubyCocoa. Il est aussi possible d'écrire un programme Objective-C Cocoa dans un simple éditeur de texte et de le compiler par la suite avec GCC (compilateur GNU C) ou en utilisant les scripts makefile de GNUstep.

Pour l'utilisateur final, les applications dites Cocoa sont considérées comme étant celles écrites en utilisant l'environnement de programmation Cocoa. Habituellement, ces applications ont un ressenti différent dû à l'automatisation d'une multitude d'aspects de l'application par l'environnement Cocoa. Ceci est ainsi fait pour suivre la politique de développement d'Apple.

Avantages et inconvénients de Cocoa.

Au fil des versions de Mac OS X, on assiste à un rapprochement progressif de Cocoa et de Carbon, construits de plus en plus à partir de la même base (Core Foundation). Certaines applications Carbon, tirant partie des nouvelles fonctionnalités de Mac OS X, ne fonctionnent plus sous Mac OS 9: la limite se fait de plus en plus ambiguë. Il est rare de trouver des applications Cocoa qui ne font aucun appel à l'API Carbon.

Cocoa est principalement constituée de deux bibliothèques appelées frameworks. Les *frameworks* sont similaires en fonctionnalité aux bibliothèques partagées — un objet compilé peut être dynamiquement chargé dans une adresse mémoire du programme pendant l'exécution — mais les *frameworks* offrent aussi des ressources liées, des fichiers d'en-tête, ainsi qu'une documentation fournie.

- ✓ *Foundation Kit*, ou plus simplement **Foundation** est apparu pour la première fois dans OpenStep. Sur Mac OS X, il est basé sur Core Foundation. *Foundation* est une bibliothèque orientée objet générique, fournissant des outils de manipulation de chaîne de caractères, de valeurs numériques ou encore diverses structures de données. Des outils pour le calcul

distribué et les boucles événementielles ainsi que des fonctionnalités indirectement liées à l'interface graphique utilisateur sont aussi proposés par *Foundation*.

- ✓ *Application Kit* ou **AppKit** est directement hérité de l'original NeXTSTEP Application Kit. Il contient du code permettant de créer et d'interagir avec les interfaces graphiques utilisateurs. AppKit est construit comme une surcouche de *Foundation* et utilise par conséquent le même préfixe «NS».
- ✓ *Core Data* ou ***frameworks*** **de persistance**, il facilite l'enregistrement des objets dans un fichier, et par la suite leur chargement en mémoire.

Une partie clef de l'architecture Cocoa est son modèle de visualisation complet. L'organisation, bien que réalisée de façon conventionnelle pour un framework application, est basée sur le modèle de dessin PDF fourni par Quartz. Ce modèle permet la création de contenu dessiné personnalisé en utilisant des commandes de dessin dans le style PostScript, ce qui implique aussi une prise en charge automatique de l'impression. Par ailleurs, depuis que le *framework* Cocoa se charge du cadrage, de la taille ou du déplacement d'objets graphiques, ainsi que d'autres tâches inhérentes à la création de ces objets, le programmeur n'a plus besoin d'implémenter ces fonctionnalités basiques et peut se concentrer uniquement sur les fonctionnalités propres à son application.

1.3.4 Xlib.

Xlib est le nom d'une bibliothèque logicielle offrant une implémentation de la partie cliente du protocole *X Window System* en langage C[w4].

Description de Xlib.

Elle contient des fonctions de bas niveau pour interagir avec un serveur X. Ces fonctions permettent aux programmeurs d'écrire des programmes sans connaître les détails du protocole X. Peu d'applications utilisent la Xlib directement; en général, elles exploitent d'autres bibliothèques qui reposent sur la Xlib pour fournir des éléments d'une interface graphique.

Cette bibliothèque est apparue vers 1985, implémentée par XFree86, et elle est toujours très utilisée par des systèmes UNIX et dérivés. Elle sert de base pour la plupart des toolkits graphiques de haut niveau comme: Intrinsics (Xt), Motif, GTK, Qt, etc...

Avantages et inconvénients de Xlib.

Système graphique le plus populaire du système d´exploitation UNIX et dérivés. Sa large distribution, le fait qu'il soit libre de tous droits de distribution et surtout ses qualités techniques exceptionnelles en ont fait un standard de l'industrie du logiciel. Ses caractéristiques principales sont:

- ✓ L'utilisation d'une architecture client/serveur et cela dans une totale hétérogénéité (le serveur et le client peuvent fonctionner sur des architectures totalement différentes). Le dialogue entre le serveur et les clients se fait conformément au protocole X ce qui permet d'assurer la transparence du dialogue. La portabilité des librairies et des applications X11 (le même code tourne sur une grande station de travail ou un petit PC sous Linux).
- ✓ En dépits des avantages de X11, il serait faux de croire que le développement d'une application X soit une chose simple. Les concepts utilisés sont complexes (programmation bas niveau), le nombre de fonctions à connaitre est important et la documentation titanesque parfois difficile à aborder.

1.3.5 Java (AWT et SWING).

L'une des raisons de la popularité du langage java tient à ce qu'il facilite la création de programmes aux graphismes étonnants [b1]. Ces prouesses sont rendues possibles grâce aux bibliothèques graphiques, AWT (Abstract Window Toolkit) et SWING. Cette dernière s'appuie sur AWT et permet aux applications de posséder l'apparence et le style de la machine hôte.

Swing donne aux développeurs un meilleur contrôle sur les entités graphiques. Ces deux bibliothèques sont multiplate-formes.

Description de Java.

Java est le nom de marque d'une technique développée par Sun Microsystems : la « technologie Java™ ». Elle correspond à plusieurs produits et spécifications de logiciels qui, ensemble, constituent un système pour développer et déployer des applications. Java est utilisée dans une grande variété de plates-formes depuis les systèmes embarqués et les téléphones mobiles jusqu'aux serveurs, les applications d'entreprise, et dans une moindre mesure pour les interfaces

graphiques comme les applets Java du Web.

Depuis des années, Sun Microsystems appelle Java la «technologie Java» dans son ensemble. En pratique, beaucoup de programmeurs utilisent le mot «Java» pour désigner son langage de programmation, tandis que la plate-forme d'exécution est appelée «JRE» (Java Runtime Environment) et le système de compilation : «JDK» (Java Development Kit) plutôt que «compilateur Java». Java est sous licence GNU GPL depuis novembre 2006.

Le langage Java est un langage de programmation orienté objet. Un programme java s'exécute dans une machine virtuelle, dite machine virtuelle Java.

Le bytecode Java est le résultat de la compilation d'un programme écrit en Java par le compilateur Java.

La plate-forme Java correspond à la machine virtuelle Java plus des spécifications d'API : Java SE, Java EE, Java ME.

Java Platform, Standard Edition (Java SE) contient les API de base destinées aux ordinateurs de bureau. Java Platform, Enterprise Edition (Java EE) contient (en plus du précédent) les API orientées entreprise destinées aux serveurs. Java Platform, Micro Edition (Java ME) est destiné aux appareils mobiles tels que assistants personnels ou smartphones.

Avantages et inconvénients de Java.

Les avantages de la technologie Java sont entre autres la portabilité, la souplesse, une documentation organisée. Cependant, ses failles résident pratiquement autour de la machine virtuelle Java (JVM) qui est certes portable mais n'offre pas de possibilité de gestion «personnalisée» de la mémoire. L'interprétation du «bytecode» se fait en fonction de la JVM. Elle augmente le temps d'exécution (les programmes écrits donc en Java mettent beaucoup plus de temps pour afficher leur résultats).

1.3.6 GTK+

GTK+ est une bibliothèque qui permet de créer des interfaces graphiques (GUI en anglais) très facilement.

Description de Gtk+.

A l'origine, GTK+ fut développée pour les besoins du logiciel de traitement d'images GIMP (GNU Image Manipulation Program)[w5]. Aujourd'hui, le domaine d'application ne se limite plus seulement à GIMP car utilisé également dans d'autres projets. Ainsi, l'environnement GNOME (GNU Network Object Model Environment) est basé sur GTK+.

Avantages et inconvénients de Gtk.

L'utilisation de GTK+ pour la création de GUI est très intéressante sur plusieurs points :

- GTK+ est sous licence GNU LGPL. Cela fait de GTK+ une bibliothèque libre, permettant ainsi de l'utiliser ou de la modifier sans aucune contrainte financière;
- GTK+ existe sur plusieurs plates-formes. En effet, GTK+ fonctionne sur les plates-formes linux, Windows, Macintosh;
- GTK+ est utilisable avec plusieurs langages de programmation. Même si les créateurs de GTK+ ont écrit cette bibliothèque en C, sa structure orientée objet et sa licence ont permis à d'autres développeurs d'adapter GTK+ à leur langage préféré. Ainsi, il est maintenant possible de programmer des GUI GTK+ en C, C++, Ada, Perl, Python, PHP et bien d'autres langages.

Comme de nombreuses autres bibliothèques de la même catégorie, il existe des documents sur Internet qui décrivent Gtk. Cependant, cette documentation pléthorique est mal organisée c'est-à-dire qu'il n'en existe pas une sur la toile qui décrit au complet toutes ses fonctionalités.

Conséquence: documentation très découpée exigeant le nouveau lecteur à passer beaucoup de temps pour rassembler toutes les informations.

1.3.7 wxWidget.

wxWidgets (anciennement wxWindows) est une bibliothèque graphique libre utilisée comme boîte à outils de programmation d'interface utilisateur multiplate-formes[w6].

Description de wxWidget.

À la différence d'autres boîtes à outils qui tentent de restituer une interface utilisateur identique sur toutes les plateformes, wxWidgets restitue des abstractions similaires, mais avec l'apparence native de chaque environnement cible; ce qui est moins dépaysant pour les utilisateurs finaux. wxWidgets est disponible entre autres pour Macintosh, GNU/Linux et Unix, Microsoft Windows, ainsi que pour du matériel embarqué sous GNU/Linux ou Windows CE.

Avantages et inconvénients de wxWidget.

wxWidgets est diffusée sous licence wxWidgets License, similaire à la licence LGPL, avec pour différence cependant qu'une compilation statique n'impose pas que le programme soit également sous license LGPL. La bibliothèque originale est écrite en C++ mais il existe de nombreux binding (liaison entre deux éléments) vers les langages de programmation courants: Python – wxPython, Perl – wxPerl, BASIC – wxBasic, Lua – wxLua, OCaml – wxCaml, JavaScript – wxJavaScript, Java - wxJava ou wx4j, Ruby – wxRuby, Eiffel – wxEiffel, Haskell - wxHaskell, C#/.NET – wx.NET, Euphoria – wxEuphoria, D – wxD.
Certains sont plus développés que d'autres les plus populaires étant wxPython, wxPerl et wxBasic.
Sous le nom «wx», wxWidgets est la base de l'interface utilisateur des applications développées avec C++BuilderX, de Borland.

1.3.8 Qt.

Qt est un framework C++ permettant de développer des applications GUI multiplates-formes en se basant sur l'approche suivante: «Ecire une fois, compiler n'importe où». Qt permet aux programmeurs d'employer une seule arborescence source pour des applications qui s'exécuteront sous Windows 98, à Xp, Mac OS X, Linux, Solaris, HP-UX et de nombreuses autres versions d'unix avec X11. Les bibliothèques et outils Qt font également partis de Qtopia Core, un produit qui propose son propre système de fenêtre au-dessus de Linux Embarqué.[b2]

Description de Qt.

Qt a bâti sa réputation de framework multiplate-forme mais en raison de son API intuitive et puissante, de nombreuses organisations l'utilisent pour un développement sur une seule plate-forme. Adobe Photoshop Album est un exemple d'application Windows de masse écrit en Qt. De mutilples logiciels sophistiqués d'entreprise, comme les outils d'animation 3D, le traitement de films numériques, la conception automatisée électronique (pour concevoir des circuits), l'exploration de pétrole de gaz, les services financiers et l'imagerie médicale sont gérés avec Qt. Un utilisateur qui travaille sur une application écrite en Qt peut facilement changer d'environnement simplement grâce à la recompilation.

Avantages et inconvénients de Qt.

Qt est disponible sous diverses licences. Si vous devez concevoir des applications commerciales, vous devez acheter une licence Qt commerciale; si vous voulez générer des programmes open source, vous avez la possibilité d'utiliser l'édition open source (GPL). Qt constitue la base sur laquelle sont conçus l'environnement KDE (K Desktop Environment) et de nombreuses applications open sources qui y sont liées.

1.4 Choix du langage de programmation.

Parmi les langages implémentant le concept orienté objet, notre choix s'est porté sur le C++.

Pourquoi le C++?

En matière de conception des objets graphiques capables d'interagir entre eux, il existe des langages standards dont le C++. Ce choix permet de prendre en compte le code existant de notre application auquel sera greffé le nouveau code.

1.5 Justification du choix de Qt.

Avec Qt comme bibliothèque graphique, nous profiterons à la fois de son aspect portable, de son concept puissant, de son intuitivité et de sa facilité d'installation pour toutes les catégories d'utilisateurs (un simple utilisateur pourra l'installer sans rencontrer beaucoup de difficultés). Notons par ailleurs que la documentation de Qt est très organisée[w8]. Outre sa capacité de créer et gérer des plug-in, il est possible de créer des applications qui pourront communiquer en réseau ou de s'interfacer avec une base de données telle que MySQL.

1.6 Qt et le modèle MVC.

Nous attaquons maintenant l'une des parties les plus intéressantes sur Qt, mais aussi l'une des plus délicates.

Nous apprendrons à manipuler 3 widgets complexes :

- ✓ QListView : une liste d'éléments à un seul niveau.
- ✓ QTreeView : une liste d'éléments à plusieurs niveaux, organisée en arbre.
- ✓ QTableView : un tableau.

On ne peut pas utiliser ces widgets sans un minimum de théorie. Nous allons découvrir l'architecture MVC, une façon de programmer (on parle de design pattern) très puissante qui va nous donner énormément de flexibilité.

Qu'est-ce que l'architecture MVC ? A quoi ça sert ? Quel rapport avec notre projet?

MVC est l'abréviation de Model-View-Controller, ce qui signifie en français : "Modèle-Vue-Contrôleur". Il s'agit d'un design pattern, une technique de programmation bien pensée visant à rendre le code plus lisible, plus clair et plus souple. L'architecture MVC propose de séparer les éléments du programme en 3 parties :

- ✓ *Le modèle* : c'est la partie qui contient les données. Le modèle peut par exemple contenir la liste des élèves d'une classe, avec leurs noms, prénoms, âges, etc...
- ✓ *La vue* : c'est la partie qui s'occupe de l'affichage. Elle affiche ce que contient le modèle. Par exemple, la vue pourrait être un tableau. Ce tableau affichera la liste des élèves si c'est ce que contient le modèle.
- ✓ *Le contrôleur* : c'est la partie "réflexion" du programme. Lorsque l'utilisateur sélectionne 3 élèves dans le tableau et appuie sur le bouton "Supprimer", le contrôleur est appelé et se charge de supprimer les 3 élèves du modèle.

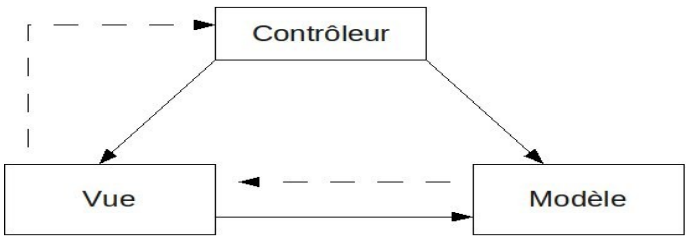

Figure 2: Architecture MVC.

Le contrôleur est intégré à la vue avec Qt. Cela signifie qu'il faudra connaître toutes les fonctionalités de Qt pour pouvoir apporter une quelconque modification à ce contrôleur. Ainsi, avec l'atténuation de la complexité du modèle MVC, les données sont toujours séparées de leur affichage et on évite au programmeur d'avoir à gérer les 3 parties.

Qt adapte ce principe à sa manière, pour garder les bonnes idées principales sans pour autant obliger le programmeur à "trop" découper le code; ce qui aurait pu être complexe et répétitif à la longue.

Les classes gérant le modèle de Qt.

Il y a plusieurs types de modèles différents. Deux (2) possibilités sont offertes :

- ✓ **Créer sa propre classe de modèle**. Il faut alors que cette dernière hérite de QAbstractItemModel. C'est la solution la plus flexible.
- ✓ **Utiliser une des classes génériques toutes prêtes offertes par Qt** :
  - ➔ *QStringListModel*: une liste de chaînes de caractères, de type QString. Très simple, très basique. Ca peut suffire pour les cas les plus simples.
  - ➔ *QStandardItemModel*: une liste d'éléments organisés sous forme d'arbre (chaque élément peut contenir des sous-éléments). Ce type de modèle est plus complexe que le précédent, car il gère plusieurs niveaux d'éléments. Avec QStringListModel, c'est un des modèles les plus utilisés.
  - ➔ *QDirModel*: la liste des fichiers et dossiers stockés sur l'ordinateur. Ce modèle va analyser en arrière-plan le disque dur, et restitue la liste des fichiers sous la forme d'un modèle prêt à l'emploi.
  - ➔ *QSqlQueryModel*, *QSqlTableModel* et *QSqlRelationalTableModel*: données issues d'une base de données. On peut s'en servir pour accéder à une base de données (MySQL, Oracleetc...).

