الجمهورية الجزائرية الديموقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي و البحث العلمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

THESE

Présentée
A

جامعة باتنة
UNIVERSITE DE BATNA

FACULTE DES SCIENCES DE L'INGENIEUR
DEPARTEMENT D'ELECTROTECHNIQUE

Pour l'obtention du diplôme de
DOCTORAT

En
ELECTROTECHNIQUE

Option
RESEAUX ELECTRIQUES

Par
LOUIZA BENFARHI
Ingénieur d'état en Electrotechnique

--THEME--

DEVELOPPEMENT D'UN OUTIL GRAPHIQUE POUR LA SIMULATION DES RESEAUX ELECTRIQUES PAR LA PROGRAMMATION ORIENTEE OBJETS

Soutenue le 29/06/2006 devant le jury :

Dr. R. ABDESSEMED	Professeur	U. Batna	Président
Dr. M. BELKACEMI	Professeur	U. Batna	Rapporteur
Dr. K. ZEHAR	Professeur	U. Sétif	Examinateur
Dr. A. CHAKER	Professeur	U. Oran	Examinateur
Dr. A. CHAGHI	Maître de Conférences	U. Batna	Examinateur
Dr. M. BOUCHERMA	Docteur	U. Constantine	Invité

2006

ملخص

هذا البحث عرض لفلسفة جديدة تعتمد على التقنية الموجهة نحو الأهداف لإنجاز برامج الشبكات الكهربائية. أهم خصائص هذه التقنية التي استعملت هي الفصل بين المعطيات عن طريق التجرد، الصيانة، هيئة المعلومات و إعادة استعمال البرامج. تم إنجاز أرضية معلوماتية لمحاك الشبكات الكهربائية يمكن استعماله كنواة لأبحاث في المستقبل. يتكون هذا المحاكي من برنامج رسومات لعناصر الشبكات الكهربائية، قاعدة بيانات و طرق أو دوال الحساب. نظرا للفوائد الكثيرة لتقنية الأهداف الموجهة في ميدان الشبكات الكهربائية اعتمدت في أغلب البحوث المنشورة مؤخرا. في هذا البحث تم تمثيل العناصر الفيزيائية المكونة للشبكات الكهربائية و طرق حسابها أيضا بواسطة هذه التقنية. اعتمادا على الهيئة الحقيقية للشبكات الكهربائية، تم إنجاز ثلاثة أقسام خاصة بالعناصر الفيزيائية وهي قسم الموصل العمومي، قسم العناصر المتصلة بموصلين عموميين اثنين وقسم العناصر المتصلة بوصل عمومي واحد.بالنسبة لطرق الحساب، تم إنجاز قسم لحساب توزيع الأحمال و قسم لحساب دارات القصر. كلا القسمين ينشق عن قسم الشبكات الكهربائية الذي يضم كل العناصر الفيزيائية المكونة له.

Résumé

Dans ce travail, une nouvelle philosophie de développement de logiciels des réseaux électriques basée sur la Technique Orientée Objets est présentée. Des aspects généraux telles que, la séparation des concepts à travers différentes abstraction du système, implémentation du code, la maintenance, la structure de données et la réutilisation du code sont utilisés. Une structure informatique graphique de base est développée, qui peut servir de noyau pour les travaux futurs. Cette structure de classes est conçue non seulement pour représenter les éléments physiques du système, mais également les concepts abstraits telles que les fonctions d'analyse des réseaux électriques (applications). Ces applications sont organisées en structure de classes liées aux structures de classes des éléments physiques du réseau électrique. Deux applications sont conçues et implémentées en langage de programmation C++. Ces deux applications, l'écoulement de puissances et le calcul des courants de court circuit, la GUI avec l'éditeur graphique et la base de données visuelle développés forment un prototype de simulation des réseaux électriques.

Abstract

In this work a new philosophy for the software development for Power Systems based on the Object Oriented Technique paradigm is presented. General aspects, such as a separation of concepts through different abstractions of the system, code implementation, the concepts of maintainability, data structures, and code reutilization are used. An oriented object graphical and computational base for the representation of power systems is developed, serving as a kernel for future developments. This class structure is suited not only to represent physical elements that form the system, but also abstract concepts, such as analysis and synthesis methodologies applied to power system (applications). These applications are organized in class structures connected to the representative structures of the physical elements of the power system. Two applications are designed and implemented in C++ programming language. These applications, a power flow and a short circuit analysis, The GUI with the graphical editor and the visual data base developed form a prototype of a simulation tool for power system analysis.

Remerciements

Je tiens à remercier très vivement, Dr. M. Belkacemi, professeur à l'université de Batna et directeur de thèse, pour la confiance qu'il m'a accordée à réaliser ce projet ainsi que pour sa grande attention et sa patience tout au long de ce travail. Je lui suis très reconnaissante de m'avoir aidé à surmonter les conditions difficiles que j'ai affrontées afin de finaliser cette thèse.

Je remercie vivement, Dr. R. Abdessemed, professeur à l'université de Batna de m'avoir honoré en acceptant d'être président de jury de cette thèse ainsi que pour ses encouragements.

J'adresse également mes remerciements aux personnes ayant accepté de participer au jury de cette thèse malgré leurs obligations :
Dr. K. Zehar professeur à l'université de Sétif,
Dr. A. Chaker professeur à l'université d'Oran,
Dr. A. Chaghi maître de conférences à l'université de Batna,
Dr. M. Boucherma de l'université de Constantine.

Je remercie toutes mes amies et tous mes collègues pour leur soutien moral et pour tellement de choses.

Table des matières

Liste des figures

Abréviations

TOO	Technique Orientée Objets
OOT	Object Oriented Technique
GUI	Graphical User Interface
RAD	Développement Rapide d'Applications
COO	Conception Orientée Objets
OOD	Object Oriented Design
AOO	Analyse Orientée Objets
OOA	Object Oriented Analysis
MOO	Modélisation Orientée Objets
OOM	Object Oriented Modeling
OMT	Object Modeling Technique
OO	Object Oriented
UML	Unified Modeling Language
OOSE	Object Oriented Software Engineering
FDLF	Fast Decoupled Load Flow
NRLF	Newton Raphson Load Flow

Introduction générale

1. Pourquoi l'orientée objets

De nombreuses applications ont été développées ces dernières années en utilisant la Technologie orientée objets (TOO). Cet engouement pour la TOO se justifie largement aujourd'hui. En effet, les systèmes informatiques (logiciels) sont de plus en plus hétérogènes et plus complexes, de plus ils ont une durée de vie assez longue dans laquelle ils subissent des adaptations et des modifications (maintenance). La TOO s'appuie sur le métaphore des objets communiquant entre eux. Selon la TOO, un système peut être vu comme un ensemble d'objets qui collaborent pour assurer une mission globale.

Pour satisfaire les attentes mises sur les approches orientées objets, des méthodes et des outils pour mener à bien et maîtriser le processus de conception sont nécessaires. Donc, comme pour la conception de tout système complexe, apparaît la nécessité de modèles préalables sous forme de schémas et de plans. Ces modèles sont le produit des activités recommandées dans les phases dites d'analyse et de conception par objets.

2. Motivation

Dans le domaine de réseaux électriques, à cause de l'alimentations des charges plus exigeantes, la nécessité de maîtrise des régimes de fonctionnements perturbés, l'intégration de nouveaux systèmes de protection intelligents et l'optimisation économique des systèmes électriques, il est question de réseaux électriques de plus en plus complexes. C'est pourquoi, les gens de réseaux électriques, comme tous les autres domaines impliquant des systèmes complexes, ont migré vers cette TOO pour développer les outils informatiques pouvant satisfaire ces nouveaux besoins [1,2,3].

Les logiciels utilisés actuellement dans l'industrie de l'énergie électrique comme supports pour la planification et le fonctionnement sont, en général, très efficaces. Cependant,

ces logiciels possèdent des caractéristiques qui, face aux nouvelles conditions, ont besoin d'être mis à jour [1,4,5]. Dans beaucoup de cas, le changement d'une partie spécifique du code produit de grands effets sur d'autres routines du programme [1] et finissent par ajouter des problèmes à ceux qui avaient menés au processus de maintenance. La solution à ces problèmes semble être délicate puisque dans la plus part des cas, elle nécessite la re-écriture de grandes parties de codes [6,7].

En dépit du succès et des caractéristique des logiciels existants, ils constituent des obstacles face aux nouveaux défis que présentent les réseaux électriques. C'est pourquoi, une nouvelle génération de logiciels basés sur la TOO est considérée, avec les caractéristiques suivantes :

- supporter des structure de données pour un grand nombre d'applications ;
- une plus grande facilité pour le développement, la mise à jour et l'extension du code, permettant une grande souplesse dans l'inclusion des modèles des nouveaux équipements ainsi que les méthodes d'analyse ;
- degré de modularité élevé et réutilisation du code existant, sans perte d'efficacité.

D'un autre coté, l'une des difficultés de l'enseignement des réseaux électriques est qu'il y a beaucoup d'identités à introduire et à imaginer en même temps. En plus, il y a souvent des calculs très lourds à faire pour pouvoir voir les effets de certaines modélisations ou analyses. Pour permettre aux étudiants de se focaliser sur ces effets sans qu'ils soient perdus dans les énormes calculs, un logiciel de simulation de réseaux électriques est plus que nécessaire [8,9].

Les phénomènes physiques ainsi que les données d'un réseau électrique sont mieux assimilées si l'information est représentée sous forme graphique contrairement à la forme numérique. Pour des systèmes complexes, où les interactions homme-machine sont nombreuses, la spécification de la GUI (Graphical User Interface) par une approche orientée objets offre des avantages indéniables en terme de génie logiciel. La partie la plus importante de la GUI concerne la représentation des diagrammes unifilaires ou l'éditeur graphique.

3. Objectifs

Basés sur la puissance de la TOO et exploitant les moyens graphiques disponibles dans la majorité des outils de développement rapides d'applications (RAD), plusieurs logiciels ont été développés récemment pour l'enseignement, la recherche et l'entraînement dans le domaine de réseaux électriques [1,2,8,9,10,11]. Nos travaux de thèse se situent dans ce contexte. Il s'agit de développer une plate forme logicielle pour l'analyse des réseaux électriques.

La stratégie retenue quant à l'architecture des simulateurs de réseaux électriques a conduit à quatre grandes parties :

- Un éditeur graphique est spécialement développé pour la conception des diagrammes unifilaires des réseaux électriques avec des fenêtres et des boites de dialogue. Il utilise des symboles graphiques pour dessiner les éléments du réseau électrique tels que les jeux de barres, les lignes de transmission, les charges, les générateurs, etc. Il est également possible de déplacer, supprimer, modifier ou éditer les différentes fenêtres de données et de résultats.
- Une base de données visuelle est développée pour que l'utilisateur puisse faire entrer et modifier les données avec souplesse sur écran. Les données sont liées au diagramme unifilaire et aux applications à exécuter.
- Les applications qui simule le fonctionnement d'un réseau électrique, les applications réalisées actuellement dans cet outil sont l'écoulement de puissances et le calcul des courants de court circuit.
- Les facilitées de calcul telles que les opérations sur des matrices et l'utilisation des matrices creuses.

Actuellement la MOO des réseaux électriques fait sortir deux grandes structures de classes : la structure des éléments physiques du système électrique et les méthodes d'analyse (applications) appliquées au système ou sur ces éléments. Dans ce travail, le processus de création des structures de classes représentatives des entités diverses du réseau électrique dans son ensemble sont divisées en abstractions distinctes. On considère deux abstractions principales: le réseau électrique (classes des éléments physiques) et les fonctions de calcul (classes des applications).

La majorité des travaux apparus ces dernières années ont classifié les classes des éléments physiques du réseau électrique selon une hiérarchie tirée de sa structure réelle [1,4]. Cette classification est basée en premier lieu sur le nombre de jeux de barres de connexion que chaque élément possède.