Toutes ces classes proposent donc des modèles prêts à l'emploi, qui héritent de QabstractItemModel. Si aucune de ces classes ne vous convient, vous devrez créer votre propre classe en héritant de QAbstractItemModel.

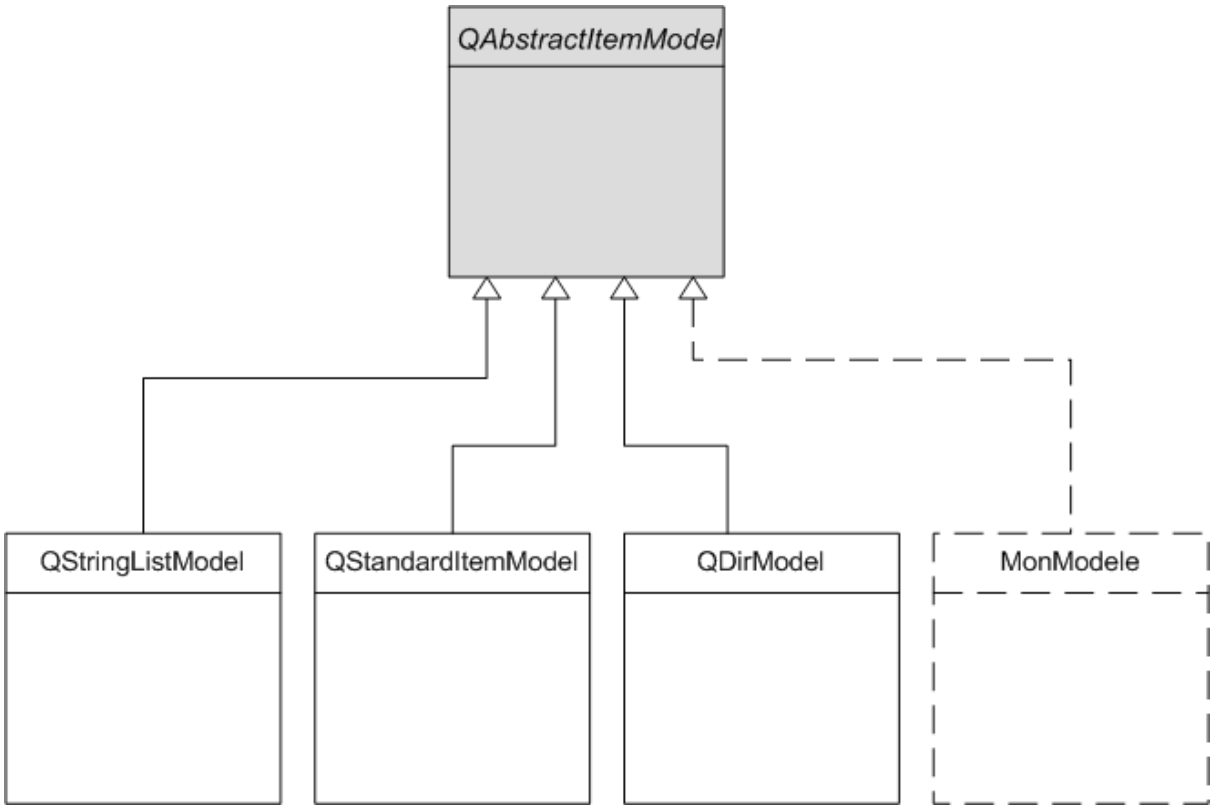

Figure 3: Quelques classes qui gérent le modèle de Qt.

Les classes gérant la vue.

Pour afficher les données issues du modèle, il nous faut une vue. Avec Qt, la vue est un widget, puisqu'il s'agit d'un affichage dans une fenêtre.

Tous les widgets de Qt ne sont pas bâtis autour de l'architecture modèle/vue. On compte 3 widgets adaptés pour la vue avec Qt :

- ✓ QListView : une liste d'éléments.

Figure 4: Une liste d'éléments

- ✓ QTreeView : un arbre d'éléments, où chaque élément peut avoir des éléments enfants.

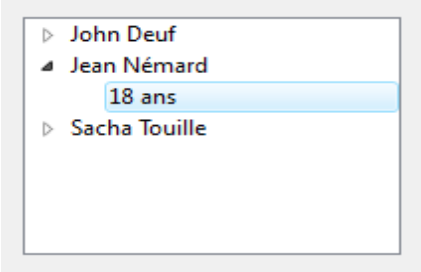

Figure 5: Un arbre d'éléments.

- ✓ QTableView : un tableau d'éléments.

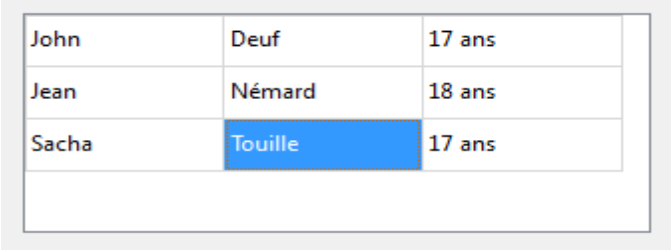

Figure 6: un tableau d'éléments.

Appliquer un modèle à la vue.

L'architecture modèle/vue de Qt, oblige le programmeur à diviser le travail en trois (3) temps. Pour cela il faut créer le modèle, créer la vue et associer la vue et le modèle. La dernière étape est essentielle. Cela revient en quelque sorte à "connecter" notre modèle à notre vue. Si on ne donne pas de modèle à la vue, elle ne saura pas quoi afficher, donc elle n'affichera rien.

La connexion se fait toujours avec la méthode setModel() de la vue :

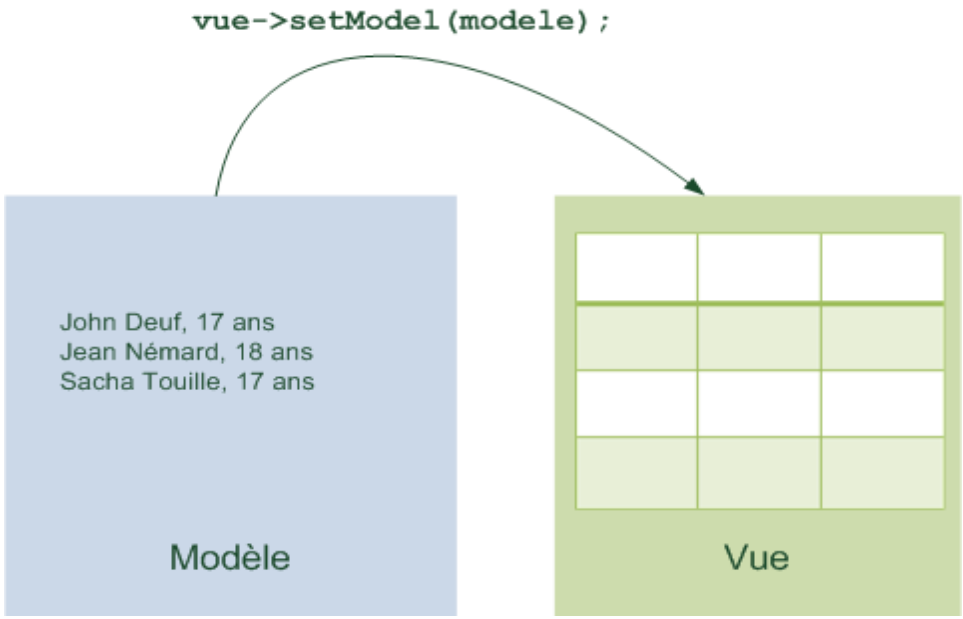

Figure 7: Application d'un modèle à une vue.

Ce système présente l'avantage d'être flexible. On peut avoir deux (2) modèles et appliquer soit l'un soit l'autre à la vue en fonction des besoins. Dès que l'on modifie le modèle, la vue affiche instantanément les nouveaux éléments.

Plusieurs modèles ou plusieurs vues.

Imaginons que l'on ait 2 modèles : un qui contient une liste d'élèves, un autre qui contient une liste de capitales avec leur pays. Notre vue peut afficher soit l'un, soit l'autre :

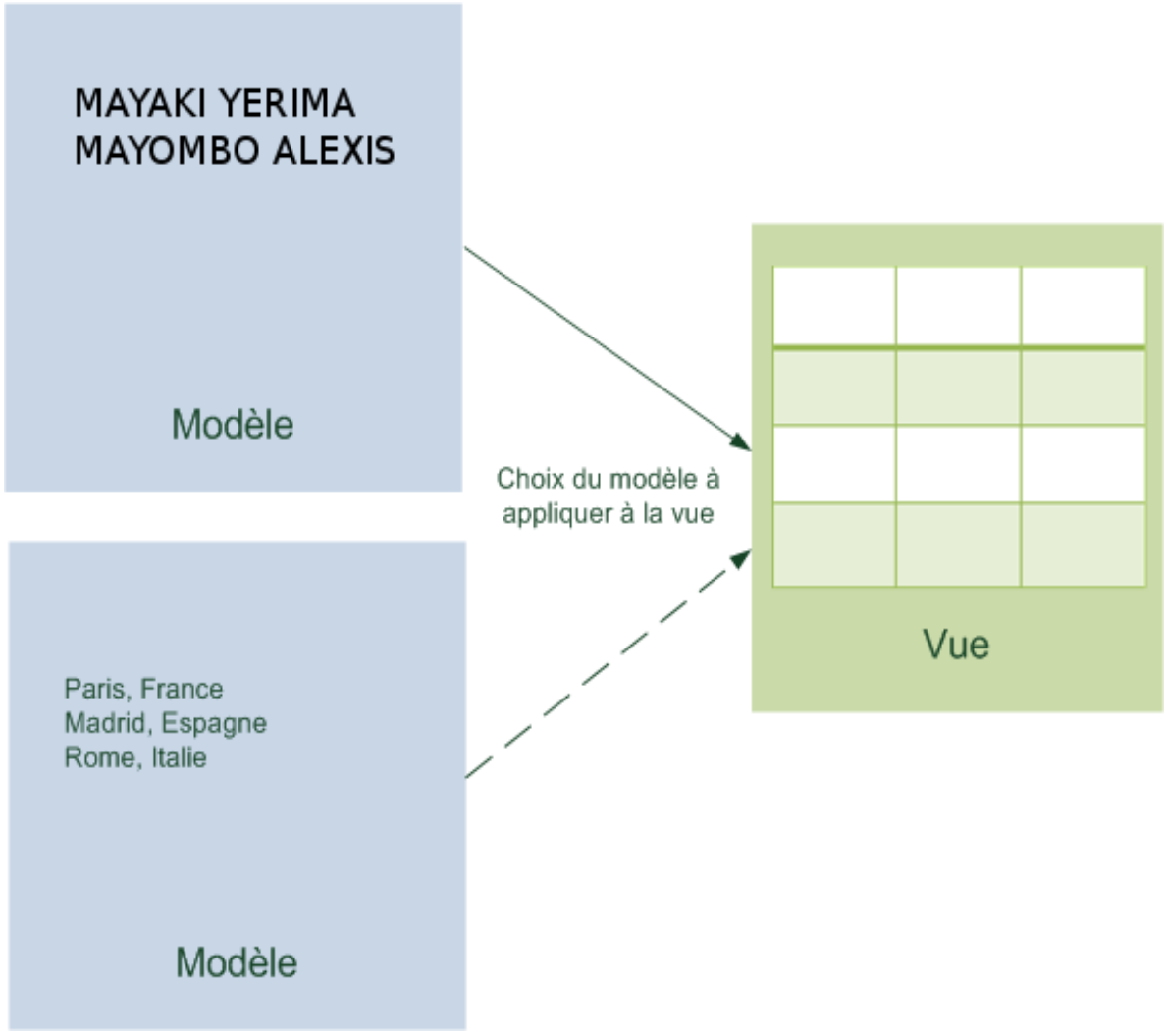

Figure 8: Plusieurs modèles et une vue.

Il faut faire un choix: une vue ne peut afficher qu'un seul modèle à la fois. L'avantage, c'est qu'au besoin on peut changer le modèle affiché par la vue en cours d'exécution, en appelant juste la méthode setModel().

Imaginons le cas inverse. On a un modèle, mais plusieurs vues. Cette fois, rien empêche d'appliquer ce modèle à 2 vues en même temps:

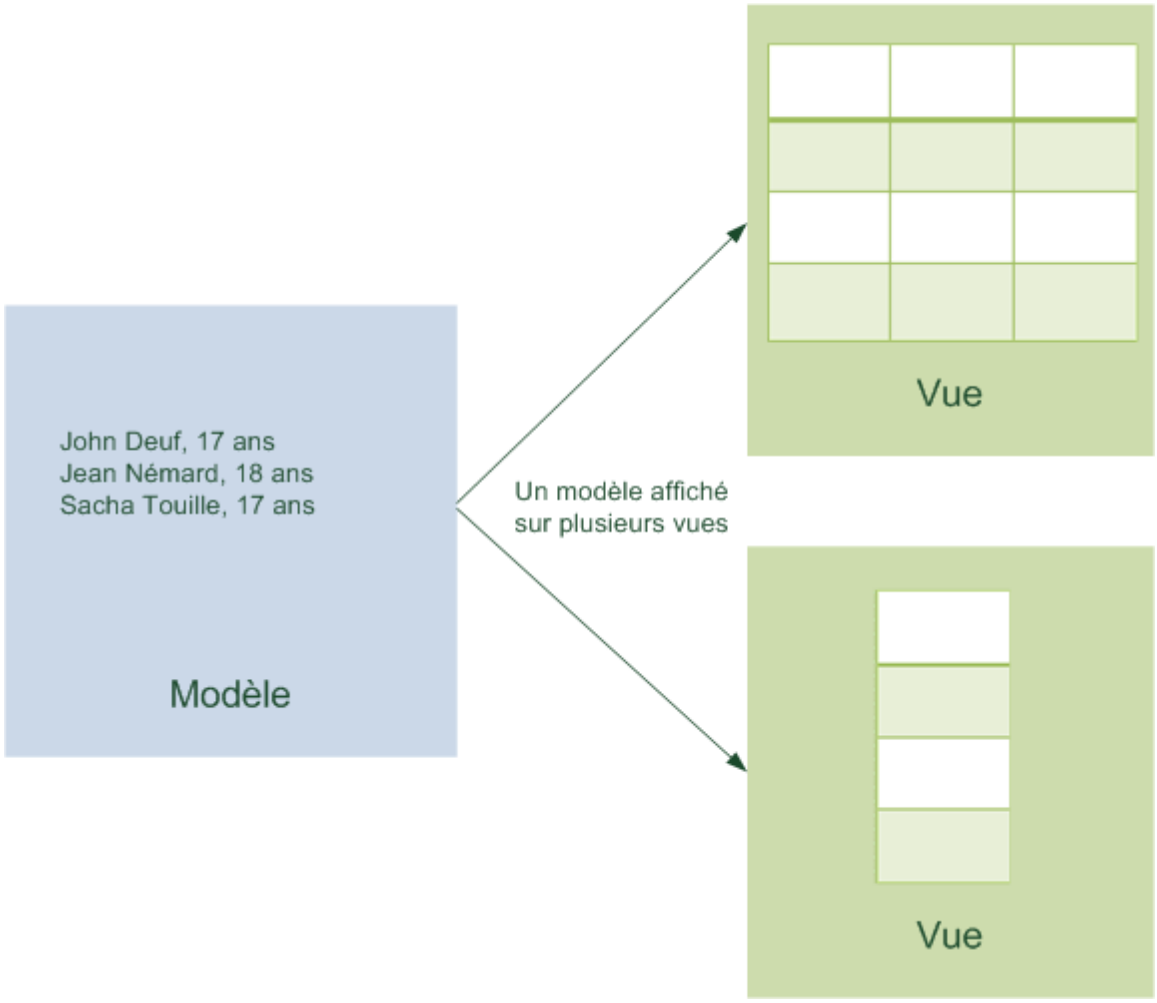

Figure 9: Un modèle pour plusieurs vues.

On peut ainsi visualiser le même modèle de 2 façons différentes (ici sous forme de tableau ou de liste dans mon schéma). Comme le même modèle est associé à 2 vues différentes, si le modèle change alors les 2 vues changent en même temps.

Utilisation d'un modèle simple.

Nous allons étudier un cas simple de l'architecture modèle/vue de Qt : QDirModel.

Sa particularité, c'est qu'il est très simple à utiliser. Il analyse votre disque dur et génère le modèle correspondant. Pour créer ce modèle :

QDirModel *modele = new QDirModel;

Il existe désormais un modèle qui représente notre disque. On va l'appliquer à une vue. Toutes les vues peuvent afficher n'importe quel modèle. C'est toujours compatible. Par contre, certaines sont plus adaptées que d'autres en fonction du modèle utilisé. Par exemple, pour un QDirModel la vue la plus adaptée est l'arbre (QTreeView). Nous essaierons toutefois toutes les vues avec ce modèle pour comparer le fonctionnement.