Deux applications sont considérées, l'écoulement de puissances et le calcul des courants de court circuit. D'une manière générale, chaque application à besoin d'une série d'informations sur le réseau électrique et ses éléments constituants pour monter les structures mathématiques (matrices, systèmes linéaires, ensemble d'équations algébriques, etc.). C'est pourquoi, dans ce travail, les applications sont représentés par une structure hiérarchique qui dérive directement de la classe qui représente le réseau électrique.

4. Structure de la thèse

— Le Chapitre 1 introduit et présente les différents concepts fondamentaux de la technique orientée objets. Il s'agit de donner les définitions strictes et précises de ces concepts. Il présente également la notation UML et une brève description des patrons de conception.

— Le Chapitre 2 décrit les aspects généraux de l'application de la MOO pour la conception du logiciel. Ce chapitre présente les grandes abstractions des réseaux électriques adaptées.

— Le Chapitre 3 décrit les structures de classes des éléments physiques du réseau électrique, ainsi que leurs implémentations.

— Le Chapitre 4 présente deux applications avec leurs structures de classes et leur implémentation, il s'agit de l'écoulement de puissances et le calcul des courants de court circuit.

— Le Chapitre 5 présente l'environnement de la plate forme logicielle développée à travers un exemple visant à montrer l'usage qui peut être fait.

— Enfin, la Conclusion qui résume brièvement ce travail et les contributions majeures, et une discussion des travaux futurs conclut la présentation.

Chapitre 1

Approche Orientée Objets

1.1 Introduction

Il s'agit dans ce chapitre de présenter l'approche orientée objets ou les technologies orientées objets (TOO) et de mettre en évidence leur intérêt dans le processus de développement d'applications logicielles. Nous décrivons d'abord les principes et les mécanismes de base de l'orienté objets, ensuite nous évoquons le cadre normatif et les différents modèles de représentation objets du standard UML (Unified Modeling Language) en se basant sur les définitions et les descriptions présentées dans les ouvrages des initiateurs[12,13,14,15] dont nous disposons et enfin, nous dégageons l'intérêt des TOO dans le développement des applications réseaux électriques.

1.2 Les fondements de l'orienté objets

Les prémisses de la notion d'objet datent du projet du missile Minuteman en 1957. La conception et la simulation du fonctionnement de ce missile reposaient sur une poignée de composants logiciels prenant en charge la partie physique du missile, les trajectoires, les différentes phases de vol,… Le fonctionnement du système global reposait sur l'échange de messages d'information entre les différents composants. Chaque composant logiciel était conçu par un spécialiste et possédait ses données privées. De même le composant était virtuellement isolé du reste du programme par l'ensemble de ses méthodes servant d'interface [12].

Au cours des années soixante, le besoin de trouver des solutions plus générales aux problèmes de simulation, a conduit Ole-Johan Dahl et Kristen Nygaard à introduire les langages SIMULA, qui proposent et implémentent un ensemble de concepts révolutionnaires à l'époque. Ces concepts qui ne seront réellement exploités que beaucoup plus tard, ont été

dégagé par le Norvégien Kristen Nygaard alors qu'il travaillait dans la cellule de la défense Norvégienne.

L'objet se base sur une idée centrale : Pourquoi les informaticiens devraient-ils toujours convertir les éléments constitutifs de la partie du monde réel qu'ils sont amenés à informatiser (le domaine d'application) en concepts informatiques structurellement incompatibles avec la réalité ? En d'autres termes, ne serait-il pas plus économique de faire entrer directement la structure du domaine d'application dans la structure du problème ? La réponse à ces questions est oui. C'est la raison d'être de tout ce qui est labellisé **orienté objets** et c'est ce qui fait actuellement le succès.

Une approche orientée objets ne se limite pas à un ensemble de nouveaux concepts, elle propose une nouvelle manière de penser qui conduit à un nouveau processus de décomposition des problèmes. L'approche orientée objets à donc comme enjeu principal la bonne compréhension du monde réel (domaine d'application) afin d'en faire une représentation fidèle au sein du logiciel. L'étape la plus difficile pour un nouveau utilisateur de l'approche orientée objets est de changer sa philosophie de penser.

D'un autre coté, les problèmes à résoudre actuellement deviennent de plus en plus complexes. Au début, le simple résultat d'un calcul était intéressant. Maintenant, le résultat brut d'un calcul ne l'est plus! Il faut qu'il soit affiché graphiquement et automatiquement utilisé pour résoudre le problème d'ordre plus global dont il fait partie. Les premiers programmes pouvaient être réalisés par une seule personne. Aujourd'hui, les gros logiciels peuvent être réalisés par des dizaines, voire des centaines de personnes! Sans compter que les outils utilisés par ces personnes sont eux mêmes développés par autant de programmeurs. L'apparition de langages de programmation de plus haut niveau permet aujourd'hui de traiter des problèmes beaucoup plus complexes. Avec l'apparition de ces problèmes plus complexes, il devient de plus en plus difficile, voire impossible pour un seul individu de comprendre tous les aspects du problème à traiter. L'approche orientée objets permet donc de rendre plus indépendant le développement des différentes composantes d'un logiciel ou d'un projet.

1.3 Eléments de l'approche orientée objets

Il apparaît deux notions principales qui sont à la base de l'approche orientée objets : l'**objet** et la **classe**.

1.3.1 Objets

1.3.1.1 Nature d'un objet

Pour un être humain, un objet est :

- une chose visible ou tangible ;
- une chose qui peut être comprise intellectuellement;
- quelque chose vers qui une action est dirigée.

Un objet modélise une partie de la réalité qui existe dans le temps et dans l'espace. De plus, un objet peut être un mécanisme grâce auquel d'autres objets peuvent interagir. De manière plus formelle, en analyse orientée **objets**, un objet est défini comme suit [13] :

Un item individuel identifiable, réel ou abstrait, ayant un rôle bien défini dans le domaine du problème en traitement.

Dans certaines applications, les objets peuvent avoir des frontières bien définies et faciles à identifier. Par contre, dans d'autres applications, cette frontière est plus floue. Ceci nous amène à l'énoncé suivant décrivant encore mieux un objet du point de vue informatique[13] :

Un objet possède un état, un comportement et une identité; la structure et le comportement d'objets semblables sont définis par leur classe commune; le terme instanciation et objet sont synonymes.

Ainsi : **Objet = Etat + Comportement + Identité**

1.3.1.2 Etat d'un objet

Un objet possède généralement des **attributs** (un attribut est une information qui décrit l'objet) qui ont une valeur donnée à un moment donné dans le temps. Ainsi [13] :

L'état d'un objet comprend toutes les propriétés d'un objet (qui sont habituellement statiques) de même que la valeur présente (habituellement dynamique) de chacune de ces propriétés

En général, il est conseillé d**'encapsuler** l'état d'un objet plutôt que de l'exposer aux accès directs d'objets clients. En Java et en C++, cela signifie que l'état d'un objet devrait être gardé privé.

1.3.1.3 Comportement d'un objet

Les objets n'existent pas isolément mais interagissent plutôt avec d'autres objets en démontrant un comportement donné. Cette constatation conduit à la définition suivante [13] :
Le comportement d'un objet décrit comment celui-ci change d'état à la réception de messages d'autres objets et comment il transmet lui-même des messages aux autres objets.

En d'autres mots, le **comportement** d'un objet décrit son **activité externe visibl**e. Lorsqu'un **objet** effectue une **action** sur un autre objet, on dit qu'il lui transmet un message. En Java ou en C++, le passage de message signifie qu'un objet appelle une fonction membre (en programmation orientée objets, une "fonction membre" est aussi appelée "méthode") d'un autre. Lorsqu'un objet reçoit un message, celui-ci peut le faire changer d'état. On peut donc ajouter dans ce contexte que: l'état d'un objet représente l'effet cumulatif de son comportement.
En général, les messages sont passés via divers types de fonctions membres [13] :

- les **modificateur**s : fonctions membres qui peuvent modifier l'état de l'objet pour lequel elles sont appelées ;
- les **sélecteurs** : fonctions membres qui lisent l'état d'un objet sans le modifier ;
- les **itérateurs** : mécanisme qui permet de visiter toutes les parties d'un objet dans un ordre défini ;

Il y a évidemment deux autres types de fonctions membres connues :

- les **constructeur**s : créent et initialisent un objet ;
- les **destructeur**s : libèrent l'espace occupé par un objet (et son état) et détruisent ensuite l'objet lui-même.

En général, les **fonctions membres** d'un objet forment son **protocole** d'interaction avec le monde extérieur.

1.3.1.4 Identité d'un objet

L'identité d'un objet permet de faire distinction entre les objets de même type. On définit l'identité d'un objet comme [13] :

L'identité d'un objet est cette propriété qui le distingue de tous les autres objets.
Il ne faut pas ici confondre le nom de l'objet avec l'objet lui-même. Le nom de l'objet n'est qu'un identificateur qui permet d'accéder à ce dernier.

1.3.1.5 Relations entre objets

La relation entre chaque couple d'objets renferme l'hypothèse que chacun connaît l'autre, y compris les opérations qui peuvent être effectuées et le comportement qui en résulte. Deux sortes de hiérarchies d'objets sont d'un intérêt particulier dans l'approche orientée objets : les relations de liens (ou utilisation) et les relations d'agrégation.

A- Relation d'utilisation

Un **lien** est une **connexion physique** ou **conceptuelle** entre des objets. Un objet coopère avec d'autres objets grâce aux liens qu'il possède avec ceux-ci. Un lien est généralement unidirectionnel (mais pas obligatoirement). Un message est habituellement initié par un client et est dirigé selon un lien vers le serveur. De manière plus complète, un participant dans un lien peut jouer trois rôles:

- **acteur (ou client)** : objet pouvant agir sur d'autres objets mais sur qui les autres objets ne peuvent agir (il initie une interaction) ;
- **serveu**r : objet qui ne peut jamais agir sur les autres objets mais sur qui les autres objets peuvent agir (il est la cible de messages) ;
- **agen**t : objet qui peut agir sur les autres objets et sur qui les autres objets peuvent agir (il combine les caractéristiques d'un client et d'un serveur).

B- Relation d'agrégation

Alors que les liens dénotent une relation client/fournisseur, la contenance (ou agrégation) correspond à une hiérarchie ensemble/composant. Les agrégats dénotent une façon différente d'associer les objets. Par exemple, un objet peut en contenir un autre qui fait partie de son état. L'objet contenant peut informer le monde extérieur de ce qu'il contient et l'objet contenu peut éventuellement informer le monde extérieur qu'il appartient à un contenant si son interface le lui permet et si cette information est contenue dans son propre état (C'est d'ailleurs cet aspect qui différencie l'agrégat du simple lien) . L'agrégation ne signifie cependant pas exclusivement l'inclusion physique mais peut être conceptuelle.

1.3.2 Classes

1.3.3.1 Nature d'une classe

Un objet est un individu appartenant à une **classe**. La création d'un objet à partir de sa classe est appelée instanciation (on dit que l'objet est une instance d'une classe...). Une **classe** est définit comme étant [13] :

Un ensemble d'objets partageant une structure et un comportement communs.

Une classe décrit donc les caractéristiques générales d'un ensemble d'objets. Une classe comprend généralement:

- une **interface** qui permet l'interaction des instances de cette classe avec les autres objets du problème (la vision externe que la classe donne d'elle-même), elle décrit le domaine de définition et les propriétés des instances de cette classe ;
- une **implantation** réalisant l'interface (le comportement interne de la classe), elle contient le corps des opérations et les données nécessaires à leur fonctionnement.

L'interface d'une classe est divisée en trois parties [12,13] :

- **publique(public)** : une déclaration accessible à tous les clients ;
- **protégée(protected)** : une déclaration qui n'est accessible qu'à la classe, à ses amies et à ses sous-classes ;
- **privée(private)** : une déclaration qui n'est accessible qu'à la classe et à ses amies.