Le modèle appliqué à un QTreeView.

```
#include "MaClasse.h"

MaClasse::MaClasse()
{
    QVBoxLayout *layout = new QVBoxLayout;

    QDirModel *modele = new QDirModel;
    QTreeView *vue = new QTreeView;
    vue->setModel(modele);

    layout->addWidget(vue);

    setLayout(layout);
}
```

On crée le modèle: QDirModel *modele = new QdirModel;

puis la vue: QTreeView *vue = new QtreeView;

et on dit à la vue "Utilise ce modèle pour savoir quoi afficher": vue->setModel(modele);

Le résultat est le suivant:

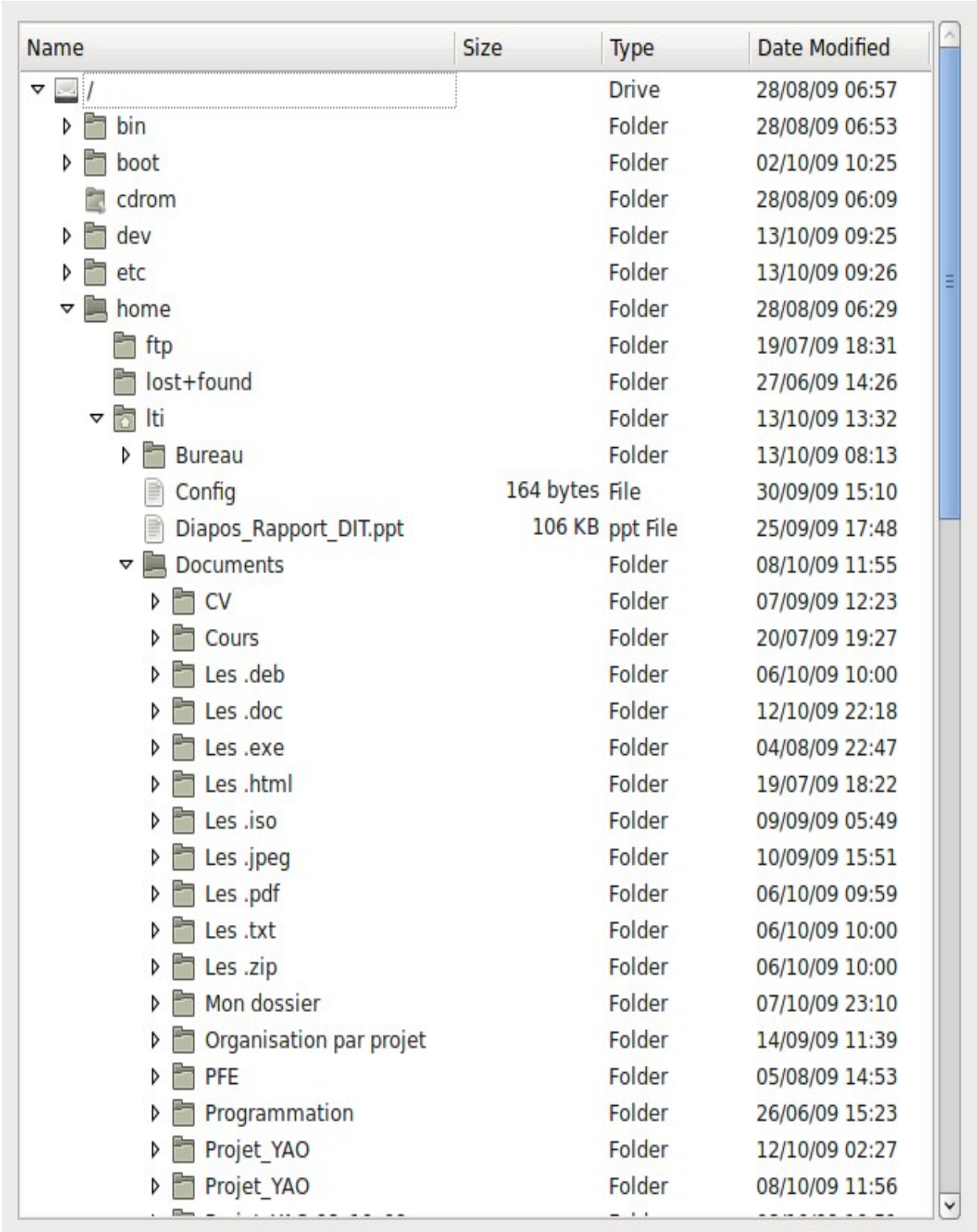

Figure 10: Un exemple d'objet QTreeView.

Le modèle appliqué à un QListView.

Essayons de faire la même chose, mais avec une liste (QListView). On garde le même modèle, mais on l'applique à une vue différente :

```
#include "FenPrincipale.h"

FenPrincipale::FenPrincipale()
{
    QVBoxLayout *layout = new QVBoxLayout;

    QDirModel *modele = new QDirModel;
    QListView *vue = new QListView;
    vue->setModel(modele);

    layout->addWidget(vue);

    setLayout(layout);
}
```

Voici le résultat:

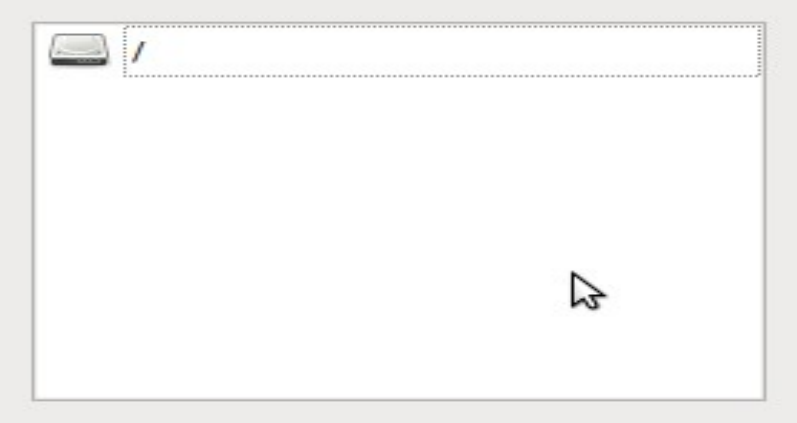

Figure 11: Un exemple d'objet QListView.

Ce type de vue ne peut afficher qu'un seul niveau d'éléments à la fois. Cela explique pourquoi on voit uniquement la liste des disques (ici un seul disque) et pourquoi on ne peut pas afficher leur contenu.

La vue avec QListView affiche ce qu'elle est capable d'afficher, c'est-à-dire le premier niveau d'éléments. D'où la preuve qu'un même modèle marche sur plusieurs vues différentes, mais que certaines vues sont plus adaptées que d'autres.

Le modèle appliqué à un QTableView.

Tout comme QlisteView, le tableau ne peut pas afficher plusieurs niveaux d'éléments (seul l'arbre QTreeView peut le faire). Par contre, il peut afficher plusieurs colonnes :

```
#include "FenPrincipale.h"

FenPrincipale::FenPrincipale()

{
   QVBoxLayout *layout = new QVBoxLayout;
   QDirModel *modele = new QDirModel;

   QTableView *vue = new QTableView;

   vue->setModel(modele);
   layout->addWidget(vue);
   setLayout(layout);
}
```

Voici le résultat:

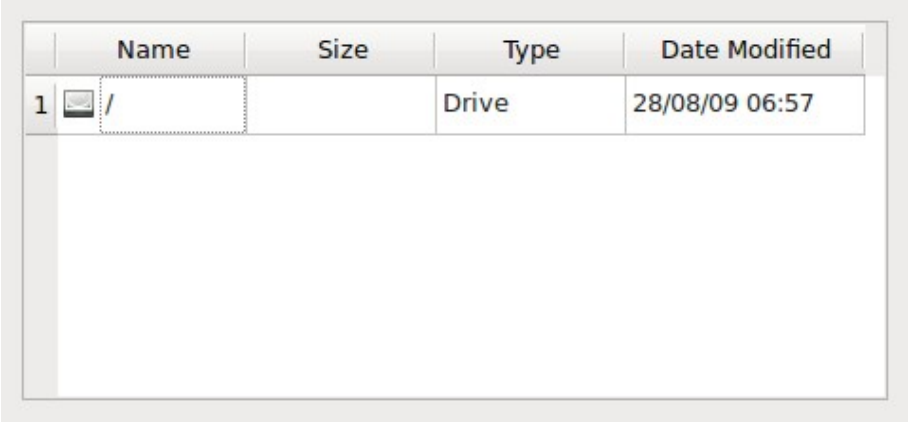

Figure 12: Un exemple d'objet QTableView.

Utilisation de modèles personnalisables.

Le modèle QDirModel que nous venons de voir était très simple à utiliser. Rien à paramétrer, rien à configurer, il analyse automatiquement le disque dur pour construire son contenu.

Cependant, on peut vouloir utiliser ses propres données, son propre modèle. C'est ce que nous allons voir ici, à travers 2 nouveaux modèles :

- ✓ QStringListModel : une liste simple d'éléments de type texte, à un seul niveau.
- ✓ QStandardItemModel : une liste plus complexe à plusieurs niveaux et plusieurs colonnes, qui peut convenir dans la plupart des cas.

Pour les cas simples, nous utiliserons donc QStringListModel, mais nous découvrirons aussi QStandardItemModel qui nous donne plus de flexibilité.

QStringListModel : une liste de chaînes de caractères QString.

Ce modèle, très simple, permet de gérer une liste de chaînes de caractères. Par exemple, si l'utilisateur doit choisir son programme TV parmi une liste:

- ✓ Documentaire,
- ✓ Films romantiques,
- ✓ Films d'action,
- ✓ Films d'horreur,
- ✓ Série,
- ✓ etc...

Pour construire ce modèle, il faut procéder en 2 temps :

- ✓ Construire un objet de type QStringList, qui contiendra la liste des chaînes.
- ✓ Créer un objet de type QStringListModel et envoyer à son constructeur le QStringList que vous venez de créer pour l'initialiser.

QStringList surcharge l'opérateur "<<" pour vous permettre d'ajouter des éléments à l'intérieur.

Toutefois il est aussi possible d'utiliser la méthode append().

Un exemple de code:

```
#include "FenPrincipale.h"

FenPrincipale::FenPrincipale() {
    QVBoxLayout *layout = new QVBoxLayout;
    QStringList listeFilms;

    listeFilms << "Documentaire" << "Films romantiques" << "Films d'action" << "Films
d'horreur"<< "Serie";

   // ou encore listeFilms.append("Documentaire"); etc...

    QStringListModel *modele = new QStringListModel(listeFilms);

    QListView *vue = new QListView;

    vue->setModel(modele);
    layout->addWidget(vue);
    setLayout(layout);
}
```

Voici le résultat:

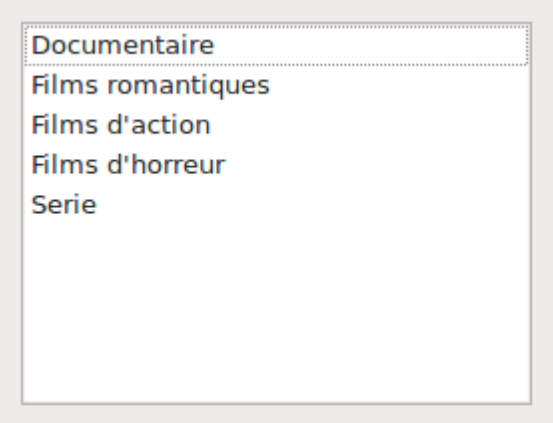

Figure 13 : Un exemple d'objet QStringListModel.

QStandardItemModel : un liste à plusieurs niveaux et plusieurs colonnes.

Ce type de modèle est beaucoup plus complet que le précédent. Il permet de créer tous les types de modèles possibles et imaginables.

Pour bien visualiser les différents types de modèles que l'on peut concevoir avec un QStandardItemModel, voici un schéma :

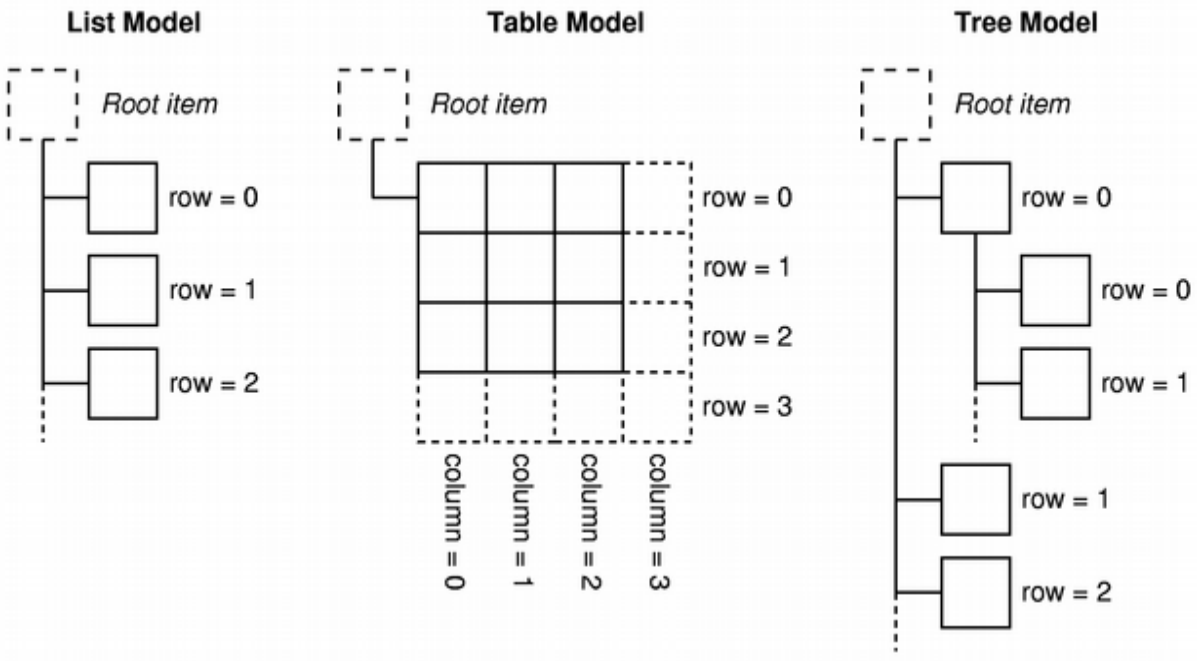

Figure 14 : Schéma complet de QstandardItemModel.

List Model : c'est un modèle avec une seule colonne. Pas de sous-éléments. C'est le modèle utilisé par QStringList, mais QStandardItemModel peut aussi le faire. Ce type de modèle est en général adapté à un QListView.

Table Model : les éléments sont organisés avec plusieurs lignes et colonnes. Ce type de modèle est en général adapté à un QTableView.

Tree Model : les éléments ont des sous-éléments, ils sont organisés en plusieurs niveaux. Ce type de modèle est en général adapté à un QTreeView.

Gérer plusieurs lignes et colonnes.

Pour construire un QStandardItemModel, on doit indiquer en paramètres le nombre de lignes et de colonnes qu'il doit gérer. Des lignes et des colonnes supplémentaires peuvent toujours être ajoutées par la suite au besoin.

```
FenPrincipale::FenPrincipale()
{
    QVBoxLayout *layout = new QVBoxLayout;

    QStandardItemModel *modele = new QStandardItemModel(5, 3);

    QTableView *vue = new QTableView;
    vue->setModel(modele);

    layout->addWidget(vue);

    setLayout(layout);
}
```

Ici, un modèle à 5 lignes et 3 colonnes sera créé. Les éléments sont vides au départ, mais on a déjà un tableau :

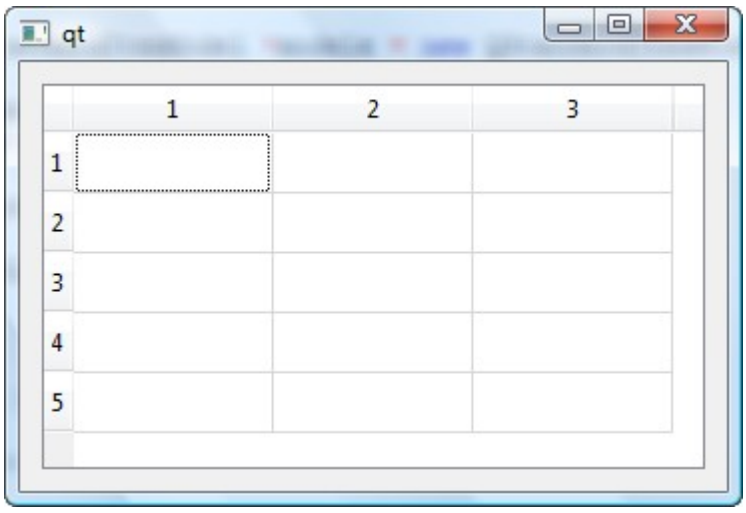

Figure 15: Un exemple d'objet QStandardItemModel.