1.3.3.2 Relations entre classes

Les langages de programmation orientée objets supportent plusieurs types de relations entre classes à savoir : l'association, l'héritage, l'inclusion et l'utilisation [12,13].

A- Association

L'**association** signifie que des classes sont en relation mais que l'on n'a pas encore choisi leur niveau plus spécifique de relation ou encore que l'on ne désire que montrer leur association sans plus de détails.

B- Héritage

L'**héritage** permet de spécialiser une classe en dérivant une autre classe dont les propriétés sont plus spécifiques que celles de la classe dont elle dérive et qui ajoute aussi certaines fonctionnalités à la classe mère. Certaines propriétés peuvent demeurer inchangées et sont donc partagées par les deux niveaux de la hiérarchie. On dit que la classe de base est

une **superclasse** et que la classe dérivée est une **sous-class**e. Certains langages comme le C++ admettent l**'héritage multiple** (c'est-à-dire qu'une classe peut hériter de deux superclasses distinctes). L'héritage multiple peut causer de sérieux problèmes que le Java a préféré éviter en permettant seulement l'héritage simple et en offrant la possibilité de créer des interfaces (le mot réservé interface en Java).

C- Inclusion

L**'inclusion** (aussi appelée **agrégation** ou **composition**) est une relation entre les classes qui est parallèle à la relation d'agrégats entre les objets. Elle permet l'expression des relations de type : « maître et esclave », « une partie de », « composé de » etc. Dans l'agrégation l'une des classes est plus importante que l'autre.

D- Utilisation

L**'utilisation** est une relation qui survient quand une classe possède une (ou des) fonction membre dont l'un des arguments est une référence à un objet d'une autre classe.

1.3.3 Rôle des classes et des objets

Classes et objets sont des concepts séparés et pourtant intimement liés. Plus précisément, chaque objet est l'instance d'une certaine classe, et chaque classe a zéro ou plusieurs instances. Pour presque toutes les applications, les classes sont statiques ; par conséquent, leur existence, leur sémantique et leurs relations sont fixées préalablement à l'exécution d'un programme. De même, la classe de la plupart des objets est statique, ce qui signifie que, dès qu'un objet est créé, sa classe est fixée. Tout au contraire, les objets sont couramment crées et détruits à un rythme élevé durant la vie d'une application [13].

Au cours de l'analyse et au début de la phase de conception d'un projet, le développeur doit accomplir deux tâches principales :

- identifier les **classes** et les **objets** qui forment le vocabulaire du domaine du problème;
- inventer les **structures** par lesquelles des ensembles d'objets travaillent en coopération pour atteindre les comportements nécessaires à la solution du problème (mécanismes de mise en œuvre).

1.3.4 Classification

La **classification** est un moyen d**'ordonner les connaissance**s. En conception orientée objets, la reconnaissance de la similitude entre les choses permet d'exposer le caractère commun à l'intérieur des abstractions et les mécanismes et conduit à des architectures plus simples. Il n'y a malheureusement pas de recette pour identifier les classes et les objets.

L'identification des classes et des objets est la partie la plus difficile de l'analyse et de la conception orientée objets. L'expérience montre que cette classification peut être trouvée grâce à la découverte et l'invention. La découverte permet de reconnaître les abstractions et mécanismes qui forment le vocabulaire du problème. Par l'invention, on peut ensuite concevoir des abstractions plus générales et des mécanismes nouveaux qui permettent aux objets de collaborer. Le principal problème de la classification est qu'il y a autant de classifications que de raisons pour établir une classification! La classification dépend souvent de la façon dont on aborde un problème [13].

Quoiqu'il en soit, la conception d'une classification intelligente est un travail intellectuel difficile et la solution est atteinte de manière **incrémentale** et **itérativ**e. Ce développement incrémental de la classification a un impact direct sur la construction des classes et des hiérarchies dans la conception de systèmes complexes. En pratique, il est donc courant de choisir une certaine structure de classe assez tôt dans la phase de conception pour ensuite la réviser. Ce n'est que rendu à un certain moment dans la conception, et plus spécialement lorsque des clients ont utilisé la classe, qu'il est possible d'évaluer la qualité de la classification. Suite à cette évaluation, il est alors courant de décider de créer de nouvelles classes à partir de classes existantes, de séparer une grosse classe en plus petites classes (factoriser les classes), ou de créer une classe à partir de plus petites classes (composer une classe) [13].

1.4 Modélisation orientée objets

L'orientée objets est une technique de modélisation des systèmes. Ainsi, si une modélisation orientée objets porte ce nom avec succés, c'est qu'elle doit effectivement modéliser n'importe quel type de système sous forme d'objets. La modélisation par objets utilise les **classes** et les **objets** comme **blocs** de base.

1.4.1 Eléments de la modélisation par objets

La modélisation par objets comporte quatre éléments principaux : l'abstraction, l'encapsulation, la modularité et la hiérarchie. Sans cette ossature conceptuelle, le programme n'est pas orienté objets même si le langage de programmation est orienté objets.

1.4.1.1 Abstraction

Une abstraction est une représentation des caractéristiques essentielles d'un objet qui permettent de le distinguer de tous les autres. Une abstraction s'intéresse à l'apparence extérieure d'un objet et permet de séparer le comportement essentiel de l'objet de son implantation. C'est ce qu'on appelle la "barrière de l'abstraction" [12, 13].

Il existe plusieurs types d'abstractions:

- **abstraction descriptive (d'identité)** : un objet qui représente un modèle utile d'une composante du domaine du problème et de sa solution ;
- **abstraction d'action** : un objet qui offre un ensemble d'opérations générales dont chacune effectue le même type de fonction ;
- **abstraction de machine virtuelle** : un objet qui regroupe des opérations qui sont toutes utilisées par un niveau de contrôle supérieur, ou les opérations qui utilisent toutes un ensemble d'opérations d'un niveau plus élémentaire ;
- **abstraction fortuite (de coincidence)** : un objet qui contient un ensemble d'opérations qui n'ont aucun lien entre elles.

Ainsi :

L'abstraction se concentre sur les caractéristiques essentielles d'un objet selon le point de vue de l'observateur [13].

1.4.1.2 Encapsulation

Une abstraction devrait d'abord être conçue indépendamment de son implantation. L'implantation doit généralement demeurer secrète et ne doit que refléter l'abstraction désirée. En résumé, aucune partie d'un système complexe ne devrait dépendre des détails internes d'un autre. L'**abstraction** et l'**encapsulation** sont deux concepts complémentaires. L'**abstraction** concerne la **description du comportement extérieur** d'un objet tandis que l'**encapsulation** vise à **implanter** (mettre en œuvre) cette description ou ce comportement.

L'encapsulation est le procédé de séparation des éléments d'une abstraction qui constituent sa structure et son comportement. Elle permet de diviser l'interface contractuelle de la mise en œuvre d'un objet. L'encapsulation occulte les détails (secrets) de mise en œuvre d'un objet [13].

L'encapsulation est habituellement obtenue grâce au **masquage des informations**, qui permet de cacher tous les secrets d'un objet lorsqu'ils ne contribuent pas à ses caractéristiques essentielles. Souvent la structure de l'objet, ainsi que le codage de ses méthodes (opérations), sont cachés [12,13].

1.4.1.3 Modularité

Dans des projets logiciels d'envergure, l'utilisation de **modules** est essentielle pour gérer la complexité.

La modularité est la capacité qu'a un système d'être décomposé en un ensemble de modules cohérents et faiblement couplés [12,13,14].

Dans des langages comme le C++ et Java, les classes et les objets forment la structure logique d'un système, les modules contiennent les abstractions décrites par ces classes et forment l'architecture physique du système. Pour des systèmes incluant des milliers de classes, les modules sont un moyen de traiter la complexité.

La modularité regroupe les entités en unités discrètes.

La création de modules vise à **regrouper des classes pouvant être compilées séparément mais qui sont reliées à des classes contenues dans d'autres modules**. Les liens entre les modules sont les hypothèses que les modules font les uns sur les autres. Un module, comme une classe, comprend une interface et une implantation. **Modularité et encapsulation sont donc des concepts très voisins**.

Il faut remarquer que la division d'un système en modules est aussi difficile que de décider de sa division en abstractions. Une bonne division est très avantageuse. D'ailleurs, le but général de la modularisation vise à réduire le coût du logiciel en permettant de concevoir et de réviser des modules de manière indépendante. La structure de chaque module doit être assez simple pour être comprise facilement. Il faut également qu'il soit possible de modifier un module sans connaître ni affecter la structure des autres modules.

1.4.1.4 Hiérarchie

L'**abstraction** est une bonne chose mais il est souvent difficile de maîtriser toutes les abstractions à cause de leur nombre. L'**encapsulation** permet de gérer partiellement la complexité. La **modularité** aide aussi en regroupant les abstractions qui sont reliées. Mais cela n'est pas suffisant. En effet, les abstractions forment aussi une **hiérarchi**e.

On définit une hiérarchie comme étant un classement ou ordonnancement des abstractions [13].

Il existe deux types importants de hiérarchies :

- les **hiérarchies d'existence ou de classe** ("est un") ;
- les **hiérarchies d'appartenance ou d'objets** ("partie de").

Un premier exemple de hiérarchie est **l'héritag**e. Ce dernier dénote une relation "est un" d'une abstraction. Une **abstraction** au bas d'une hiérarchie spécialise une abstraction plus générale.

Un second exemple de hiérarchie est **l'appartenance** ("partie de") pour laquelle une abstraction **fait partie** d'une autre abstraction. Alors que les hiérarchies "est un" désignent des relations de généralisation/spécialisation, les hiérarchies "partie de" décrivent des relations d'agrégation.

1.4.2 Phases de développement orienté objets d'un système

Les différentes phases du développement d'un système, d'un point de vue orientée objets sont : l'analyse orientée objets, la conception orientée objets et la programmation orientée objets.

1.4.2.1 Analyse orientée objets

L'**analyse orientée objets** (AOO ou OOA : Object Oriented Analysis) à pour but la compréhension du système que l'on doit développer et l'élaboration d'un modèle logique du système. Ce modèle est basé sur des objets naturels issus du domaine d'application. Ces objets contiennent des données et ont leurs propres comportements à partir desquels on peut exprimer le comportement du système entier [14]. Ainsi l'AOO est définie comme suit :

L'AOO est une méthode d'analyse qui examine les besoins d'après la perspective des classes et objets trouvés dans le vocabulaire du domaine du problème (domaine d'application).

1.4.2.2 Conception orientée objets

La **conception orientée objets** (COO ou OOD : Object Oriented Design) signifie que le modèle d'analyse et conçu, elle met l'accent sur la structuration appropriée et efficace d'un système complexe. Ainsi :

La COO est une méthode de conception incorporant le processus de décomposition orientée objets et une notation permettant de dépeindre à la fois les modèles logiques/physiques et statiques/dynamiques du système à concevoir [13].

1.4.2.3 Programmation orientée objets

La **programmation orientée objets** (POO ou OOP : Object Oriented Programming) utilise les objets et non les algorithmes comme blocs fondamentaux, chaque objet est une instance d'une certaine classe et les classes sont reliées l'une à l'autre par des relations d'héritage. Si l'un de ces éléments fait défaut, ce ne sera pas un programme orienté objets. Plus précisément, programmer sans héritage n'est pas orienté objets. Ainsi :

La POO est une méthode d'implantation par laquelle les programmes sont organisés en un ensemble d'objets coopératifs, chaque objet représentant une instance d'une classe, chaque classe faisant partie d'une hiérarchie de classes unies par des relations d'héritage[14].

A la lumière de cette définition, certains langages de programmation sont orientés objets et d'autres ne le sont pas. Un langage est orienté objets si et seulement si il répond aux conditions suivantes [12,13,14] :

- il supporte des objets qui sont des abstractions de données avec une interface d'opérations nommées et un état interne caché ;
- les objets ont un type associé (la classe) ;
- les types (les classes) peuvent hériter des attributs venant de super-types (les super-classes).