On peut aussi appeler le constructeur par défaut (sans paramètres) si on ne connaît pas du tout la taille du tableau à l'avance. Il faudra appeler appendRow() et appendColumn() pour ajouter respectivement une nouvelle ligne ou une nouvelle colonne.

Chaque élément est représenté par un objet de type QstandardItem. Pour définir un élément, la méthode setItem() du modèle est à utiliser. Donnez-lui respectivement le numéro de ligne, de colonne, et un QStandardItem à afficher. La numérotation commence à 0.

Ajouter des éléments enfants.

Il faut procéder dans l'ordre :

1. Créer un élément (par exemple "item"), de type QStandardItem.
2. Ajouter cet élément au modèle avec appendRow().
3. Ajouter un sous-élément à "item" avec appendRow().

```
FenPrincipale::FenPrincipale()

{
    QVBoxLayout *layout = new QVBoxLayout;
    QStandardItemModel *modele = new        QStandardItemModel;
    QStandardItem *item = new QStandardItem("MAYAKI");

    modele->appendRow(item);
    item->appendRow(new QStandardItem("60 ans"));
    QTreeView *vue = new QTreeView;
    vue->setModel(modele);
    layout->addWidget(vue);
    setLayout(layout);
}
```

Résultat:

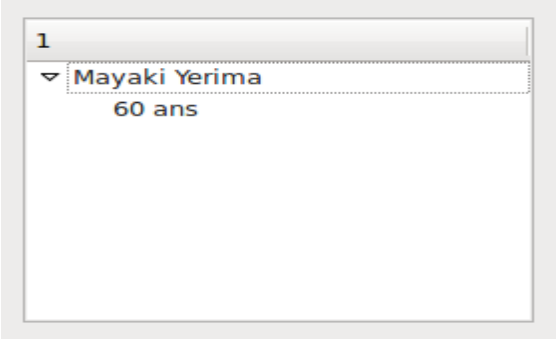

Figure 16: Un exemple d'objet QtreeView dérivant de QStandardItemModel.

Gestion des sélections.

Nous avons découvert comment associer un modèle à une vue, et comment manipuler plusieurs modèles: QDirModel, QStringListModel et QStandardItemModel.

Il nous reste à voir comment on peut récupérer le ou les éléments sélectionnés dans la vue, pour savoir quel est le choix de l'utilisateur.

L'architecture modèle/vue de Qt est extrêmement flexible, mais en contrepartie il est beaucoup plus délicat de s'en servir car il y a plusieurs étapes à suivre dans un ordre précis. Afin d'éviter de faire un chapitre beaucoup trop long et surtout trop complexe, nous nous limiterons les exemples ici aux sélections d'un QListView. L'adaptation à d'autres exemples est facile une fois que le principe est compris.

Nous allons rajouter un bouton "Afficher la sélection" à notre fenêtre. Elle va ressembler à ceci :

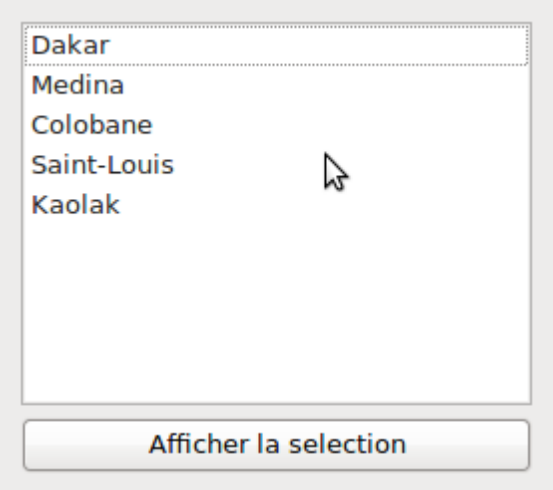

Figure 17 : Affichage et choix à l'aide d'un bouton.

Lorsqu'on cliquera sur le bouton, il ouvrira une boîte de dialogue (QMessageBox) qui affichera le nom de l'élément sélectionné.

Nous allons apprendre à gérer 2 cas :

- ✓ Lorsqu'on ne peut sélectionner qu'un seul élément à la fois.
- ✓ Lorsqu'on peut sélectionner plusieurs éléments à la fois.

Une sélection unique.

Nous allons devoir créer une connexion entre un signal et un slot pour que le clic sur le bouton fonctionne.

Modifions donc pour commencer le .h de la fenêtre :

```
#ifndef HEADER_FENPRINCIPALE
#define HEADER_FENPRINCIPALE

#include <QtGui>

class FenPrincipale : public QWidget {

   Q_OBJECT
   public:
         FenPrincipale();
   private:
      QListView *vue;
      QStringListModel *modele;
      QPushButton *bouton;
   private slots:
      void clicSelection(); };
#endif
```

Modifions également le fichier .cpp:

```
#include "FenPrincipale.h"
FenPrincipale::FenPrincipale() {
    QVBoxLayout *layout = new QVBoxLayout;
    QStringList listeRegions;
    listePays << "Dakar" << "Medina" << "Colobane" << "Saint-Louis" <<
"Kaolak";
    modele = new QStringListModel(listeRegions);
    vue = new QListView ;
    vue->setModel(modele);
    bouton = new QPushButton("Afficher la sélection");
    layout->addWidget(vue);
    layout->addWidget(bouton);
    setLayout(layout);
    connect(bouton, SIGNAL(clicked()), this, SLOT(clicSelection())); }

void FenPrincipale::clicSelection() {
   QItemSelectionModel *selection = vue->selectionModel();
   QModelIndex indexElementSelectionne = selection->currentIndex();
   QVariant elementSelectionne = modele->data(indexElementSelectionne,
Qt::DisplayRole);
    QMessageBox::information(this, "Elément sélectionné",
elementSelectionne.toString()); }
```

Une sélection multiple.

Par défaut, on ne peut sélectionner qu'un seul élément à la fois sur une liste. Pour changer ce comportement et autoriser la sélection multiple, il faut rajouter ceci dans le constructeur :

```
vue->setSelectionMode(QAbstractItemView::ExtendedSelection);
```

Avec ce mode, on peut sélectionner n'importe quels éléments. On peut utiliser la touche Shift du clavier pour faire une sélection continue, ou Ctrl pour une sélection discontinue (saut de lignes).

Voici un exemple de sélection, désormais possible :

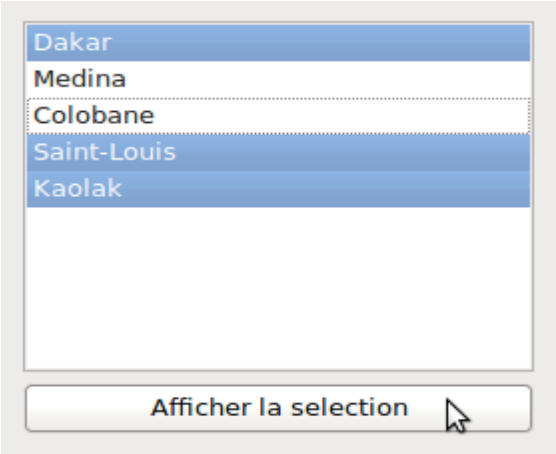

Figure 18 : Sélection multiple.

Pour récupérer la liste des éléments sélectionnés, c'est un peu plus compliqué ici parce qu'il y en a plusieurs. On ne peut plus utiliser la méthode currentIndex(), il va falloir utiliser selectedIndexes(). Nouveau code du slot:

```
void FenPrincipale::clicSelection() {

    QItemSelectionModel *selection = vue->selectionModel();
    QModelIndexList listeSelections = selection->selectedIndexes();
    QString elementsSelectionnes;

 for (int i = 0 ; i < listeSelections.size() ; i++) {
       QVariant elementSelectionne = modele->data(listeSelections[i],
Qt::DisplayRole);
        elementsSelectionnes += elementSelectionne.toString() + "<br />";}
    QMessageBox::information(this, "Eléments sélectionnés",
elementsSelectionnes);}
```

Voici le résultat:

Figure 19: Résultat d'une sélection multiple.

Chapitre 2

Etude de l'existant.

2.1 L'application existante.

YAO est une nouvelle méthode de création de modèles numériques pour l'assimilation de données développée au LOCEAN par l'équipe MMSA. YAO est basée sur la décomposition de systèmes complexes en graphes modulaires. Il s'agit d'un sujet novateur qui s'articule autour de compétences bien établies au sein de l'équipe. Le formalisme proposé est très utilisé en méthodologie neuronale. Son intérêt est d'utiliser les équations discrétisées de la dynamique-physique sous forme de graphe, où chaque maille du domaine est en relation avec des prédécesseurs lui transmettant une information et des successeurs qui reçoivent de l'information de ce point. Les équations envisagées peuvent être de type classique (fonction analytique dérivable) ou des réseaux de neurones. Le formalisme de graphe modulaire permet une modélisation à partir de laquelle a été conçu un outil général (YAO, en version 9 actuellement) permettant à l'utilisateur de se concentrer sur la spécification et de réduire sa part de programmation. Tout code écrit à partir de YAO bénéficie de façon quasi-automatique du code adjoint qui permet de calculer le gradient d'une fonction de coût. Un grand avantage de cette méthode est sa flexibilité et la facilité avec laquelle il est possible de modifier le modèle pour le faire évoluer. La méthodologie a été testée sur différents exemples (modèle Météo ISBA, modèle traceur OPA); elle a montré sa capacité à produire facilement des modèles adjoints et à permettre l'assimilation des données en 4-D VAR (prise en compte des observations à n'importe quel moment durant le temps d'assimilation). YAO peut s'avérer très intéressant pour la communauté des géophysiciens.

2.1.1 Comment installer l'application?

Avant de l'installer sur Mandriva 2008 par exemple, il faut s'assurer que certains paquets soient présents et mis à jour (avec l'outil URPMI par exemple). La mise à jour se fera au cas échéant dans un terminal ayant les droits de root. Copier et coller les lignes suivantes dans le terminal qui sont des liens vers les serveurs principaux de Mandriva 2008:

- ✓ urpmi.addmedia --update plf-free http://ftp.cica.es/mirrors/Linux/plf/mandriva/2008.0/free/release/binary/i586/ with media_info/hdlist.cz
- ✓ urpmi.addmedia plf-free_backports http://ftp.cica.es/mirrors/Linux/plf/mandriva/2008.0/free/backports/binary/i586/ with media_info/hdlist.cz
- ✓ urpmi.addmedia main http://mirror.tuxinator.org/MandrivaLinux/official/2008.0/i586/media/main/release with media_info/hdlist.cz
- ✓ urpmi.addmedia --update main_updates http://mirror.tuxinator.org/MandrivaLinux/official/2008.0/i586/media/main/updates with media_info/hdlist.cz
- ✓ urpmi.addmedia contrib http://mirror.tuxinator.org/MandrivaLinux/official/2008.0/i586/media/contrib/release with media_info/hdlist.cz
- ✓ urpmi.addmedia --update contrib_updates http://mirror.tuxinator.org/MandrivaLinux/official/2008.0/i586/media/contrib/updates with media_info/hdlist.cz

Après ces étapes, le système se met à jour. Installer les paquets suivants: gcc, g++, make, java, gfortran, gnuplot.

A Supposer que le répertoire d'installation se trouve dans /usr/local/yao, nous allons copier notre répertoire yao9 dans ce dernier. Ceci peut se faire avec la commande «cp -r yao9 /usr/local/yao».

2.1.2 Processus de compilation.

A ce stade, il est supposé que le répertoire /usr/local/yao/yao9 existe au préalable. Pour compiler Yao, il faut parcourir trois (3) répertoires. Les étapes à suivre sont les suivantes:

- ✓ Aller jusqu'au répertoire /usr/local/yao/yao9/inria/src et taper dans l'ordre toujours dans le même terminal: make clean, make et make install.
- ✓ Aller dans un deuxième répertoire, le /usr/local/yao/yao9/etc/src/ pour répéter les mêmes instructions dans le même ordre.

Aller dans le troisième et dernier répertoire, le /usr/local/yao/yao9/src/ et taper make install.

2.1.3 Schéma global d'agencement des calculs dans yao.

D'une manière simplifiée, l'utilisateur de YAO doit spécifier trois types de fichiers : le fichier de description, les fichiers sources des modules et le fichier d'instructions.

Le ***fichier de description*** (avec l'extention .d) permet de décrire un projet. Pour cela un langage dédié a été élaboré. Il consiste en des directives qui donnent des indications sur le modèle à implémenter. YAO permet, en particulier, d'implémenter des modèles bi et tri dimensionnels. L'utilisateur doit donc définir la dimension du modèle en espace et en temps, ce qui introduit la notion de maille. La description du graphe du modèle en sera d'autant simplifiée, puisque YAO prend en charge la génération du graphe complet lorsque celui-ci a un caractère répétitif (en espace ou en temps). Ce fichier contient aussi :

- ✓ La déclaration des modules et leurs attributs.
- ✓ La description du graphe, c'est-à-dire la définition des liens entre les sorties et les entrées des modules et l'ordre de calcul des modules ainsi que celui de parcours de l'espace.

Les ***fichiers sources des modules*** sont des simples fichiers (avec l'extention .h) dans lesquels doivent être programmées les différentes fonctions telles, par exemple, que les lois physiques du modèle qui établissent le lien entre les différentes variables et paramètres ainsi que leur Jacobien.

Le ***fichier des instructions*** (avec l'extention .i) intervient lorsque l'utilisateur veut utiliser son modèle pour une application précise (simulation numérique, inversion variationnelle, assimilation variationnelle), il doit spécifier l'enchaînement des tâches qu'il veut simuler. Ces tâches seront décrites à l'aide des instructions de YAO. Par exemple, il pourra spécifier les conditions initiales, la prise en compte des termes d'ébauches, le chargement des observations sur les points d'espace temps où elles sont disponibles, la fonction de coût, et lancer la méthode d'optimisation.

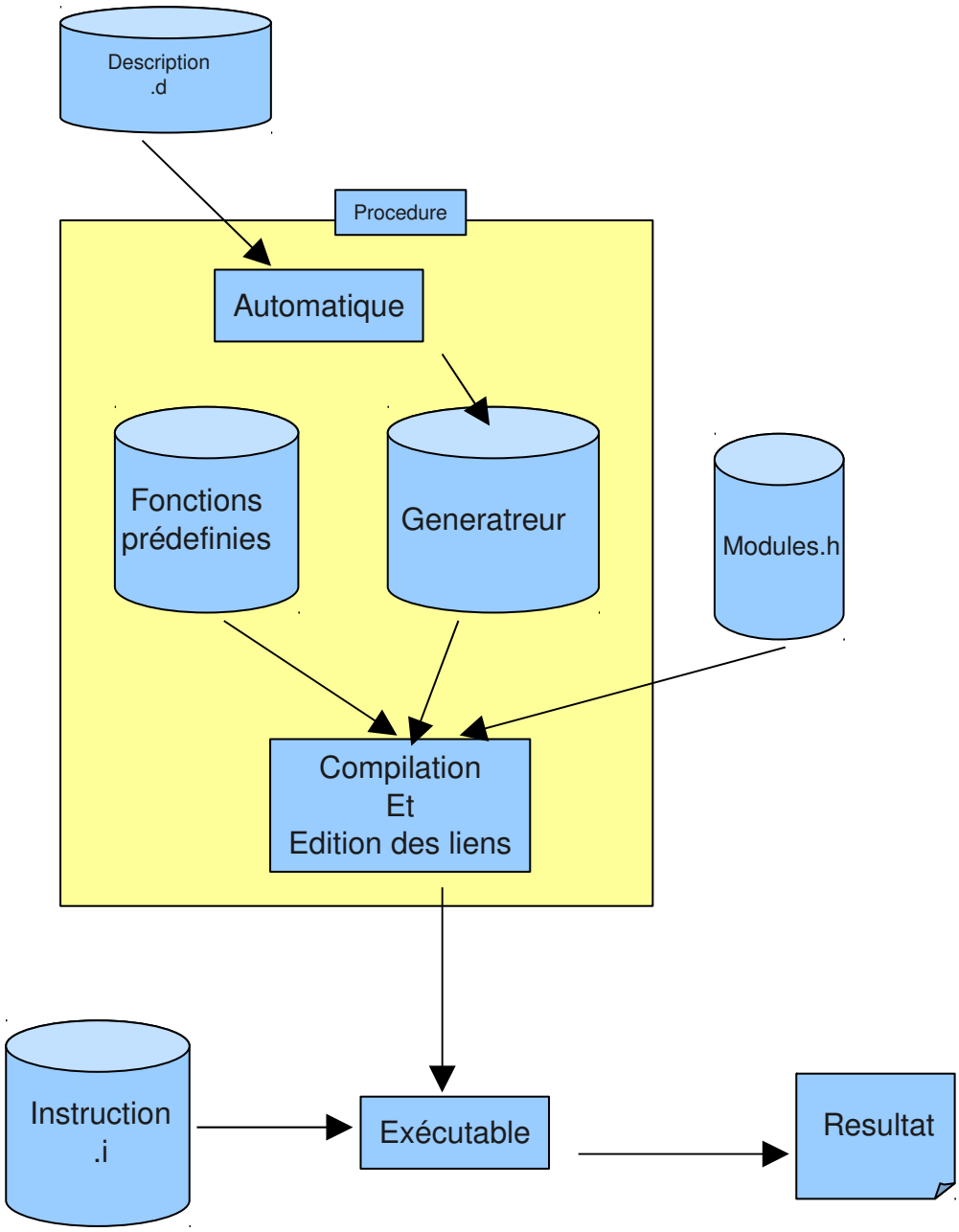

Figure 20: Principe de fonctionnement de Yao.