1.4.3 Processus de développement orienté objets

Dans ce processus, **l'analyse** et la **conception** sont étroitement liés et la frontière les séparant est floue. Le processus de développement s'intéresse aux activités suivantes :

- identifier les classes et les objets à un **niveau donné d'abstractio**n ;
- identifier la sémantique des classes et des objets ;

— identifier les relations entre les classes et les objets ;
— spécifier l'interface et l'implantation des classes et des objets.

1.4.3.1 Identification des classes et des objets

L'identification des classes et des objets permet d'établir les bornes du problème et de décomposer celui-ci en objets. Durant la phase d**'analys**e, cette étape permet de mettre à jour les abstractions qui forment le **vocabulaire du problèm**e. Durant la phase de **conception**, cette étape permet de créer de nouvelles abstractions et même de concevoir des abstractions de bas niveau qui permettent de créer des abstractions de niveau supérieur. En cours d'implantation, cette étape permet de découvrir des points communs entre les abstractions, ce qui contribue à simplifier l'architecture du système. Le résultat de cette étape est un **dictionnaire des données** qui est mis à jour graduellement en cours de développement. Au début, une liste des classes et des objets importants est suffisante. Dès ce moment, il convient d'utiliser des noms reflétant leur sens (sémantique).

1.4.3.2 Identification de la sémantique des classes et des objets

La sémantique d'une classe comprend son rôle, ses responsabilités de même que ses opérations. Le but de cette étape est d'établir le comportement et les attributs des abstractions identifiées à l'étape précédente. Les abstractions précédemment définies sont raffinées pour inclure une distribution mesurable de leurs responsabilités.

Durant la phase **d'analys**e, cette étape vise à allouer les responsabilités en fonction des différents comportements du système. A la phase de **conception**, cette étape permet de définir une séparation nette entre les différentes parties de la solution.
Au début du projet, les rôles peuvent être décrits en langage familier. Plus tard, ils doivent faire l'objet de spécifications précises des **protocoles** (ensemble des opérations qu'un client peut effectuer sur une abstraction) complets de chaque abstraction et de la description précise de la **signature** (l'ensemble des paramètres formels de même que le type de valeur de retour des méthodes de chaque opération) [12,13,14].

En plus, il est aussi pertinent de définir des **diagrammes d'objet**s, des **diagrammes d'interaction** qui permettent d'établir la **sémantique des scénarios** créés. Ces diagrammes permettent de saisir de manière formelle les planches **(storyboard**s) des scénarios et témoignent de la répartition explicite des responsabilités entre les différents objets.

1.4.3.3 Identification des relations entre les classes et les objets

Durant la phase d'**analyse**, cette phase sert à identifier les associations entre les classes et entre les objets, incluant certaines relations d'**héritage** et d'agrégation. L'existence d'une association met en évidence une dépendance sémantique entre deux abstractions de même que leur capacité à naviguer de l'une à l'autre (est-ce qu'une abstraction est seulement visible d'une autre ou se voient-elles mutuellement, etc).

Durant la phase de **conception**, cette phase sert à spécifier les collaborations qui forment les mécanismes de l'architecture, de même que les regroupements de haut niveau de classes en catégories et des modules en sous-systèmes.

Au fur et à mesure de l'avancement de l'implantation, les relations sont raffinées jusqu'à transformer les associations simples en relations plus spécifiques comme l'instanciation (inclusion) et l'utilisation.

1.4.3.4 Implantation des classes et des objets

Durant l'analyse, le but d'**implanter** les classes et les objets est de raffiner les abstractions. Durant le design, cette phase permet de donner une représentation tangible des abstractions et de permettre le raffinement successif des versions exécutables du processus. Les produits de cette phase sont les décisions sur la façon dont les abstractions seront **implantées physiquement**. Au début, du pseudo-code est suffisant. Plus l'implantation avance, plus ce pseudo-code se transforme graduellement en code réel.

1.4.4 Méthodes orientées objets

Pour développer des systèmes logiciels orientés objets de qualité, il faut utiliser une méthode de développement. Il n'existe pas de méthode universelle ni pour l'analyse ni pour la conception. Beaucoup de méthodes traitent ces deux phases du cycle de vie d'un logiciel, d'autres ne traitent que l'analyse ou la conception. De nouvelles méthodes sont introduites chaque année pour compléter ou palier les défauts des anciennes méthodes.

Alors que les concepts orientés objets existaient depuis les années soixante, les méthodes orientée objets ne sont apparues que lors des trois dernières décennies. Ceci est du essentiellement à [12,13] :

- les concepts orientés objets demandaient du temps pour mûrir dans nos esprits ;
- il était assez difficile de penser « orienté objets » tant que les langages industriellement répandus n'étaient pas orientés objets ;

— ce n'est que lors des trois dernières décennies que sont apparus les systèmes complexes de grande dimension ce qui a conduit à élaborer de nouvelles méthodes s'adaptant à ces besoins.

1.4.4.1 Méthode de conception OOD

En 1983, Grady Booch proposait une méthode de conception dite orientée objets. La deuxième version de cette méthode a été présentée en 1990 [12,13]. Cette méthode adoptait une démarche en 4 étapes pour concevoir un système :

— Identifier les classes et les objets à un niveau d'abstraction donnée.
— Identifier la sémantique de ces classes et de ces objets. Le développeur doit détailler la représentation interne des classes de manière à comprendre leur fonctionnement et leur rôle précis. Ceci permet de déterminer les interfaces de chaque classe.
— Identifier les relations entre les classes et les objets.
— Implémenter ces classes et ces objets.

Booch précise clairement qu'il ne définit pas une méthode d'analyse, mais uniquement une méthode de conception qui peut être utilisée en aval d'une méthode d'analyse.

1.4.4.2 Méthode d'analyse OOA

La technique d'analyse par objets proposée par Coad et Yourdon est une tentative d'incorporation des meilleurs idées proposées. OOA utilise de manière explicite l'héritage pour la mise en commun des attributs et des services. La démarche pour obtenir un modèle OOA se compose de cinq activités principales [12] :

— Trouver les classes et les objets.
— Identifier les structures.
— Identifier les sujets.
— Définir les attributs.
— Définir les services.

1.4.4.3 Méthode d'analyse et de conception OMT

La méthode OMT (Object Modeling Technique) , proposée par Jumes Rumbaugh et Mechael Blaha, s'applique à tous les processus de développement d'un logiciel, de l'analyse à l'implantation [12]. Cette méthode utilise trois vues différentes, chacune capturant les aspects importants du logiciel, ces trois vues sont :

— Le modèle objet qui représente l'aspect statique d'un logiciel (définitions des classes, relations d'héritages, d'agrégation,…).
— Le modèle dynamique qui présente le comportement du logiciel au cours du temps.
— Le modèle fonctionnel qui prend en compte l'aspect fonction de transformation du logiciel.

Chacun de ces modèles contient des références aux entités des autres modèles, ils ne sont donc pas complètement indépendants. La méthodologie proposée est indépendante des langages de programmation et utilise une notation graphique uniforme pour toutes les phases. Les trois modèles séparent un système en un ensemble de vues qui sont manipulées et qui évoluent tout au long du cycle de développement (analyse, conception et implémentation pour OMT). Le but de la construction du modèle objet OMT est de fournir le cadre de travail essentiel dans lequel les modèles dynamiques et fonctionnels vont se placer. Les objets sont considérés comme des composants de base qui doivent capturer les éléments de réalité que l'analyste considère comme importants pour une application [12].

Le modèle dynamique décrit les aspects temporels, les séquences d'opérations et les évènements qui entraînent des changements d'état au sein d'une classe. Le rôle du modèle dynamique est donc de présenter les différents aspects de contrôle du système. Ceci se traduit au niveau de la notation graphique par des diagrammes d'état et des séquences d'événement.

Le modèle fonctionnel décrit les transformations apportées par le système sur les fonctions réalisées et ceci sans se préoccuper de la manière dont cela est réalisé ni quand cela intervient. Le modèle fonctionnel est représenté avec des diagrammes de flots de données montrant les dépendances entre les données en entrée et celles en sortie des processus de traitement chargés de réaliser les fonctions précises du logiciel.
La méthode OMT semble relativement complète pour aborder une large catégorie de problèmes.

1.4.4.4 Méthode Objectory d'Ivar Jacobson

La méthode Objectory (ou OOSE : Object Oriented Software Engineering) est une autre méthode qui aborde aussi bien l'analyse que la conception des systèmes de taille importante. Elle couvre tout le cycle de développement d'un logiciel et propose un processus de développement qui produit cinq modèles [12] :

— modèle des besoins ;

— modèle d'analyse ;

— modèle de conception ;

— modèle d'implémentation ;

— modèle de test.

1.5 Langage unifié de modélisation(UML)

Jusqu'à tout récemment, les systèmes logiciels, qu'ils soient orientés objets ou non, ont toujours souffert d'un manque de documentation, non seulement une fois le produit complété mais également lorsque le produit est en cours de développement. C'est pour cette raison que le langage graphique UML (pour Unified Modeling Language) a été conçu. Il permet de décrire un système de manière graphique, un peu de la même manière que les circuits numériques sont décrits par des plans et des diagrammes temporels.

1.5.1 Origine et objectifs de UML

La spécification du langage de description de systèmes UML est une tâche qui a été entreprise par Grady Booch et Jumes Rumbaugh en 1994. Cet effort de spécification de UML visait à unifier des méthodes de description de systèmes orientés objets qui avaient été proposés respectivement par Booch, Rumbaugh et Jacobson. Il ne faut pas oublier que les objets logiciels peuvent avoir des structures et des comportements très complexes et il y avait par conséquent un besoin de les décrire de la manière la plus fidèle possible. C'est en Janvier 1997 que Booch, Rumbaugh et Jacobson proposèrent la version 1.0 de UML qui a évolué depuis. On constate donc que l'introduction de UML est relativement récente [15].

UML vise principalement à décrire les systèmes complexes en utilisant des concepts orientés objets. Les systèmes ne sont pas obligatoirement des systèmes logiciels. Par exemple, le fonctionnement d'organisations comme les banques, les compagnies d'assurance, ou les universités peuvent très bien être décrites en langage UML. Cependant l'un des objectifs principaux de UML est de permettre la création de modèles utilisables à la fois par des humains et des machines. UML est donc à la fois bien structuré pour être utilisable par des machines mais il est également très graphique, permettant ainsi aux humains de saisir les principaux éléments du modèle de même que les détails plus spécifiques.

Il existe deux types de visualisations : la visualisation des points fonctionnels qui décrit le fonctionnement externe du système et la visualisation logique qui décrit à la fois la structure statique (classes, objets, relations) et la structure dynamique (collaboration des objets lorsqu'ils s'échangent des messages pour réaliser une fonctionnalité donnée, etc.) du système. La structure statique d'un système est décrite par le diagramme de classes (et le diagramme d'objets). La structure dynamique du système s'exprime via les diagrammes de collaboration (ou d'interaction), les diagrammes séquentiels, et les diagrammes d'activité.

1.5.2 Diagramme de points fonctionnels : diagramme de cas d'utilisation

Un système doit accomplir certaines fonctions précises répondant aux besoins des utilisateurs. Avant tout, il convient donc de bien identifier ce que le système doit faire en identifiant ses points fonctionnels. L'ensemble des points fonctionnels décrit le fonctionnement global du système.

En UML, un point fonctionnel est représenté par **une ellipse dans laquelle le nom du point fonctionnel est inscrit**. L'utilisateur d'un point fonctionnel est pour sa part représenté par un **petit bonhomme** de fil de fer communément appelé « stickman ». Le lien qui existe entre un utilisateur et le point fonctionnel est représenté par une **flèche** allant de l'utilisateur vers le point fonctionnel. Des points fonctionnels peuvent également être reliés entre eux. En général, le diagramme des points fonctionnels permet d'illustrer graphiquement l'interaction entre l'usager et le système ou entre différents sous-systèmes. La figure 1.1 montre les symboles utilisés pour illustrer les points fonctionnels dans un diagramme de points fonctionnels.

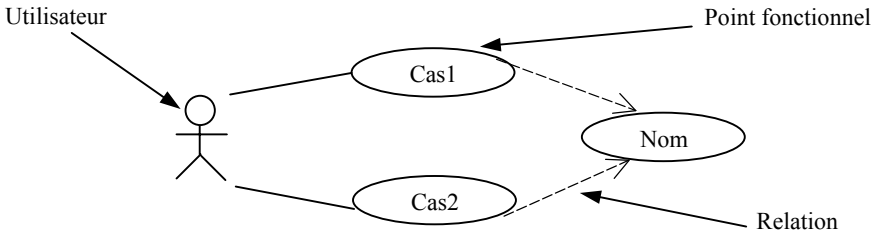


FIGURE 1.1 Symboles UML pour la description des points fonctionnels

1.5.3 Visualisation statique : Diagramme de classes

Ce diagramme décrit le système en termes de classes et de relations entre classes. Ce diagramme sert de base à tous les autres diagrammes (sauf le diagramme des points fonctionnels) y compris les diagrammes du modèle dynamique du système.

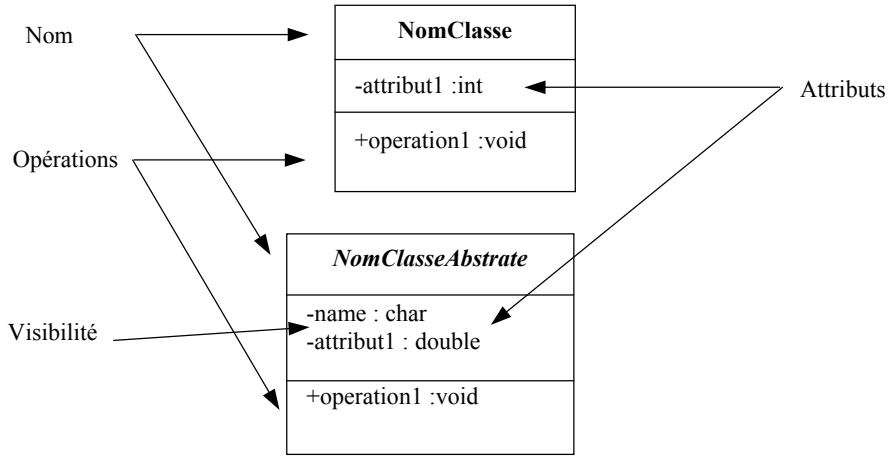


FIGURE 1.2 Symboles des classes en UML avec identification des niveaux

Le symbole pour représenter une classe est un **rectangle divisé en trois compartiments**. Normalement, la valeur des attributs d'un objet n'est pas accessible directement par un autre objet. On dit que les attributs sont masqués à l'intérieur d'un objet. L'interaction entre les objets s'opère en activant différentes opérations déclarées dans l'interface. Ainsi, les opérations de l'interface d'une classe sont accessibles à d'autre classes. Plusieurs niveaux de protections sont généralement disponibles. Par exemple C++ offre trois niveaux d'accessibilité :

- **Niveaux privé -** : Seuls les objets de même classe et les fonctions et les objets amis peuvent y accéder (c'est le niveau de protection le plus élevé).
- **Niveaux protégé #** : Seuls les objets de même classe et les objets de classes dérivées peuvent y accéder (c'est le niveau de protection intermédiaire).
- **Niveaux publique +** : L'effet de l'encapsulation est éliminé. Les opérations et les attributs placés dans ce niveau sont accessibles par tous (c'est le niveau de protection le plus faible).

Un diagramme de classes décrit non seulement les classes du système mais également, les relations entre celles-ci.

1.5.3.1 Associations

La relation d'association exprime une abstraction du lien qui existe entre les objets. Normalement, on doit spécifier les rôles des classes et la nature de cette association. Il existe plusieurs types d'associations: normale, récursive, agrégation par référence et agrégation par composition.

Une **association normale** est une **connexion** entre classes. Elle possède un nom et est généralement navigable de manière bidirectionnelle. Si c'est le cas, l'association peut avoir un **nom** pour chaque sens de navigation. Elle possède une **multiplicité** et peut être dotée de **rôles**. La figure 1.3 donne un exemple d'association normale avec des rôles et une multiplicité.

En UML, **l'association simple** est représentée par une **ligne** reliant les deux classes dans un diagramme de classes. Le **nom** de l'association apparaît près de celle-ci. La **multiplicité** apparaît aux **extrémités** de l'association pour spécifier combien d'objets d'une classe à cette extrémité peuvent être associés à la classe située à l'autre extrémité de l'association. Par exemple, dans l'association montrée à la figure 1.3, il peut y avoir entre 0 et * objets de la classe **Disjoncteur** associés à un objet de la classe **Transformateur.** Ici, * signifie "indéterminé". Comme il n'y a pas de multiplicité à l'extrémité **Transformateur** de l'association, cela signifie que la multiplicité n'a pas d'importance ou qu'elle n'a pas encore été spécifiée dans le modèle. Une association peut être dotée d'une navigabilité représentée par une **flèche** illustrant le sens dans lequel l'association est valable.

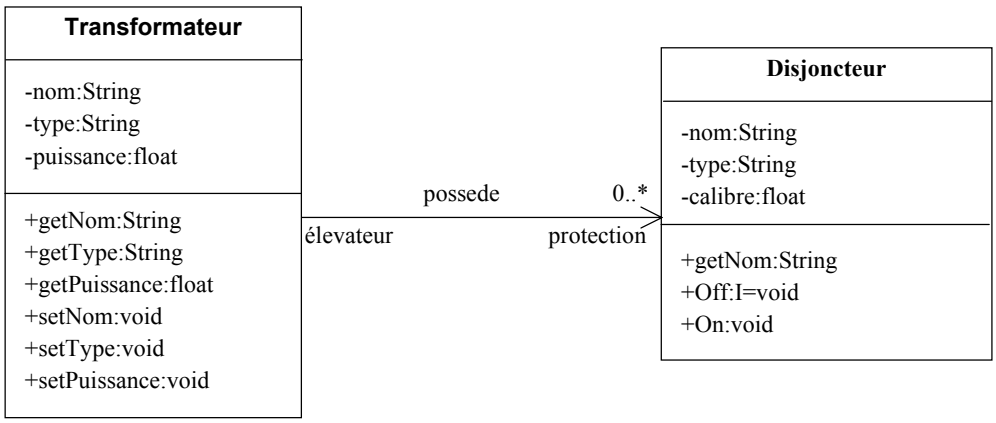


FIGURE 1.3 Association normale avec navigabilité illustrée de manière explicite

Une **association récursive** illustre qu'une classe peut être associée à elle-même. Elle représente une connexion sémantique entre des objets mais avec la différence que ceux-ci

appartiennent à la même classe. Par exemple, si l'on considère une liste liée simplement chaînée, un élément de la liste est rattaché à 0 ou 1 élément de la même classe. La figure 1.4 montre ce cas d'association récursive entre objets appartenant à la même classe.

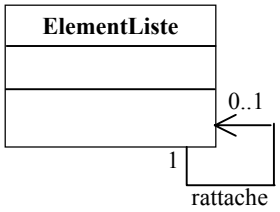


FIGURE 1.4 Association récursive

L'agrégation **par référence** est un cas intéressant d'association qui représente une relation **tout-partie**. Le symbole de l'agrégation par référence est un **losange** placé du côté **tout** de l'association. Ce losange ne peut apparaître qu'à une seule extrémité de l'association. Si on reprend l'exemple de la figure 1.3 en raffinant l'association entre la classe **Transformateur** et la classe **Disjoncteur,** on peut rendre l'association entre les deux classes plus précise en spécifiant qu'un **Transformateur** possède un **Disjoncteur** via une agrégation par référence tel que montré à la figure 1.5. L'agrégation par référence est ici correcte car si le **Transformateur** disparaît, le **Disjoncteur** continue à exister .

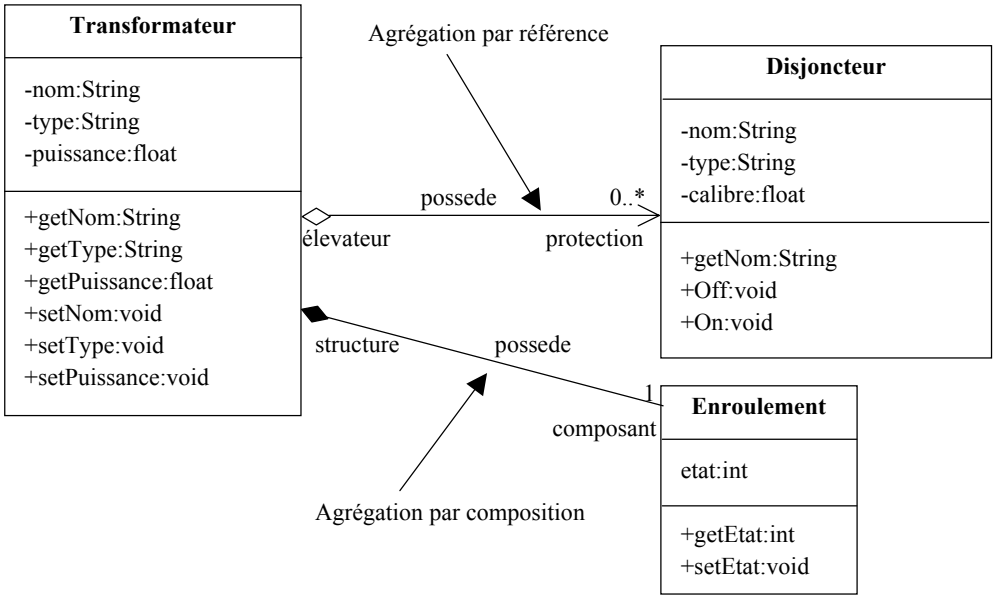


FIGURE 1.5 Agrégation par référence et agrégation par composition

La figure 1.5 montre également un autre type d'association: l'agrégation **par composition**. Tout comme l'agrégation par référence, l'agrégation par composition sert à décrire une relation **tout-partie** à la différence que cette relation implique une **inclusion physique** de la partie. L'agrégation par composition se représente par un **losange plein** placé à l'extrémité de la relation. La figure 1.5 montre un exemple d'agrégation par composition entre la classe **Transformateur** et la classe **Enroulement.** Dans ce cas, la composition est essentielle car un **Transformateur** possède un **Enroulement** et quand il disparaît, l'**Enroulement** disparaît lui-aussi (devient non fonctionnel). On remarque que l'agrégation par composition supporte également les rôles et la multiplicité.

1.5.3.2 Généralisations ou héritage

Une **généralisation** est une relation entre une classe **générale** et une classe plus **spécifique**. La classe spécifique est appelée **sous-classe** tandis que la classe générale est appelée **super-classe**. La sous-classe **hérite** des attributs et opérations de la super-classe. Une sous-classe peut évidemment être la super-classe d'une autre sous-classe dans une matrice d'héritage. En langage UML, la généralisation se représente par une **flèche** allant de la classe spécifique à la classe générale. Lorsqu'une classe spécifique hérite de plusieurs classes générales, on parle alors d'héritage multiple. L'exemple ci dessous montre une généralisation appliquée à un certain type de machines électriques. Les lignes fléchées pointent vers des classes de plus en plus générales.