La figure 20 présente les différentes composantes de la procédure du logiciel YAO : la partie encadrée en couleur jaune correspond aux traitements réalisés par YAO. Les éléments externes représentent les tâches à réaliser par les utilisateurs. YAO tient compte des 3 sortes de fichiers précédents pour générer un exécutable. Faire évoluer un modèle, consiste donc à changer certains modules ou certaines commandes d'instruction et à recompiler pour avoir l'exécutable correspondant qui donne aussi le modèle direct, le tangent linéaire et l'adjoint du modèle que l'on

vient de modifier.

YAO peut s'interfacer avec des procédures déjà existantes : intégrer des réseaux de neurones, faire appel à des optimisateurs particuliers. Il possède des fonctions de tests qui permettent de contrôler l'exactitude des dérivées des différents modules et l'adjoint généré par le modèle.

YAO a déjà été testé sur des applications en océanographie et en hydrologie : code traceur OPA et code ISBA de météo France (laboratoire CETP/IPSL).

Une première étape consiste à la génération de code à partir du fichier de description. Puis ce code généré est intégré au code prédéfini de Yao. L'ensemble est ensuite compilé (avec l'édition des liens) pour produire un exécutable qui finalement interprétera des instructions (en bach ou en interactif) pour produire un résultat.

2.2 Le diagramme de classe existant.

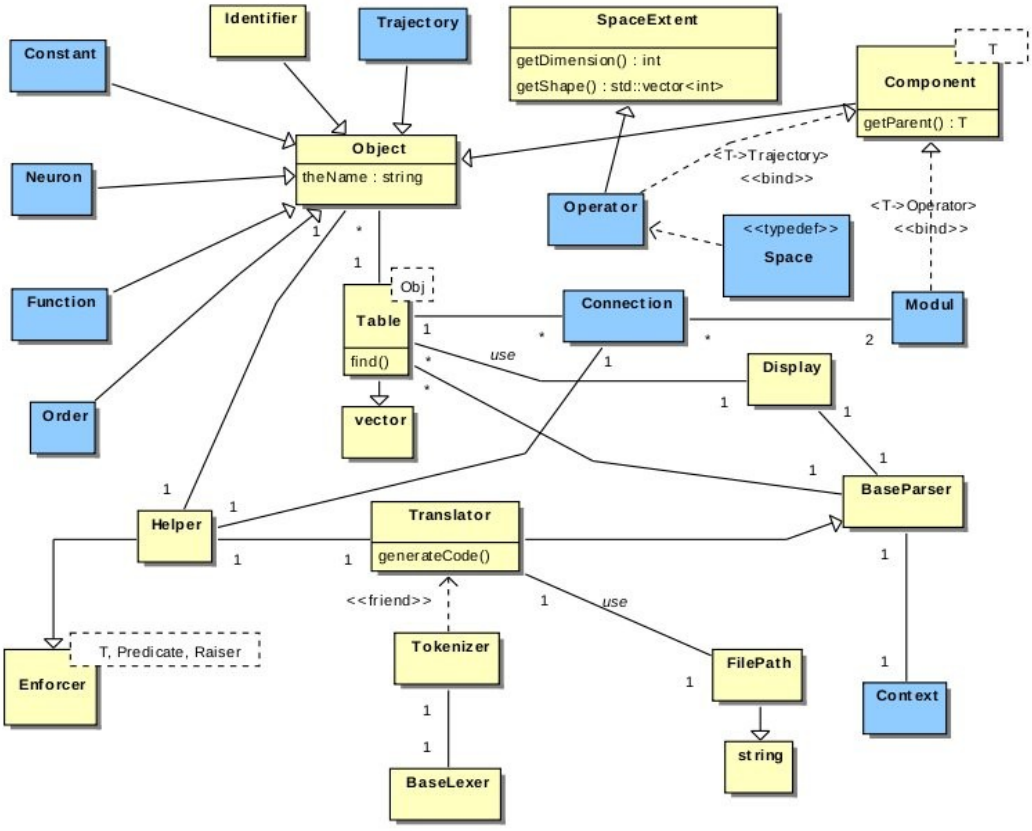

Figure 21: Diagramme de classe de Yao.

2.3 Étude des besoins

Après une première partie d'analyse, nous en arrivons à l'étude des besoins. Cette étude s'avère très intéressante dans la mesure où elle nous permet de canaliser notre objectif. Ces besoins sont à savoir:

- ✓ Les notions en shell linux.
- ✓ Connaissances en modélisation orientée objet.
- ✓ Connaissances en programmation orientée objet.
- ✓ Chercher une bibliothèque graphique capable de générer du code C++

Chapitre 3

Proposition de solutions.

3.1 Proposition d'un modèle UML.

Nous rappelons dans cette partie le modèle orienté objet choisi pour modéliser les besoins et les objectifs que nous nous sommes fixés au départ.

3.1.1 Diagramme des «Use Case».

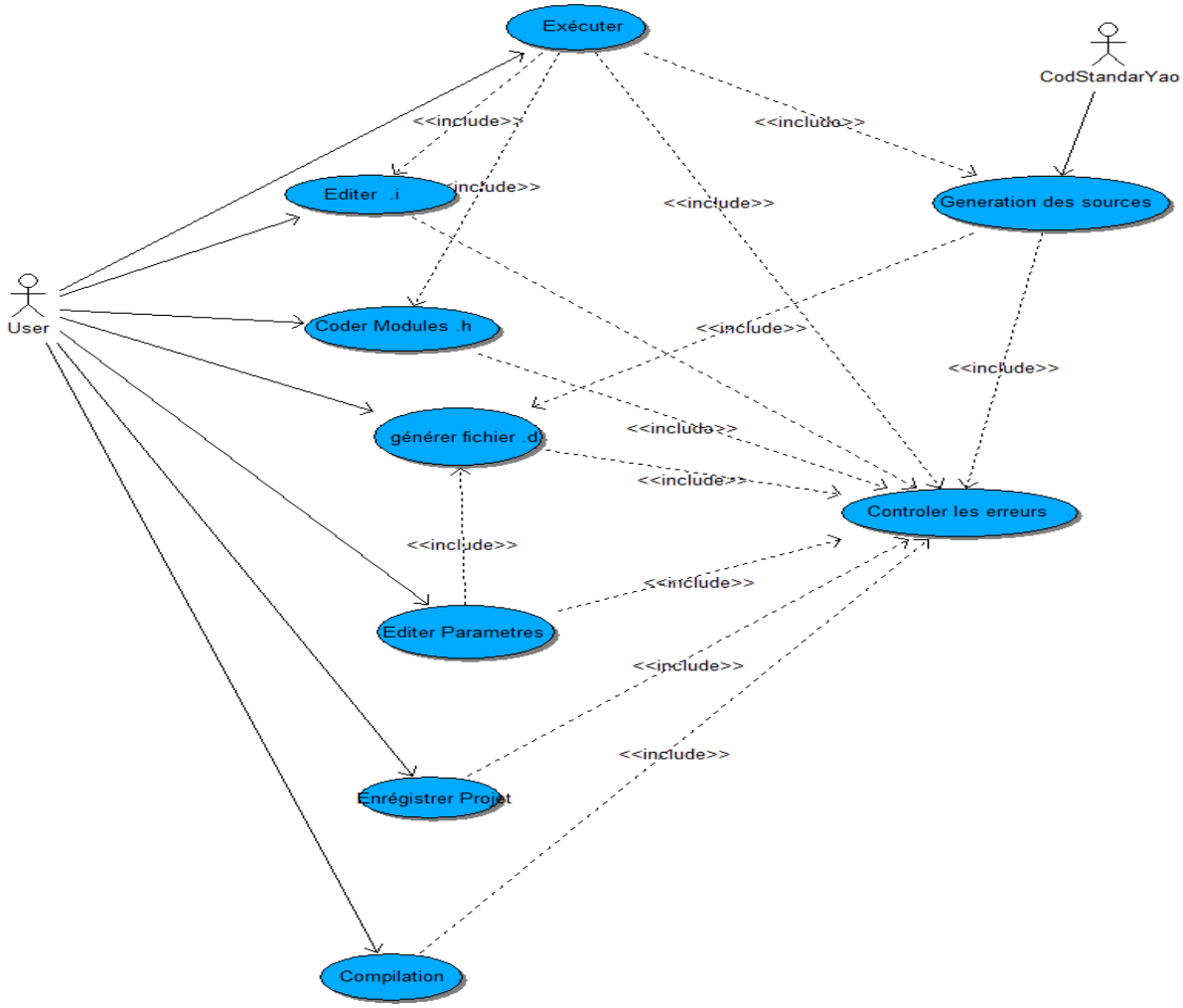

Figure 22 : Diagramme des Use Case.

3.1.2 Diagramme de séquence du Use case «Générer .d».

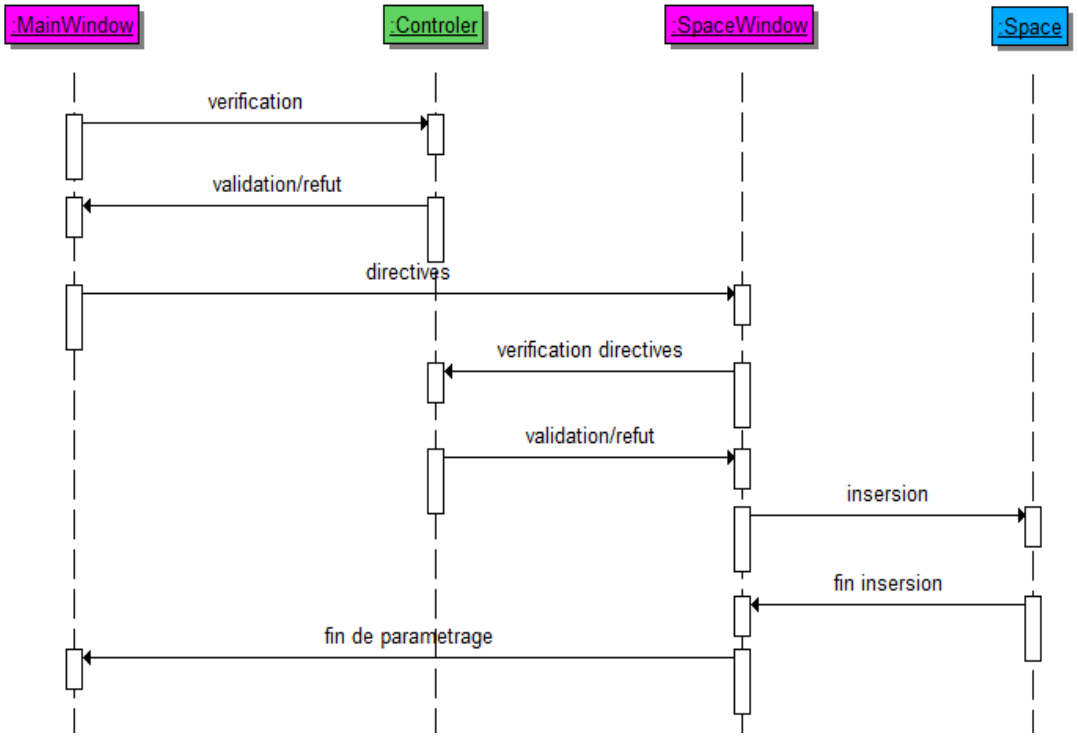

Figure 23: Diagramme de séquence du Use case «Générer .d»

3.1.3 Diagramme de séquence du Use case «Génération des sources»

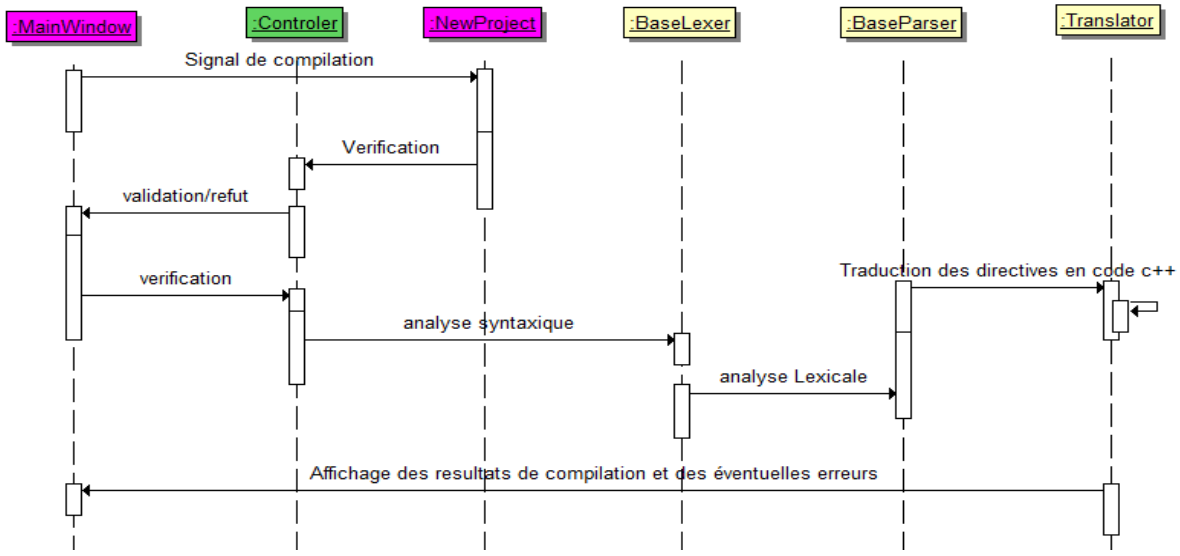

Figure 24: Diagramme de séquence du Use case «Génération des sources»

3.1.4 Diagramme d'activité «Générer .d»

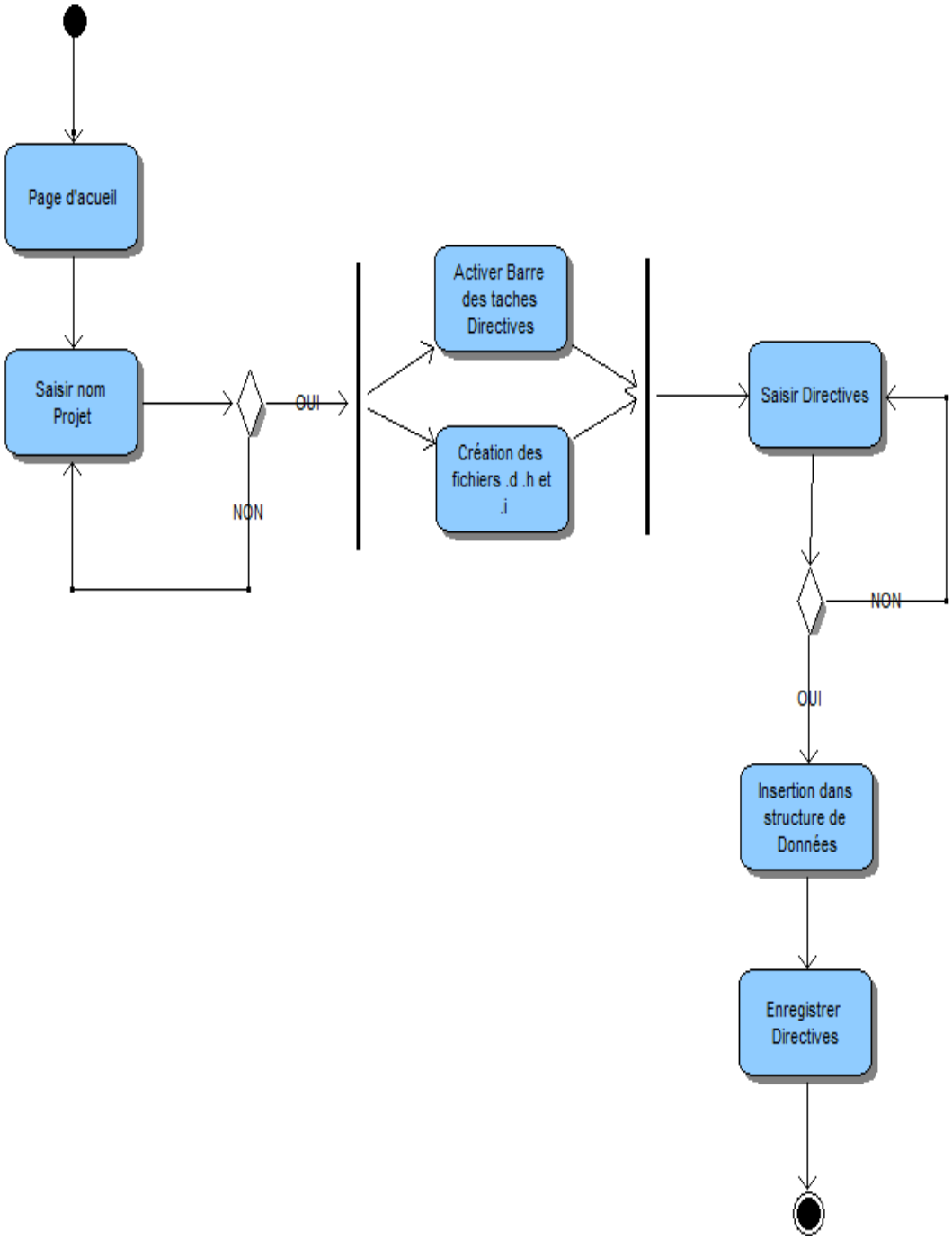

Figure 25: Diagramme d'activité «générer .d»

3.1.5 Diagramme d'activité «Générer source»

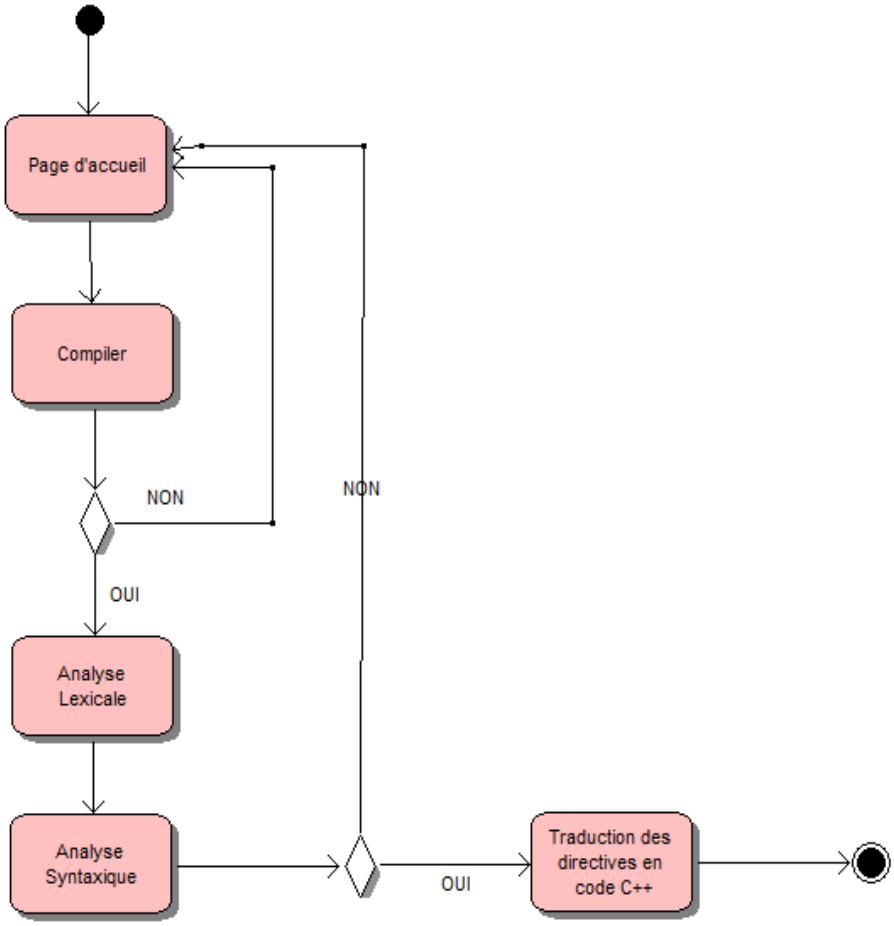

Figure 26: Diagramme d'activité «générer source»

3.1.6 Diagramme général

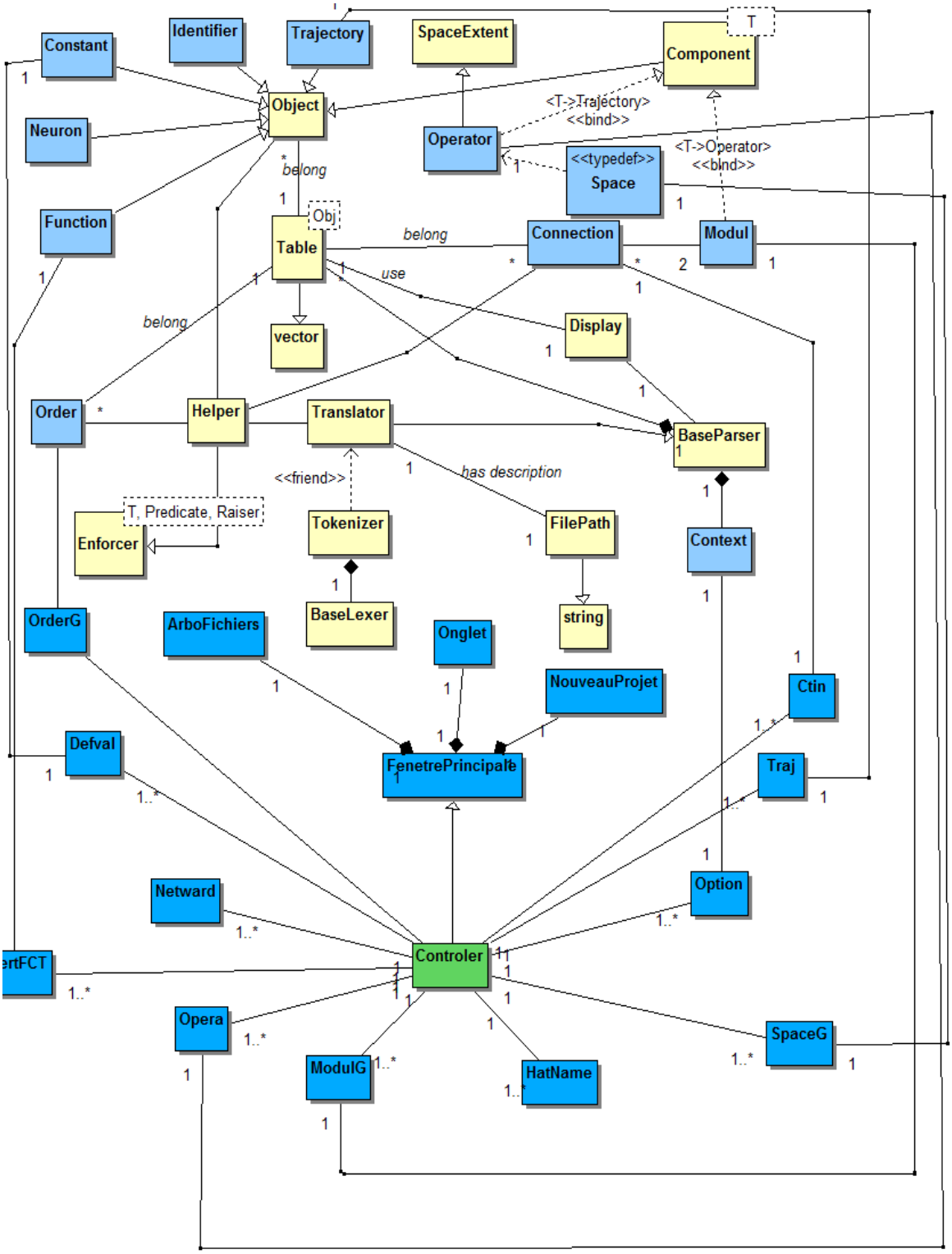

Figure 27: Diagramme général.

3.2 Proposition d'une solution graphique.

Après avoir étudié la méthode d'analyse et de conception que nous avons choisie, dans cette partie nous allons implémenter le modèle de Yao proposé c'est-à-dire passer à la conception proprement dite des fenêtres graphiques de Yao. Nous allons présenter les principales classes de notre application à savoir: FenetrePrincipale, NouveauProjet, AfficherParametres, ArboFichiers et Defval.

3.2.1 Etude de la classe FenetrePrincipale.

Comme son nom l'indique, c'est la classe principale de notre application. Dès que le programme est lancé, cette fenêtre appaîtra. Elle contient tous les boutons et liens qui mènent directement ou indirectement vers les autres objets de notre application. Ceci dit, sans cette fenêtre, rien ne fonctionnera.

Voici son apparence:

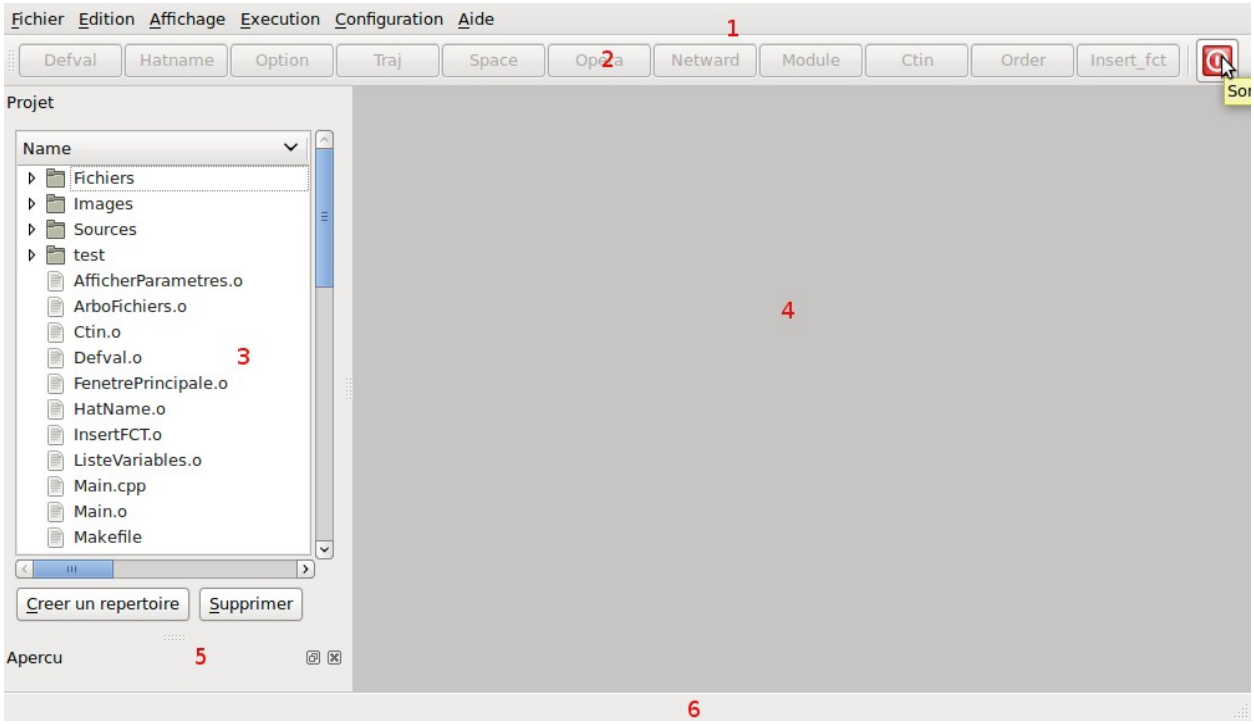

Figure 28: La fenêtre principale.

Nous voyons bien quelques boutons et menus.

Explication des numéros:

1: C'est notre barre de menus. Elle contient six (6) menus dont: Fichier, Edition, Affichage, Exécution, Configuration et Aide. Certains sont juste présents pour être fonctionnels plus tard. Les principaux sont le menu fichier et édition.

2: C'est la barre d'outils. Elle présente l'avantage d'offrir un raccourci vers un objet ou une application. Ici, nous y avons placé les variables et constantes de notre application.

3: Vue complète sur notre répertoire de travail.

4: Espace de travail (workspace en anglais). C'est un espace réservé sensé contenir les autres fenêtres qui apparaîtront lors d'un clic sur un bouton par exemple.

5: Affiche un aperçu sur les objets selectionnés.

6: C'est la barre de messages de notre application. Nous pouvons guider l'utilisateur en affichant progressivement des messages à cet endroit. Lorsque la souris pointe sur un bouton par exemple, nous pouvons afficher dans la barre de messages la fonctionnalité de bouton. Message qui peut disparaître automatiquement au bout d'un certain temps. Il existe des méthodes spécifiques dans Qt qui permettent d'inter-agir avec cette barre.

3.2.1.1 Les méthodes de la classe «FenetrePrincipale»

Nous allons donner le rôle de chacune des méthodes de la classe FenetrePrincipale. Ces dernières sont:

- ✓ creerMenus(): Comme son nom l'indique, cette méthode permet de créer les menus lorsqu'on l'appelle (voir figure 26, numéro 1). Nous avons choisi de la rendre publique afin que toutes les autres classes puissent l'appeler.
- ✓ creerActions(): avec Qt, un élément de menu est représenté par une action. C'est l'exemple des sous-menus qui représentent des actions. Cette méthode publique permet de les créer et de les gérer (que faire après un clic de souris sur une action?).
- ✓ creerBarreOutils(): permet de créer une barre d'outils (voir figure 26, numéro 2). Par défaut,

nous avons fait en sorte que cette barre cache les objets qu'elle contient. Cette fonction est publique.

- ✓ actionsBarreOutils(QWorkspace *espace): tout comme les éléments d'un menu, ceux de la barre d'outils sont considérés comme des actions. Ils sont gérés de même façon. Notons qu'ici la méthode reçoit en parametre l'espace de travail où elle doit s'afficher. Cela signifie que chaque objet de la barre d'outils s'affichera dans l'espace de travail passé en parametrage. Cette fonction est publique.
- ✓ ActiveBoutonsBarreOutils(): comme nous l'avons signalé précédemment, après la création de la barre d'outils les objets sont masqués. Cette méthode permet de les activer après la création d'un nouveau projet.
- ✓ DesactiveBoutonsBarreOutils(): méthode appelée pour désactiver les boutons de la barre d'outils.

3.2.2 Illustration de la classe NouveauProjet.

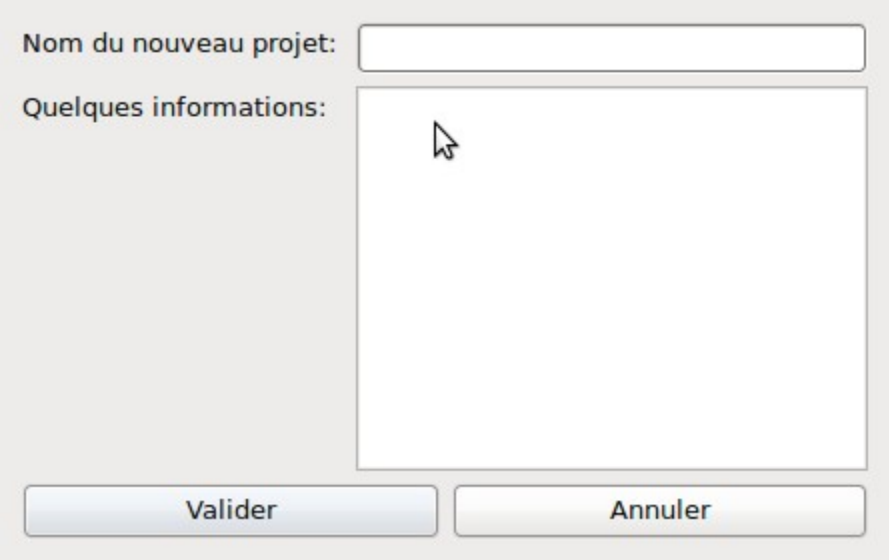

Figure 29: Illustration de la classe NouveauProjet.

3.2.3 Illustration de la classe Defval.

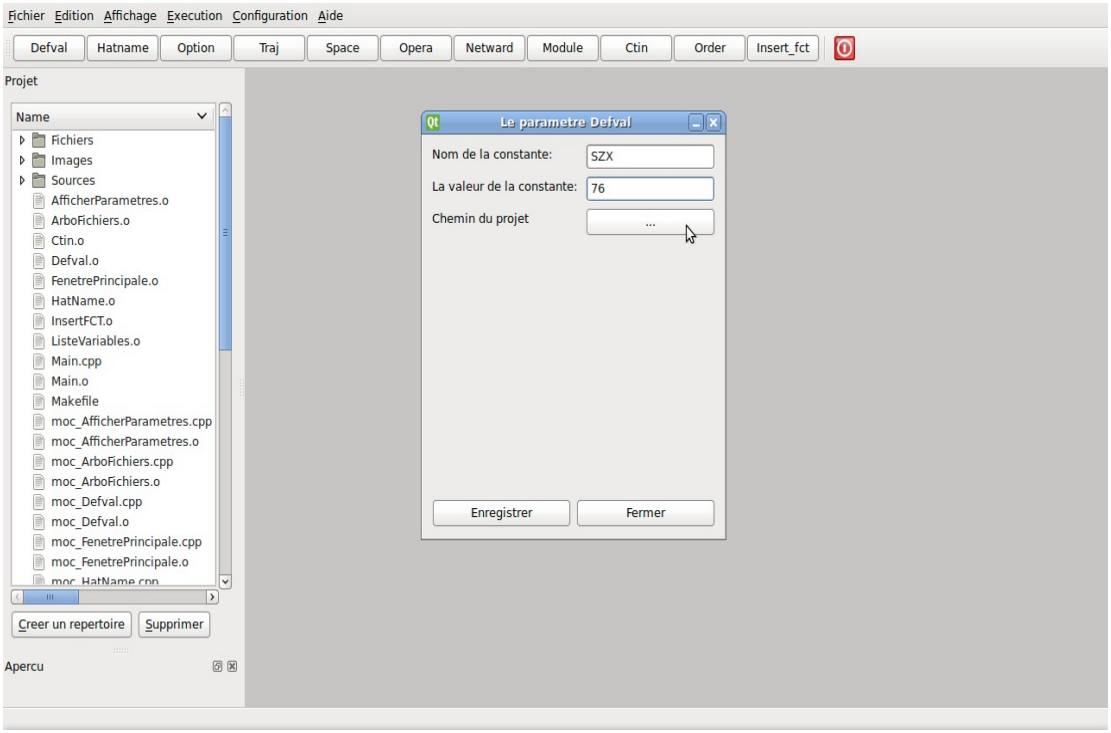

Figure 30: Illustration de la classe Defval.