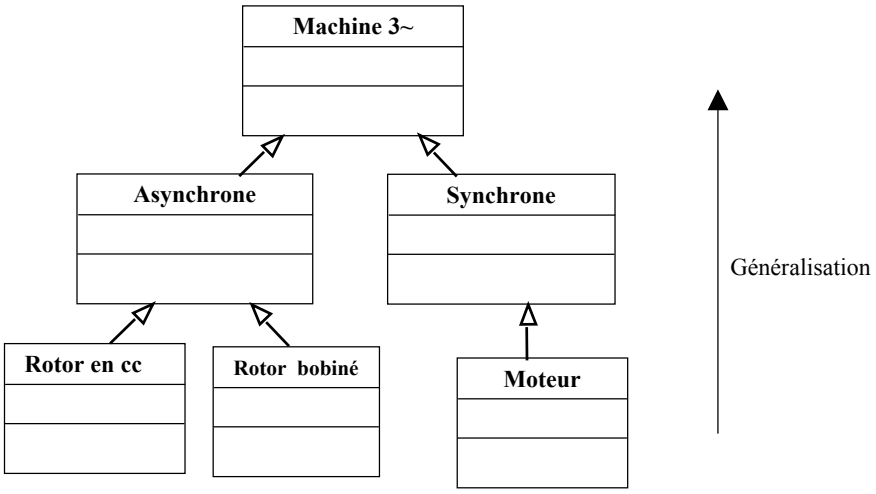


FIGURE 1.6 Généralisation des classes

La spécialisation permet l'ajout de nouvelles caractéristiques à un ensemble d'objets qui n'ont pas été identifiées auparavant dans la hiérarchie des classes. La spécialisation permet donc l'extension des capacités d'une manière cohérente à un ensemble d'objets.

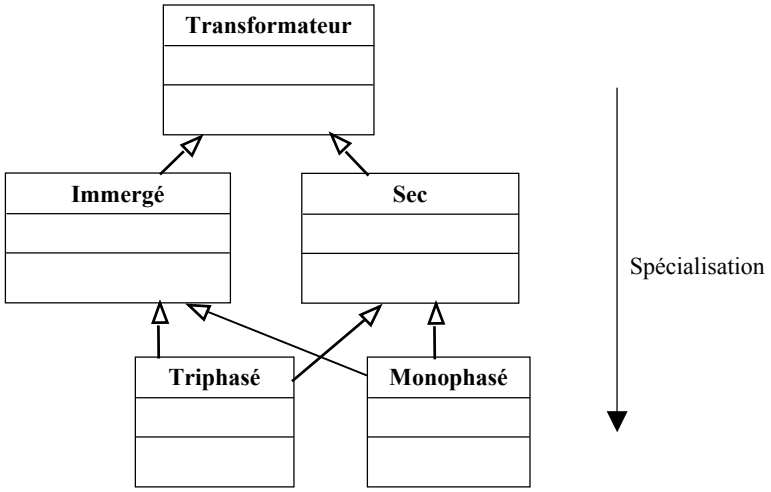


FIGURE 1.7 Spécialisation des classes

La généralisation et la spécialisation sont deux points de vue opposés. Dans la pratique elles sont utilisées en même temps.

1.5.3.3 Interfaces

Les **interfaces** représentent un type spécial de classe contenant uniquement des prototypes d'opérations sans implantation. La responsabilité d'implanter le code de ces opérations incombe à une classe qui implante cette interface. En UML, une interface est représentée de la même manière qu'une classe mais en ajoutant le stéréotype **interface** au dessus du nom de la classe. Le symbole indiquant qu'une classe implante une interface est une **flèche pointillée** allant de la classe vers l'interface.

1.5.3.4 Packages

Pour les systèmes logiciels formés d'un très grand nombre de classes, il convient de regrouper celles-ci en entités logiques distinctes. En langage UML, ces entités logiques s'appellent des **packages ou paquets**. Chaque package est muni d'une interface formée de ses classes publiques qui lui permet de communiquer sa fonctionnalité aux autres packages qui peuvent l'importer. Un package peut également importer des éléments d'un modèle compris dans d'autres packages.

En langage UML, un package est représenté graphiquement par une **icône de dossier**. Les packages sont également dotés d'un niveau de **visibilité**, et peuvent avoir des **relations**

entre eux. Ces relations sont de trois types: la **dépendance**, le **raffinement**, et la **généralisation**. Un package dépend d'un autre package s'il importe des éléments de celui-ci. Il en est un raffinement s'il décrit la même partie du modèle mais à un niveau différent d'abstraction. La généralisation prend la même signification que pour les classes. Les liens et relations entre les packages sont illustrés à la figure 1.8 par une **flèche pointillée** (allant dans le sens de la dépendance, de la généralisation ou du raffinement. Les classes d'un package ne sont pas accessibles à l'extérieur du package inconditionnellement.

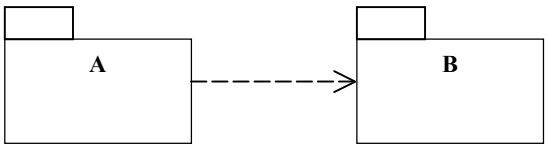


FIGURE 1.8 Package A importe les services du package B

1.5.3.5 Templates

Les classes paramétriques ou **templates** sont des modèles de classes. Comme certains langages supportent les **templates**, UML inclut ce type d'éléments dans ses capacités de modéliser les systèmes. Dans le cas des **templates**, il faut être en mesure de modéliser un **template** comme tel et aussi, de modéliser une instanciation particulière de ce dernier. La figure 1.9 montre ces deux types de modélisation. A gauche de la figure, on remarque le symbole pour représenter un **template** (ayant pour nom **Vecteur**) qui se compose d'un **rectangle** (comme pour une classe) assorti d'un **petit rectangle** contenant les paramètres de type du template (ici, le paramètre **T**). La classe **VecteurCreux** est un raffinement d'une instanciation du template **Vecteur.** Ce raffinement est montré grâce à une **flèche pointillée** avec le mode d'instanciation du template. A droite de la figure, on montre une instanciation simple du template **Vecteur**.

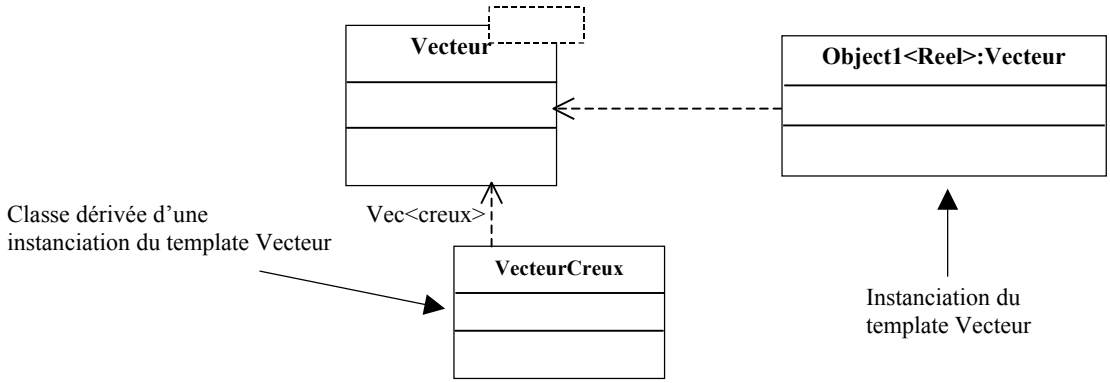


FIGURE 1.9 Template et instanciation d'un template

1.5.3.6 Classe utility ou utilitaire

Les classes **utility** ou **utilitaires** ne sont pas des classes dans le sens de l'approche orientée objets. Elles servent plutôt à représenter des modules regroupant des fonctions semblables. Par exemple, il est possible de regrouper dans un module les fonctions nécessaires pour réaliser l'inversion des matrices creuses. On ne peut pas créer un objet à partir d'une classe utilitaire.

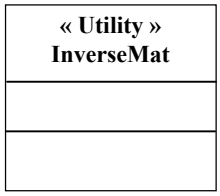


FIGURE 1.10 Représentation d'une classe utilitaire

1.5.4 Visualisation dynamique

Les systèmes ne sont généralement pas statiques mais évoluent dans le temps en réponse à des commandes provenant de l'extérieur ou encore en fonction de leur état interne à un instant donné. Si la visualisation statique permet de modéliser les principaux éléments formant le système, la **visualisation dynamique** a pour but de modéliser l'évolution de chaque élément en fonction des commandes reçues de son environnement (les messages qu'ils reçoivent) et les **interactions** entre ces éléments tant dans le temps que dans l'espace[15]. De manière plus spécifique, le modèle dynamique d'un système sert à montrer comment les objets d'un système **collaborent** pour réaliser les **points fonctionnels**. Il montre comment les objets échangent des **messages**. Ces échanges sont appelés **interactions** et sont représentées en UML via des diagrammes.

1.5.4.1 Objets en UML

En UML, les objets sont dessinés comme des classes (i.e. rectangle) mais le **nom** de l'objet est **souligné**. Les liens entre les objets sont illustrés par des **lignes** équivalentes aux lignes représentant les associations dans les diagrammes de classes mais sans l'information de multiplicité). Un **nom de message** accompagné d'un **numéro** est souvent associé à un lien pour montrer le sens et l'ordonnancement de la transmission du message.

Lorsque le nom de l'objet n'est pas encore déterminé, le nom des types est utilisé à sa place. Les types sont désignés par le symbole **:** placé devant le nom. Afin de clarifier le rôle

des objets, un rectangle avec le coin supérieur droit plié sert de commentaire (à travers une **linge pointillée**).

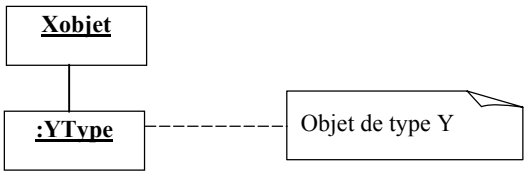


FIGURE 1.11 Représentation d'un objet

La figure 1.12 donne la description complète d'un message. La direction du message est indiquée par une ligne fléchée. La direction des données est indiquée par une ligne fléchée débutant par un cercle.

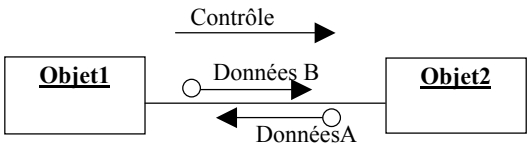


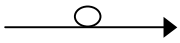
FIGURE 1.12 Description d'un message entre deux objets

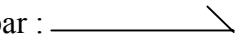
On peut distinguer plusieurs types de messages :

- constructeur : sert à créer les objets ;
- destructeur : déclenche la destruction des objets ;
- sélecteur : sert à obtenir des informations des objets ;
- modificateur : utilisé pour modifier les attributs des objets ;
- itérateur : sert à obtenir un objet parmi un ensemble d'objets.

Il existe également plusieurs types de messages de synchronisation.

- **Diffusion simple** : utilisé dans le cas où il n'y a qu'un seul objet actif à la fois. Représenté par : ⟶
- **Diffusion synchrone** : un message synchrone déclenche une opération de l'objet récepteur seulement si ce dernier est prêt à recevoir le message. L'objet expéditeur est bloqué jusqu'à ce que le récepteur accepte le message. Représenté par : ⟶|⟶
- **Rendez vous** : l'objet récepteur du message doit d'abord se placer en mode bloqué en attente d'un message, c'es l'inverse de la diffusion synchrone. Représenté par : ⟵⌐

— **Message temporisé** : ce message met l'objet expéditeur en état bloqué pendant une période de temps déterminée afin de permettre à l'objet récepteur de lui signifier la réception du message. Après la période d'attente, l'objet expéditeur est libéré d'une manière inconditionnelle. Représenté par : 

— **Message asynchrone** : la transmission d'un message asynchrone ne bloque pas l'objet expéditeur ou l'objet récepteur. Représenté par : 

1.5.4.2 Diagramme d'états

En UML, le **diagramme d'états** sert à décrire le **cycle de vie** d'objets, de systèmes, ou de sous-systèmes. Il montre explicitement les états que peuvent prendre les objets et comment les **événements** extérieurs (messages reçus, temps écoulé, occurrence d'erreurs, conditions logiques vraies) peuvent provoquer un changement d'état. Un tel type de diagramme doit être associé à toute classe dont les objets peuvent prendre des états facilement identifiables. Il décrit le **comportement** des objets et montre la manière dont ceux-ci évoluent en fonction de l'état présent.