Conclusion

Notre séjour au laboratoire LTI de l'Ecole Supérieure Polytechnique de Dakar nous a ouvert l'esprit sur les techniques de programmation orientée objet. Avant notre arrivée au LTI, nous ne connaissons pas le C++. Nous avons apporté notre pierre à l'édifice car le projet qui nous a été confié, à savoir la «réalisation d'une version graphique de Yao», connait aujourd'hui une fin heureuse et fructueuse. Grâce à notre application, les utilisateurs de Yao se voient faciliter l'installation, la génération et l'édition des fichiers (.d, .i et .h).

Pour créer des applications graphiques, il fallait nécessairement se servir d'un outil adéquat. Dans notre cas, cet outil est la bibliothèque graphique Qt. Il y'en a certes beaucoup d'autres; heureusement que les objectifs fixés nous ont guidé à faire ce choix. Notons également l'influence non négligeable de l'outil existant dans ce choix.

A ce stade, notre application fonctionne. Nous pensons que la seconde étape du projet peut commencer. Cette dernière concerne l'implémentation des fonctionnalités prévues à cet effet depuis la première étape. Cela nécessite de nouveaux objectifs qui peuvent faire l'objet d'une autre étude.

Glossaire

API (**A**pplication **P**rogramming **I**nterface): Une interface de programmation ou ensemble de fonctions, procédures et classes mises à disposition des programmes informatiques par une bibliothèque logicielle, un système d'exploitation ou un service.

Apple (anciennement Apple Computer Inc.): société multinationale américaine d'informatique.

Binaire: Le système binaire est un système de numération utilisant la base 2. On nomme couramment bit (de l'anglais binary digit, soit « chiffre binaire ») les chiffres de la numération binaire. Ceux ci ne peuvent prendre que deux valeurs, notées par convention 0 et 1.

Carbon: API de Apple pour les systèmes d'exploitation Macintosh pouvant être écrite en langage C ou C++. Elle est disponible sur Mac OS (depuis la version 8.1) et sur Mac OS X Ce qui permet le portage simplifié d'applications entre ces deux versions du système d'exploitation d'Apple.

Classes: En programmation orientée objet, une classe déclare des propriétés communes à un ensemble d'objets. La classe déclare des attributs représentant l'état des objets et des méthodes représentant leur comportement.

Client/Server: L'architecture client/serveur désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs clients du serveur. Chaque logiciel client peut envoyer des requêtes à un serveur.

Fonctions: également appelées methodes. Voir de défintion de la classe.

Framework: ensemble de bibliothèques, d'outils et de conventions permettant le développement d'applications. Il fournit suffisamment de briques logicielles et impose suffisamment de rigueur pour pouvoir produire une application aboutie et dont la maintenance est aisée. Ces composants sont organisés pour être utilisés en interaction les uns avec les autres (voir urbanisation).

GIMP (**G**NU **I**mage **M**anipulation **P**rogram): est un logiciel libre de traitement d'images. Il est souvent considéré comme une alternative libre au logiciel Adobe Photoshop.

GNOME (**G**NU **N**etwork **O**bject **M**odel **E**nvironment): est un environnement de bureau libre convivial du système d'exploitation GNU; cette interface est actuellement populaire sur les systèmes

GNU/Linux et fonctionne également sur la plupart des systèmes de type UNIX.

GNU (**G**nu's **N**ot **U**nix): est un projet de système d'exploitation composé exclusivement de logiciels libres.

GNU LGPL (**GNU L**esser **G**eneral **P**ublic **L**icence): La Licence publique générale limitée GNU

GUI (Graphical User Interface): « interface utilisateur graphique ». Elle s'oppose à CLI pour Command Line Interface, soit « interface en ligne de commande ».

Kilo-octets : L'octet est une unité de mesure en informatique mesurant la quantité de données. Un octet est lui-même composé de 8 bits. Un kilo-octet vaut 2^{10} octets (tel que défini par IEC 60027-2).

Makefile: make est un logiciel traditionnel d'UNIX. C'est un « moteur de production » : il sert à appeler des commandes créant des fichiers. À la différence d'un simple script shell, make exécute les commandes contenue dans le fichier Makefile seulement si elles sont nécessaires. Le but est d'arriver à un résultat (logiciel compilé ou installé, documentation créée, etc.) sans nécessairement refaire toutes les étapes.

MVC (**M**odèle-**V**ue-**C**ontrôleur): architecture et une méthode de conception qui organise l'interface homme-machine (IHM) d'une application logicielle.

.NET: est le nom d'un ensemble de produits et de technologies informatiques de l'entreprise Microsoft pour rendre ses applications portables ou facilement accessibles par Internet.

Objets: Un objet est l'instanciation d'une classe.

Passerelle: dispositif permettant de relier deux réseaux informatiques d'autorités différentes.

Perl: langage de programmation crée par Larry Wall en 1987 reprenant des fonctionnalités du langage C et des langages de scripts.

Python: langage de programmation interprété multi-paradigme. Il favorise la programmation impérative structurée et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions.

Ruby: langage de programmation libre. Il est interprété, orienté objet et multi-paradigme.

Script: programme en langage interprété.

Serveur X: application graphique.

Templates: sont une particularité de la programmation en language C++, qui autorise l'écriture d'un code sans considération envers le type des données avec lesquelles il sera finalement utilisé.

Tera-Octet: Un tera-octet vaut 2^{12} (tel que défini par IEC 60027-2). Voir également kilo-octet.

UNIX: Nom d'un système d'exploitation multitâche et multi-utilisateur crée en 1969, conceptuellement ouvert et fondé sur une approche par laquelle il offre de nombreux petits outils chacun dotés d'une mission spécifique.

Bibliographie et Webographie

Bibliographie:

[b1]: Programmer en Java, JOHN R. HUBBARD

[b2]: Qt4 et C++ Programmation d'interfaces GUI, Jasmin Blanchette et Mark Summerfield.

[b3]: Le langage C++, Bjarne Stroutrup, Edition spéciale revue et corrigée.

Webographie:

Référence	*Titre*	*URL*	*Consulté le*
[w1]	Documents sur Windows API.	http://fr.wikipedia.org/wiki/Windows_API	07/04/09
[w2]	Documents sur MFC	http://fr.wikipedia.org/wiki/MFC	08/04/09
[w3]	Documents sur Cocoa	http://fr.wikipedia.org/wiki/Cocoa_(Apple)	09/04/09
[w4]	Documents sur Xlib	http://fr.wikipedia.org/wiki/Xlib	10/04/09
[w5]	Documents sur GTk+	http://fr.wikipedia.org/wiki/GTK%2B	11/04/09
[w6]	Documents sur wxWidget	http://fr.wikipedia.org/wiki/WxWidgets	12/04/09
[w7]	Introduction sur C++ et Qt	http://www.siteduzero.com/tutoriel-3-11121-introduction-au-c.html	20/07/09
[w8]	Les classes de Qt.	http://doc.trolltech.com/4.5/classes.html	30/08/09

Codes sources de quelques classes.

A. La classe FenetrePrincipale.

Le fichier FenetrePrincipale.h:

```
#ifndef FENETREPRINCIPALE_H
#define FENETREPRINCIPALE_H

#include <QtGui>
#include <QMainWindow>
#include "Defval.h"
#include "HatName.h"
#include "Option.h"
#include "Traj.h"
#include "Space.h"
#include "Opera.h"
#include "Netward.h"
#include "Modul.h"
#include "Ctin.h"
#include "Order.h"
#include "InsertFCT.h"
#include "ArboFichiers.h"
#include "NouveauProjet.h"
#include <QWorkspace>

class FenetrePrincipale : public QMainWindow {
        Q_OBJECT
                public :
                        FenetrePrincipale(QWidget *parent = 0);
                        void creerActions();
                        void creerMenus();
                        void creerBarreOutils();
                        void actionsBarreOutils(QWorkspace *espace);
                        ~FenetrePrincipale();
                public slots:
                        void activeBoutonsBarreOutils();
                        void desactiveBoutonsBarreOutils();
                        bool nouveauProjet();
                        void ouvrirProjet();
                        bool sauvegarderProjet(const QString &fileName);
                        void cacherBarreOutils();
                        void montrerActionGenererD();
                        void montrerActionCompilation();
                private :
```

```
// Les attributs des menus
        QMenu *menuFichier;
        QMenu *menuEdition;
        QMenu *menuAffichage;
        QMenu *menuExecution;
        QMenu *menuConfig;
        QMenu *menuAide;
// Les actions du menu Fichier
        QMenu *nouveau;
        QAction *actionFichierDescription;
        QAction *actionFichierHat;
        QAction *actionFichierInstruction;
        QAction *actionGrapheModulaire;
        QAction *actionInfosProjet;
        QAction *actionNouveauProjet;
        QAction *actionOuvrirProjet;
        QAction *actionOuvrirProjetsRecents;
        QAction *actionImporterProjet;
        QAction *actionEnregistrerProjet;
        QAction *actionEnregistrerSousProjet;
        QAction *actionImprimerProjet;
        QAction *actionSortieProjet;
// Les actions du menu edition
        QAction *actionAnnuler;
        QAction *actionRevenir;
        QAction *actionCouper;
        QAction *actionCopier;
        QAction *actionColler;
        QAction *actionSelectionnerTout;
// Les actions du menu affichage
        QAction *actionCacherBarreOutils;
        QAction *actionCacherFenetreApercu;
        QAction *actionCacherFenetreProjet;
        QAction *actionCacherFenetreDialog;
        QAction *actionPleinEcran;
// Les actions du menu Execution
        QAction *actionGenerationD;
        QAction *actionCompilation;
        QAction *actionExecution;
// Les actions du menu configuration
        QAction *actionEditeur;
// Les actions du menu aide
        QAction *actionDocYao;
        QAction *actionRapport;
```

```
        // La barre d'outils
            QPushButton *m_defval;
            QPushButton *m_hatName;
            QPushButton *m_option;
            QPushButton *m_traj;
            QPushButton *m_space;
            QPushButton *m_opera;
            QPushButton *m_netward;
            QPushButton *m_modul;
            QPushButton *m_ctin;
            QPushButton *m_order;
            QPushButton *m_insertFCT;
            QToolBar *yaoToolBar; //Crée une barre d'outils
            QStatusBar *barreEtat;
            QWorkspace *espace;
            QTableWidget *tableWidget;
            QString curFile;
            NouveauProjet *np;
};
#endif // FENETREPRINCIPALE_H
```