Un diagramme d'états ne peut avoir qu'un seul **point de départ** mais peut par contre avoir plusieurs **points d'arrivée**. Il est important de noter que le diagramme d'états associé à une classe peut être construit indépendamment de la connaissance des autres classes car il ne s'intéresse qu'à la façon dont les objets changent d'état et non à l'objet spécifique responsable de l'événement qui provoque ce changement.

La figure 1.13 montre les principaux symboles rencontrés dans les diagrammes d'états. Un **point de départ** est symbolisé par un disque. Un **point d'arrivée** est symbolisé par un **cercle circonscrivant un disque**. Un **état** est symbolisé par un **rectangle aux coins arrondis** dans lequel le nom de l'état est inscrit. La **transition** entre deux états est représentée par une **flèche**. Le rectangle représentant un état est divisé en trois compartiments comme pour une classe.

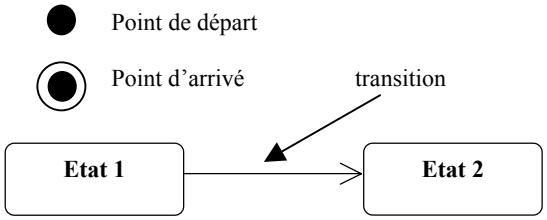


FIGURE 1.13 Les principaux symboles servant à construire les diagrammes d'états

1.5.4.3 Diagramme d'activités

Si le diagramme d'états s'intéresse à la façon dont un objet se comporte lorsqu'il reçoit des messages, le diagramme d'activité sert pour sa part à décrire le travail effectué dans les opérations d'une classe. En ce sens, le diagramme d'activités en UML se rapproche conceptuellement aux organigrammes introduits en génie logiciel il y a plusieurs années. Evidemment, UML offre une grande flexibilité de description des activités dans une opération et permet de visualiser les algorithmes de manière efficace.

1.5.4.4 Diagrammes séquentiels

Si le diagramme d'états sert à montrer l'évolution dynamique d'un objet pris séparément, le diagramme séquentiel s'intéresse à la manière dont plusieurs objets **interagissent** entre eux. Le diagramme séquentiel se concentre sur la **séquence** des messages envoyés entre les objets (c'est-à-dire quand et comment les messages sont envoyés et reçus par les objets participant à un point fonctionnel).

Les diagrammes séquentiels possèdent deux axes: **l'axe vertical** représente le **temps** et **l'axe horizontal** montre l'ensemble des **objets** en interaction dans un scénario, une scène spécifique, ou un point fonctionnel complet. L'axe du temps illustre le moment où, dans un scénario, un objet transmet un message à un autre objet. La transmission de ce message s'illustre entre les lignes de vie des deux objets.

Le diagramme séquentiel peut être utilisé sous deux formes: la forme générique et la forme d'instanciation. La forme générique décrit tous les déroulements possibles d'un scénario et peut contenir des branchements, des conditions, et des boucles. La forme d'instanciation décrit le comportement du système pour un aspect spécifique d'un scénario et ne contient pas de branchements, boucles, ou conditions. Dans les diagrammes séquentiels, les messages représentent une communication entre les objets et véhiculent une information avec l'anticipation qu'une action sera entreprise. Les mots **message** et **action** doivent être pris dans le même sens que pour les diagrammes d'états. L'activation d'un objet est représentée par **un rectangle étroit** sur sa ligne de vie.

La figure 1.14 montre un diagramme de classes représentant les classes en présence lors de l'ouverture d'un transformateur. Les différentes classes en présence sont un **Acteur** (l'usager travaillant dans un poste), un **Poste** doté de l'opération **Ouvrire()**, d'un système de

protection (objet **SystProtection**) dont la tâche est de commander les actions du disjoncteur, et un **Disjoncteur** responsable de l'ouverture du transformateur (objet de la classe **Transformateur)**. La figure 1.15 montre le diagramme séquentiel réalisant le point fonctionnel associé à l'ouverture d'un transformateur à travers l'ouverture de son disjoncteur.

L'usager **Acteur** demande l'ouverture d'un transformateur en envoyant le message **Ouvre(Transformateur)** au poste (objet **Poste**). Ce dernier envoie le message **Ouvre(Transformateur)** au disjoncteur (objet **Disjoncteur**) si la condition de franchissement des seuils est réalisée.

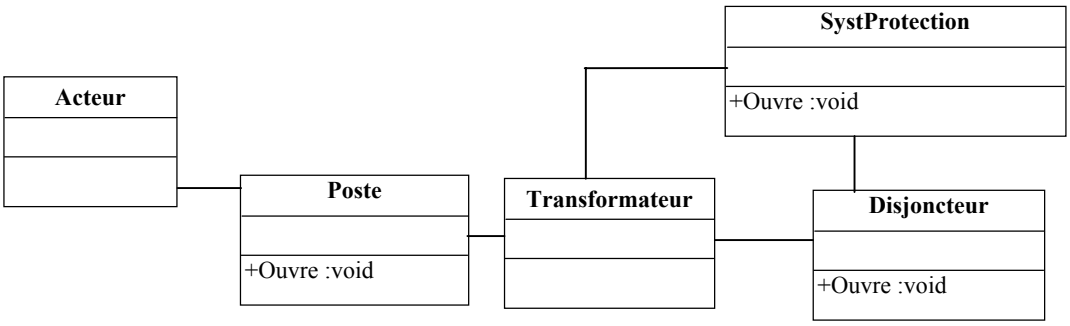


FIGURE 1.14 Diagramme de classes pour illustrer les diagrammes séquentiels.
Point fonctionnel: ouverture d'un Transformateur

Le diagramme séquentiel dépeint donc clairement la séquence de transmission des messages entre les différents objets impliqués dans une interaction lors de la réalisation d'un point fonctionnel ou d'un sous ensemble d'un point fonctionnel. Une telle modélisation peut sembler fastidieuse, spécialement si le système comporte de nombreux points fonctionnels. Il faut comprendre qu'il est beaucoup plus facile de comprendre les interactions via un tel type de diagramme que via une lecture des commentaires dans le code ou d'une documentation générale sur le logiciel.

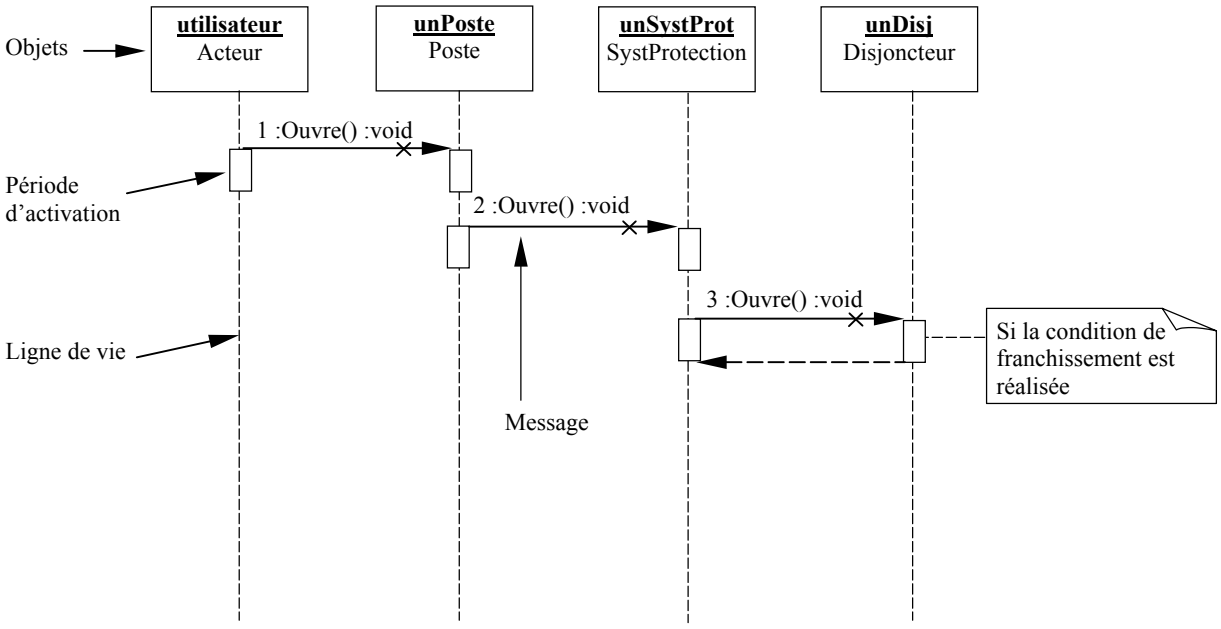


FIGURE 1.15 Diagramme séquentiel montrant une interaction entre les objets de la figure 1.14

1.5.4.5 Diagramme de collaboration

Comme le diagramme séquentiel, le diagramme de collaboration s'intéresse à la modélisation dynamique de l'interaction entre les objets participant à la réalisation d'un point fonctionnel. Cependant, le diagramme de collaboration vise à représenter l'aspect "spatial" de cette interaction, c'est-à-dire à monter les objets en présence, à illustrer les liens entre ces objets, et à décrire comment ils interagissent, le temps étant ici une variable secondaire. On peut généralement dire que les diagrammes de collaboration sont plus utiles que les diagrammes séquentiels pour décrire de manière plus complète la réalisation de points fonctionnels. Si on reprend le diagramme de classes de la figure 1.14 et son diagramme séquentiel de la figure 1.15, nous pouvons construire le diagramme de collaboration de la figure 1.16.

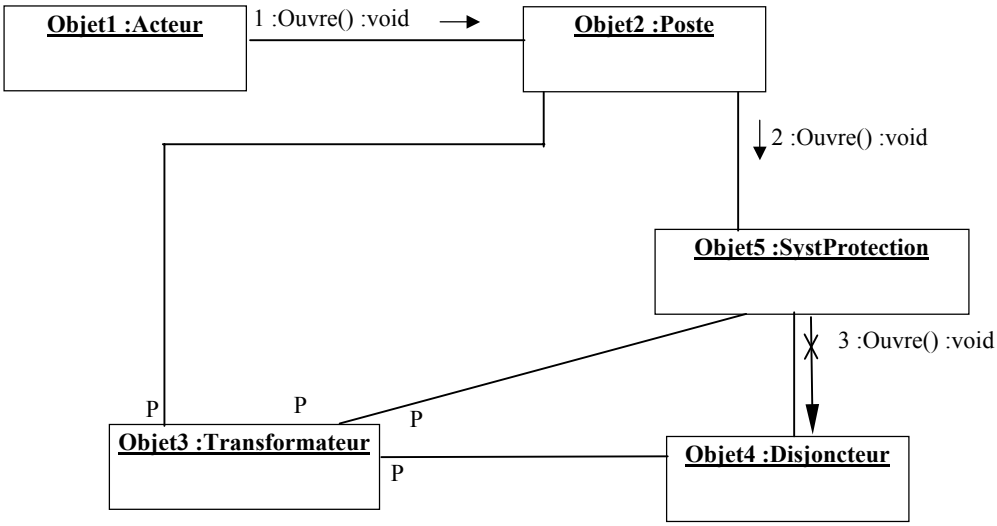


FIGURE 1.16 Diagramme de collaboration pour le diagramme de classes de la figure 1.14

On remarque que, contrairement au diagramme séquentiel, le diagramme de collaboration inclut un objet de la classe **Transformateur**. En effet, dans l'interaction des objets de ce point fonctionnel, l'objet de la classe **Transformateur** ne reçoit aucun message et il n'est donc pas pertinent de le montrer dans le diagramme séquentiel. Par contre, l'objet de la classe **Transformateur** intervient dans la collaboration soit en tant qu'objet global que comme paramètre dans des messages. Le diagramme de collaboration rend explicite la présence de l'objet de la classe **Transformateur** et montre comment il intervient dans la collaboration. On remarque également les étiquettes avec leur numéro accompagnant les liens entre les objets d'une collaboration. Il convient de remarquer que la classe **Transformateur** possède des liens avec les autres classes dans le diagramme de classes, ce qui permet d'inclure les mêmes liens entre les objets du diagramme de collaboration pour le point fonctionnel illustrant l'impression d'un fichier.