Le fichier FenetrePrincipale.cpp:

```
#include "Include/FenetrePrincipale.h"

FenetrePrincipale::FenetrePrincipale(QWidget *parent) : QMainWindow(parent) {

	this->setWindowTitle("Visual Yao"); //Donne un titre à la fenêtre principale

	espace = new QWorkspace; //Crée un espace de travail. Ici, ça serait notre zone centrale
	QDockWidget *fenetreProjet = new QDockWidget("Projet", this);
	QDockWidget *dock2 = new QDockWidget("Apercu", this);

	addDockWidget(Qt::LeftDockWidgetArea, fenetreProjet);
	addDockWidget(Qt::LeftDockWidgetArea, dock2);

	ArboFichiers *directoryViewer = new ArboFichiers(); // Affiche la fenêtre projet

	fenetreProjet->setWidget(directoryViewer);
	fenetreProjet->setFeatures(QDockWidget::NoDockWidgetFeatures);

	setCentralWidget(espace); //Ici, se précise la zone centrale de la QMainWindow.

	creerActions(); //Appel de la methode creerActions(), voir le code ci-après.
	creerMenus(); // Appel de la méthode creerMenu(), voir le code ci-après.
	creerBarreOutils(); //Cree la barre d'outils

	barreEtat = statusBar(); //Affiche la barre d'état.

	this->showMaximized(); // Maximise la fenetre principale
}

// Implémentation de la methode creerMenus()
void FenetrePrincipale::creerMenus() {

	menuFichier = menuBar()->addMenu(tr("&Fichier"));
	menuFichier->addAction(actionNouveauProjet);
	menuFichier->addAction(actionOuvrirProjet);
	menuFichier->addAction(actionOuvrirProjetsRecents);
	menuFichier->addSeparator();
	menuFichier->addAction(actionImporterProjet);
	menuFichier->addSeparator();
	menuFichier->addAction(actionEnregistrerProjet);
	menuFichier->addAction(actionEnregistrerSousProjet);
	menuFichier->addSeparator();
```

```
        menuFichier->addAction(actionSortieProjet);

        menuEdition = menuBar()->addMenu("&Edition");
        menuEdition->addAction(actionFichierDescription);
        menuEdition->addAction(actionFichierHat);
        menuEdition->addAction(actionFichierInstruction);
        menuEdition->addAction(actionGrapheModulaire);
        menuEdition->addAction(actionInfosProjet);

        menuAffichage = menuBar()->addMenu("&Affichage");
        menuAffichage->addAction(actionCacherBarreOutils);
        menuAffichage->addAction(actionCacherFenetreApercu);
        menuAffichage->addAction(actionCacherFenetreProjet);
        menuAffichage->addAction(actionCacherFenetreDialog);

        menuExecution = menuBar()->addMenu("&Execution");
        menuExecution->addAction(actionGenerationD);
        menuExecution->addAction(actionCompilation);
        menuExecution->addAction(actionExecution);

        menuConfig = menuBar()->addMenu("&Configuration");
        menuConfig->addAction(actionEditeur);

        menuAide = menuBar()->addMenu(tr("&Aide"));
        menuAide->addAction(actionDocYao);
        menuAide->addAction(actionRapport);
}

// Implémentation de la methode creerActions()
void FenetrePrincipale::creerActions() {

        actionNouveauProjet = new QAction(tr("Nouveau projet"), this);
                actionNouveauProjet->setIcon(QIcon("Images/icons/newfile.png"));
                actionNouveauProjet->setStatusTip("Creer un nouveau projet");

connect(actionNouveauProjet, SIGNAL(triggered()), this, SLOT(activeBoutonsBarreOutils()));
connect(actionNouveauProjet, SIGNAL(triggered()), this, SLOT(nouveauProjet()));
connect(actionNouveauProjet, SIGNAL(triggered()), this, SLOT(montrerActionGenererD()));

        actionOuvrirProjet = new QAction(tr("&Ouvrir un projet"), this);
        actionOuvrirProjet->setIcon(QIcon("Images/icons/openfile.png"));
        actionOuvrirProjet->setShortcut(tr("Ctrl+O"));
        actionOuvrirProjet->setStatusTip("Ouvrir un projet existant");
```

```
connect(actionOuvrirProjet, SIGNAL(triggered()), this, SLOT(ouvrirProjet()));
connect(actionOuvrirProjet, SIGNAL(triggered()), this, SLOT(montrerActionCompilation()));

        actionOuvrirProjetsRecents = new QAction(tr("Ouvrir projets recents"), this);
        actionOuvrirProjetsRecents->setShortcut(tr("Ctrl+P"));
        actionOuvrirProjetsRecents->setStatusTip("Projets recemment ouverts");

        actionImporterProjet = new QAction(tr("&Importer"), this);
        actionImporterProjet->setShortcut(tr("Ctrl+E"));
        actionImporterProjet->setStatusTip("Importer un projet");

        actionEnregistrerProjet = new QAction(tr("Enregi&strer"), this);
        actionEnregistrerProjet->setIcon(QIcon("Images/icons/save.png"));
        actionEnregistrerProjet->setEnabled(false); // Option désactivée par défaut.
        actionEnregistrerProjet->setStatusTip("Pour sauvegarder le projet en cours");

        actionEnregistrerSousProjet = new QAction(tr("Enregistrer sous"), this);
        actionEnregistrerSousProjet->setIcon(QIcon("Images/icons/saveas.png"));
        actionEnregistrerSousProjet->setEnabled(false); // Option désactivée par défaut.
        actionEnregistrerSousProjet->setStatusTip("En le projet sous un autre nom");

        actionImprimerProjet = new QAction(tr("Imprimer"), this);
        actionImprimerProjet->setStatusTip("Options d'impression");

        actionSortieProjet = new QAction(tr("&Sortir"), this);
        actionSortieProjet->setIcon(QIcon("Images/icons/exit.png"));
        actionSortieProjet->setShortcut(tr("Ctrl+Q"));
        actionSortieProjet->setStatusTip("Fermer le programme");

connect(actionSortieProjet, SIGNAL(triggered()), this, SLOT(close()));

// Les actions du menu Edition
        actionFichierDescription = new QAction(tr("Fichier de description (.d)"), this);
        actionFichierHat = new QAction(tr("Fichier hat (.h)"), this);
        actionFichierInstruction = new QAction(tr("Fichier instruction (.i)"), this);
        actionGrapheModulaire = new QAction(tr("Graphe modulaire"), this);
        actionInfosProjet = new QAction(tr("Edition infos projet"), this);

// Les actions du menu Affichage
        actionCacherBarreOutils = new QAction(tr("Cacher la barre d'outils"), this);
        actionCacherBarreOutils->setCheckable(true);
        connect(actionCacherBarreOutils, SIGNAL(triggered()), this, SLOT(cacherBarreOutils()));
        actionCacherFenetreApercu = new QAction(tr("Cacher la fenetre apercu"), this);
        actionCacherFenetreApercu->setCheckable(true);
```

```
        actionCacherFenetreProjet = new QAction(tr("Cacher la fenetre Projet"), this);
        actionCacherFenetreProjet->setCheckable(true);
        actionCacherFenetreDialog = new QAction(tr("Cacher la fenetre dialog"), this);
        actionCacherFenetreDialog->setCheckable(true);
        actionPleinEcran = new QAction(tr("Plein ecran"), this);
        actionPleinEcran->setCheckable(true);

// Les actions du menu Execution
        actionGenerationD = new QAction(tr("Generer.d"), this);
        actionGenerationD->setEnabled(false);
        actionCompilation = new QAction(tr("Compilation"), this);
        actionCompilation->setEnabled(false);
        actionExecution = new QAction(tr("Execution"), this);
        actionExecution->setEnabled(false);
// Les actions du menu Configuration
        actionEditeur = new QAction(tr("Editeur par defaut"), this);

// Les actions du menu Aide
        actionDocYao = new QAction(tr("Doc Yao"), this);
        actionRapport = new QAction(tr("Rapport"), this);

// Les actions de la barre d'outils

        actionsBarreOutils(espace);

}

// Implémentation de la methode creerBarreOutils()
void FenetrePrincipale::creerBarreOutils() {

        yaoToolBar = addToolBar(tr("&YaoToolbar"));
        yaoToolBar->addWidget(m_defval);
        yaoToolBar->addWidget(m_hatName);
        yaoToolBar->addWidget(m_option);
        yaoToolBar->addWidget(m_traj);
        yaoToolBar->addWidget(m_space);
        yaoToolBar->addWidget(m_opera);
        yaoToolBar->addWidget(m_netward);
        yaoToolBar->addWidget(m_modul);
        yaoToolBar->addWidget(m_ctin);
        yaoToolBar->addWidget(m_order);
        yaoToolBar->addWidget(m_insertFCT);
        yaoToolBar->addSeparator();
        yaoToolBar->addAction(actionSortieProjet);
```

```
 }

// Implémentation de la methode cacherBarreOutils
void FenetrePrincipale::cacherBarreOutils() {
        yaoToolBar->hide();
        barreEtat->showMessage(tr("La barre des outils est cachee!"), 2000);
}

// Implementation de la methode actionsBarreOutils()
void FenetrePrincipale::actionsBarreOutils(QWorkspace *espace) {

        m_defval = new QPushButton(tr("Defval"), this);
        Defval *defval = new Defval;
        espace->addWindow(defval);
        defval->hide();
        connect(m_defval, SIGNAL(clicked()), defval, SLOT(show()));
        m_hatName = new QPushButton(tr("Hatname"), this);
        HatName *hat = new HatName;
        espace->addWindow(hat);
        hat->hide();
        connect(m_hatName, SIGNAL(clicked()), hat, SLOT(show()));
        m_option = new QPushButton(tr("Option"), this);
        Option *op = new Option;
        espace->addWindow(op);
        op->hide();
        connect(m_option, SIGNAL(clicked()), op, SLOT(show()));
        m_traj = new QPushButton(tr("Traj"), this);

        Traj *trajectoire = new Traj;
        espace->addWindow(trajectoire);
        trajectoire->hide();
        connect(m_traj, SIGNAL(clicked()), trajectoire, SLOT(show()));

        m_space = new QPushButton(tr("Space"), this);
        Space *esp = new Space;
        espace->addWindow(esp);
        esp->hide();
        connect(m_space, SIGNAL(clicked()), esp, SLOT(show()));

        m_opera = new QPushButton(tr("Opera"), this);
        Opera *opera = new Opera;
        espace->addWindow(opera);
        opera->hide();
        connect(m_opera, SIGNAL(clicked()), opera, SLOT(show()));
```

```
        m_netward = new QPushButton(tr("Netward"), this);
        Netward *netward = new Netward;
        espace->addWindow(netward);
        netward->hide();
        connect(m_netward, SIGNAL(clicked()), netward, SLOT(show()));

        m_modul = new QPushButton(tr("Module"), this);
        Modul *module = new Modul;
        espace->addWindow(module);
        module->hide();
        connect(m_modul, SIGNAL(clicked()), module, SLOT(show()));

        m_ctin = new QPushButton(tr("Ctin"), this);
        Ctin *ctin = new Ctin;
        espace->addWindow(ctin);
        ctin->hide();

        connect(m_ctin, SIGNAL(clicked()), ctin, SLOT(show()));
        m_order = new QPushButton(tr("Order"), this);

        Order *order = new Order;
        espace->addWindow(order);
        order->hide();
        connect(m_order, SIGNAL(clicked()), order, SLOT(show()));

        m_insertFCT = new QPushButton(tr("Insert_fct"), this);
        InsertFCT *insert = new InsertFCT;
        espace->addWindow(insert);
        insert->hide();
        connect(m_insertFCT, SIGNAL(clicked()), insert, SLOT(show()));
        desactiveBoutonsBarreOutils(); // Désactive tous les boutons de la barre d'outils
}

// Implémentation de la methode activeBoutonsBarreOutils()
void FenetrePrincipale::activeBoutonsBarreOutils() {

        m_defval->setEnabled(true);
        m_hatName->setEnabled(true);
        m_option->setEnabled(true);
        m_traj->setEnabled(true);
        m_space->setEnabled(true);
        m_opera->setEnabled(true);
        m_netward->setEnabled(true);
```

```
        m_modul->setEnabled(true);
        m_ctin->setEnabled(true);
        m_order->setEnabled(true);
        m_insertFCT->setEnabled(true);
}

// Implémentation de la methode desactiveBoutonsBarreOutils()
void FenetrePrincipale::desactiveBoutonsBarreOutils() {

        m_defval->setEnabled(false);
        m_hatName->setEnabled(false);
        m_option->setEnabled(false);
        m_traj->setEnabled(false);
        m_space->setEnabled(false);
        m_opera->setEnabled(false);
        m_netward->setEnabled(false);
        m_modul->setEnabled(false);
        m_ctin->setEnabled(false);
        m_order->setEnabled(false);
        m_insertFCT->setEnabled(false);
}

// Implémentation de la methode nouveauProjet (SLOT)
bool FenetrePrincipale::nouveauProjet() {

        np = new NouveauProjet;
        np->show();
    return true;
}
// Implémentation de la methode sauvegardeProjet()
bool FenetrePrincipale::sauvegarderProjet(const QString &fileName) {

        statusBar()->showMessage(tr("Projet sauvegarde"), 2000);
        return true;
}

// Implémentation de la methode ouvrirProjet() (SLOT)
void FenetrePrincipale::ouvrirProjet() {
        QFileDialog *fd = new QFileDialog(this);
        QStringList filtres;
        fd->setDirectory(".");
        fd->setFileMode (QFileDialog::ExistingFiles);
        filtres << "*.*"<< "Tous les fichiers (*.*)";
```

```
        fd->setFilters(filtres);

        QStringList lstF = fd->selectedFiles();
    }
  }

// Implémentation de la methode montrerActionGenererD()
void FenetrePrincipale::montrerActionGenererD() {

        actionGenerationD->setEnabled(true);
 }

// Implémentation de la methode montrerActionCompilation()
void FenetrePrincipale::montrerActionCompilation() {

        actionCompilation->setEnabled(true);
}

// Implémentation de la methode ecrireFichier
void FenetrePrincipale::ecrireFichier(const QString &nomFichier, const QString &cheminFichier)
{

        QFile fichier(cheminFichier + "/" + nomFichier + ".d");
                if (!fichier.open(QIODevice::WriteOnly)) {
                        std::cerr << "Ouverture impossible en écriture: "
                                << qPrintable(fichier.errorString()) << std::endl;
                                }
        QTextStream out(&fichier);
        out << "texte" << endl;

fichier.close();
}


// Implémentation de la methode creerRepertoire
void FenetrePrincipale::creerRepertoire() {

QString dirName = QInputDialog::getText(this, tr("Creer un repertoire"), tr("Nom du repertoire"));

        if (!dirName.isEmpty()) {
                QMessageBox::information(this, tr("Creer un repertoire"), tr("Impossible de creer le
repertoire"));
                }
}
```

B. La classe NouveauProjet.

Le fichier NouveauProjet.h:

```
#ifndef NOUVEAUPROJET_H
#define NOUVEAUPROJET_H

#include "Sources/Include/Entete.h"
#include "Sources/Include/ArboFichiers.h"
#include <QLineEdit>
#include <QTextEdit>
#include <QInputDialog>
#include <QFormLayout>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QString>
#include <QMessageBox>
#include <QDir>
#include <QChar>

class NouveauProjet : public QWidget {

   Q_OBJECT

   public :
        NouveauProjet(QWidget *parent=0);
        void setNomProjet(QString chaine);
        QString getNomProjet();
        void setCheminProjet(QString chaine);
        QString getCheminProjet();
        void creerFichier(QString nomProjet, QString extension);
        void setFichier(QString parametre);
        QString getFichier();
   public slots:
        void creerRepertoireProjet();
   private :
        QString m_nomProjet;
        QString m_cheminProjet;
        QChar *accesNomProjet;
        QLineEdit *ligneNomRep;
        QTextEdit *ligneInfos;
        QPushButton *m_creerRepertoireProjet;
        QPushButton *m_quitProjet;
        QHBoxLayout *m_layoutH;
        QFormLayout *m_layoutF;
        QString m_infosProjet;
```

```
 };
#endif // NOUVEAUPROJET_H
```

Le fichier NouveauProjet.cpp:

```
#include "Include/NouveauProjet.h"

NouveauProjet::NouveauProjet(QWidget *parent) : QWidget(parent) {

	this->setWindowTitle("Nouveau projet");

	ligneNomRep		= new QLineEdit;
	ligneInfos		= new QTextEdit;
	m_layoutF		= new QFormLayout;
	m_layoutH		= new QHBoxLayout;
	m_creerRepertoireProjet	= new QPushButton("Valider", this);
	m_quitProjet		= new QPushButton("Annuler", this);

	m_layoutF->addRow("Nom du nouveau projet: ", ligneNomRep);
	m_layoutF->addRow("Quelques informations: ", ligneInfos);

	m_layoutH->addWidget(m_creerRepertoireProjet);
	m_layoutH->addWidget(m_quitProjet);

	m_layoutF->addRow(m_layoutH);

	this->setLayout(m_layoutF);

	connect(m_quitProjet, SIGNAL(clicked()), this, SLOT(close()));
	connect(m_creerRepertoireProjet, SIGNAL(clicked()), this,SLOT(creerRepertoireProjet()));
}

// Implementation de la méthode creerRepertoireProjet
void NouveauProjet::creerRepertoireProjet() {

	setCheminProjet(QDir::currentPath());
	setNomProjet(ligneNomRep->text());
	 QDir rep(getCheminProjet());
	if (!rep.exists(getNomProjet()))
	{
		rep.mkdir(getNomProjet());

		creerFichier(getNomProjet(), ".d");
		creerFichier(getNomProjet(), ".h");
```

```
                creerFichier(getNomProjet(), ".txt");

                QMessageBox::about(this, tr("Operation reussie"),
                     tr("<p>Un repertoire portant le nom du projet a ete cree avec succes.<p> Cliquer
                        sur <b>ok</b> pour fermer cette fenetre."));
        }
       else
                QMessageBox::about(this, tr("Nouveau projet"), tr("<p>Le projet existe deja! <p>
Veuillez changer le nom."));
}

// Implementation de la méthode setNomProjet
void NouveauProjet::setNomProjet(QString chaine) {

        m_nomProjet = chaine;
}

// Implementation de la methode getNomProjet
QString NouveauProjet::getNomProjet() {

        return m_nomProjet;
}

// Implementation de la methode
void NouveauProjet::setCheminProjet(QString chaine) {

        m_cheminProjet = chaine ;
}

// Implementation de la methode
QString NouveauProjet::getCheminProjet() {

        return m_cheminProjet;
}
//
void NouveauProjet::setFichier(QString parametre) {

        QFile file("test/test.d");
        if (!file.open(QIODevice::WriteOnly | QIODevice::Append)) {
           std::cerr << "Ouverture impossible en écriture: "
                  << qPrintable(file.errorString()) << std::endl;
           }
           QTextStream out(&file);
           if (file.atEnd())
```

```
            out << parametre << endl;
            file.close();
}

void NouveauProjet::creerFichier(QString nomProjet, QString extension) {
   QFile file(nomProjet + "/" + nomProjet + extension);
        if (!file.open(QIODevice::WriteOnly)) {
            std::cerr << "Ouverture impossible en écriture: "
                    << qPrintable(file.errorString()) << std::endl;
            }
     QTextStream out(&file);
            out << "" << endl;
            file.close(); }
```

C. La classe ArboFichiers.

Le fichier ArboFichiers.h:

```
#ifndef ARBOFICHIERS_H
#define ARBOFICHIERS_H

#include <QDialog>

class QDirModel;
class QPushButton;
class QTreeView;

class ArboFichiers : public QDialog {

   Q_OBJECT
        public:
                ArboFichiers(QWidget *parent = 0);
        public slots:
                void createDirectory();
                void remove();
        private:
                QTreeView *treeView;
                QDirModel *model;
                QPushButton *mkdirButton;
                QPushButton *removeButton;
};
#endif //ARBOFICHIERS_H
```

Le fichier ArboFichiers.cpp:

```
#include <QtGui>

#include "ArboFichiers.h"

ArboFichiers::ArboFichiers(QWidget *parent) : QDialog(parent)

{
        model = new QDirModel;
        model->setReadOnly(false);
        model->setSorting(QDir::DirsFirst | QDir::IgnoreCase | QDir::Name);

        treeView = new QTreeView;
        treeView->setModel(model);
        treeView->setRootIndex(model->index("."));
        treeView->header()->setStretchLastSection(true);
        treeView->header()->setSortIndicator(0, Qt::AscendingOrder);
        treeView->header()->setSortIndicatorShown(true);
        treeView->header()->setClickable(true);

        QModelIndex index = model->index(QDir::currentPath());
        treeView->expand(index);
        treeView->scrollTo(index);
        treeView->resizeColumnToContents(0);

        mkdirButton = new QPushButton(tr("&Creer un repertoire"));
        removeButton = new QPushButton(tr("&Supprimer"));

        connect(mkdirButton, SIGNAL(clicked()), this, SLOT(createDirectory()));
        connect(removeButton, SIGNAL(clicked()), this, SLOT(remove()));

        QHBoxLayout *buttonLayout = new QHBoxLayout;
                buttonLayout->addWidget(mkdirButton);
                buttonLayout->addWidget(removeButton);
                buttonLayout->addStretch();

        QVBoxLayout *mainLayout = new QVBoxLayout;
                 mainLayout->addWidget(treeView);
                 mainLayout->addLayout(buttonLayout);
        setLayout(mainLayout);

   setWindowTitle(tr("Arborescence des fichiers"));
}
```

```
void ArboFichiers::createDirectory() {

   QModelIndex index = treeView->currentIndex();
   if (!index.isValid())
      return;

   QString dirName = QInputDialog::getText(this, tr("Creer un repertoire"), tr("Nom du
repertoire"));

   if (!dirName.isEmpty()) {
      if (!model->mkdir(index, dirName).isValid())
         QMessageBox::information(this, tr("Creer un repertoire"),
               tr("Impossible de creer le repertoire"));
   }
}

void ArboFichiers::remove() {
   QModelIndex index = treeView->currentIndex();
   if (!index.isValid())
      return;
   bool ok;
   if (model->fileInfo(index).isDir()) {
      ok = model->rmdir(index);
   } else {
      ok = model->remove(index);
   }

   if (!ok)

      QMessageBox::information(this, tr("Supprimer"),
            tr("Impossible de supprimer %1").arg(model->fileName(index)));
}
```