1.6 Patrons de conception (design patterns)

L'expérience acquise au fil de nombreux projets a permis aux ingénieurs en logiciel de constater que des structures de classes ou que des schémas de collaboration entre objets se répétaient souvent dans des conceptions(designs) différents. C'est de cette expérience et de ces observations que sont nés les **design patterns**, qui sont en fait des pièces détachées fonctionnelles d'architecture de logiciel qui peuvent être mises à profit dans de nombreuses

conceptions, à la manière de blocs de construction, pour rendre ceux-ci plus robustes, plus clairs, et plus réutilisables. Il faut préciser que les design patterns ne sont pas des composantes logicielles réutilisables mais bien des structures de hiérarchies ou de collaborations entre objets qui peuvent contribuer à édifier des hiérarchies et des collaborations.

En génie Electrique par exemple, l'alimentation des appareils à courant continu passe par le redressement du courant alternatif. Le circuit réalisant cette tâche est un circuit standard. Il est utilisé sans avoir à analyser au préalable son comportement parce qu'il a été appliqué avec succès dans plusieurs systèmes. Ce circuit ou schéma est un patron de conception, il porte un nom et son champ d'application est bien défini. Les patrons de conception ne sont pas propres au génie électrique, ils existent dans bien des disciplines et sont appliqués avec succès depuis bien longtemps. Donc un patron de conception est caractérisé par son nom et son contexte d'application.

Au fil des ans, des concepteurs et chercheurs ont établi un catalogue de **design patterns** qui est mis à la disposition des autres concepteurs pour qu'ils les utilisent dans leurs propres designs [16,17]. Les patterns appartiennent à trois principales catégories (contexte d'application): les patterns de création, les patterns structurels et les patterns comportementaux .

1.6.1 Patterns de création

Les patterns de création tentent de rendre plus abstrait le processus d'instanciation des objets. Ils aident à rendre un système indépendant de la façon dont les objets qu'il manipule sont créés, composés, et représentés. On trouve deux types de patterns de création :

- les patterns de création de classe : ils utilisent l'héritage pour changer la classe qui est instanciée ;
- les patterns de création d'objets : ils délèguent la création d'objets à un autre objet.

1.6.2 Patterns structurels

Les patterns structurels s'intéressent à la façon dont les classes et les objets peuvent être combinés pour former des structures plus complexes (ou simplement plus grandes). Plutôt que de modifier les interfaces de classes existantes pour augmenter leur complexité et leur niveau de fonctionnalité, les patterns structurels d'objets décrivent des façons de

composer des objets pour atteindre cette nouvelle fonctionnalité. La flexibilité accrue offerte par la composition d'objets se justifie principalement par le fait qu'elle permet de modifier la fonctionnalité au run-time, ce qui n'est évidemment pas possible avec la composition statique de classes.

1.6.3 Patterns comportementaux

Les patterns comportementaux s'intéressent aux algorithmes et à la distribution des responsabilités entre les objets responsables de la solution d'un problème. Ils ne décrivent pas seulement des patterns entre objets mais aussi à la communication entre eux. Ces patterns représentent des mécanismes de communication complexes qu'il est difficile de suivre au run-time. Ils permettent d'éviter de porter trop d'attention au contrôle des interactions entre les objets et de concentrer les efforts sur la façon dont les objets sont interconnectés. Les patterns comportementaux de **classes** utilisent l'**héritage** pour répartir les comportements entre les classes.

Les patterns comportementaux d'**objets** utilisent la **composition** plutôt que l'héritage. Certains de ces patterns décrivent comment un groupe d'objets coopèrent pour accomplir une tâche qu'aucun objet isolé ne pourrait accomplir seul tout en demeurant à un niveau de complexité acceptable du point de vue de l'évolution et de l'entretien.

1.7 Conclusion

L'approche orientée objets s'est révélé applicable à une large variété de domaines d'applications. Elle constitue peut être la seule méthode pouvant de nos jours faire face à la complexité inhérente aux très gros systèmes. L'approche orientée objets présente de nombreux avantages, ainsi que quelques risques, l'expérience montre que les avantages l'emportent de beaucoup sur les risques.

Les avantages évoqués dans les paragraphes précédents s'avèrent très important pour le développement d'applications ou systèmes réseaux électriques. Il n'existe évidemment pas de recette miracle pour la conception de tels systèmes. Cependant les mécanismes de base intervenant au cœur même du système sont connus et spécifient clairement comment doivent se dérouler les choses. En revanche, certaines décisions dépendent du type d'application réseaux électriques elle même (analyse de répartition de charges, analyse dynamique ou analyse d'estimation d'états, etc.).

Le début de l'application de la TOO aux réseaux électriques remonte au début des années 1990. La majorité des travaux traitent l'application de l'écoulement de puissances [1,5,18]. On rencontre également l'application d'interface graphique utilisateur [8,9,10,11], la simulation dynamique [2,3] et la restauration des réseaux électriques [19,20]. Récemment, quelques travaux ont abordé le calcul générique des systèmes linéaires pour les réseaux électriques [7,21].

Depuis l'année 2000, les auteurs défendent l'idée de séparation entre la modélisation des éléments physiques du réseau électrique et la modélisation de ses applications. En résumé la majorité des travaux actuels optent pour le développement de trois architectures : modélisation des réseaux électriques (éléments physiques), modélisations des applications (fonctions de calculs) et modélisation des facilitées mathématiques (moyens de calcul).

Ainsi, dans ce travail de thèse on présente une proposition dans cette direction.

Chapitre 2

Modélisation Orientée Objets des Réseaux Electriques

2.1. Introduction

Dans le domaine de réseaux électriques, on constate que l'application de la MOO à beaucoup d'avantages vue que la structure physique d'un réseau électrique est adaptable à une structure de classes [2,4,18]. La MOO d'un réseau électrique ne vise pas seulement la construction d'une bonne structure de classes de ses éléments, mais également les méthodes d'analyse de ce système et les GUI. La construction des GUI est devenue une partie importante du génie logiciel [8,9,10,11]. En effet les phénomènes physiques ainsi que les données d'un réseau électrique sont mieux assimilées si l'information est représentée sous forme graphique contrairement à la forme numérique. Pour des systèmes complexes, où les interactions homme-machine sont nombreuses, la spécification de la GUI par une approche orientée objets offre des avantages indéniables en terme de génie logiciel.

Dans ce chapitre, on essaye de combiner la MOO, la simulation des réseaux électriques et la GUI pour dégager la structure générale de l'outil informatique développé. Mais avant tout, nous passons en revue les principales caractéristiques et propriétés de tout système logiciel. Et enfin on présente les grandes structures de classes adaptées pour le développement de cet outil.

2.2 Développement des systèmes logiciels

Le développement de systèmes logiciels étant une industrie relativement neuve, il n'a pas encore atteint le niveau de maturité des branches plus traditionnelles de l'industrie. Par conséquent, les produits que l'on développe sur la base de la technologie du logiciel

souffre du manque de pratiques établies nécessaires à leur développement et à leur exploitation en tant que produits commerciaux [14].

2.2.1 Cycle de vie d'un système logiciel

Tous les systèmes évoluent au cours de leur cycle de vie. On doit garder cela à l'esprit lorsque l'on développe des systèmes qui sont censés évoluer au delà de la première version. Normalement, on développe un système au fil des modifications, de version en version. La première version représente une partie mineure de la consommation de ressources totale qui aura lieu durant le cycle de vie d'un système. A chaque nouvelle version, on effectue les mêmes activités de développement que pour le développement d'un nouveau système. La différence vient du fait que les données d'entrée décrivent des besoins de modification.

Historiquement, dans la plupart des projets à connotation informatique, on spécifie les besoins du système comme un tout. Après les spécifications viennent l'analyse, la conception et le test du système complet. Cette méthode peut fonctionner si on connaît tous les besoins du système final dès le début, mais c'est rarement le cas. D'habitude, on ne connaît pas complètement les besoins au démarrage du projet pour les systèmes techniques aussi bien que pour les systèmes d'information. On accroît sa connaissance du système au fur et à mesure que le travail progresse. Lorsque la première version du système est opérationnelle, apparaissent de nouveaux besoins, et les anciens changent. On ne peut donc pas développer le système dans son ensemble en croyant que la spécification des besoins restera la même durant le temps de développement, qui peut atteindre plusieurs années pour les gros systèmes.

Dans la plupart des cas, il vaut mieux développer le système étape par étape en commençant par quelques unes de ses fonctions clés. On peut ajouter de nouvelles fonctions ensuite jusqu'à ce que l'on ait atteint le niveau désiré.

2.2.2 Réutilisation et composants

Un désir commun à tous les travaux de développement est de pouvoir réutiliser les résultats des travaux précédents. La nécessité de réutilisabilité est applicable durant le codage, puisqu'elle peut avoir une influence significative sur la productivité. C'est dans ce contexte que les gens du logiciel parlent de réutilisabilité. La réutilisation au niveau des

autres phases du développement peut améliorer encore plus la productivité. Le problème de réutilisation est de trouver et de comprendre ce que l'on doit réutiliser, et de juger de la pertinence de la réutilisation. L'orientée objets offre une technique totalement nouvelle qui fournit une aide efficace pour résoudre ces problèmes. Etre capable de réutiliser des parties déjà développées (composants) dans un produit constitue un facteur significatif de diminution du coût du cycle de vie du produit.

Les composants logiciels sont traditionnellement disponibles sous la forme de procédures et de fonctions pour les applications numériques et statiques. Il faut y rajouter les composants logiciels permettant entre autre de gérer des tampons de mémoire, des files d'attente, des listes et des arbres, dont on a souvent besoin lorsque l'on programme des algorithmes. On trouvera aussi des fenêtres, des icônes et des barres de défilement pour les interfaces homme/machine.

2.2.3 Stratégie de développement

L'une des propriétés clés d'un système logiciel est sa structure interne. Une bonne structure rend le système facile à comprendre, modifier, tester et maintenir. C'est pourquoi les propriétés de l'architecture d'un système déterminent la manière dont on le traitera au cours de son cycle de vie. Les plus gros systèmes seront sujets à modification au cours de leur cycle de vie. Une approche industrielle du génie logiciel doit prendre ce fait en compte. En fait, le développement de systèmes est un processus de modifications successives qui dure aussi longtemps qu l'on impose de nouveaux besoins au produit. Le processus de développement logiciel est un ensemble d'activités reliées à la création, réalisation et maintenance des systèmes logiciels. Il n'existe pas de méthode universelle applicable à tous les processus de développement.

En pratique, l'approche orientée objets s'inscrit dans un processus de développement itératif. Le système logiciel n'est pas construit étape par étape mais bien itération en itération. Un sous ensemble d'exigences est analysé, des solutions de conception sont proposées, la conception et la validation des solutions sont réalisées dans chacune des itérations. De cette façon le système logiciel est construit par incrément et sa complexité devient gérable. De plus, à la fin de chaque itération, il est possible d'apporter des modifications et corriger les erreurs d'analyse et de conception.

2.3 Développement de logiciels de réseaux électriques

La nécessité de modéliser les réseaux pour en simuler le fonctionnement remonte sans doute aux origines des réseaux électriques eux-mêmes. Les premiers simulateurs étaient des simulateurs analogiques où des modèles réduits de réseaux permettaient de mieux prévoir ou de connaître le comportement du système. Ils permettaient de couvrir certains besoins comme la mise au point ou le test d'équipements de régulation et de protection des ouvrages. De nombreux simulateurs analogiques de cette génération, fonctionnant en temps réel sont toujours utilisés. Aujourd'hui, il est de plus en plus question de simulateurs numériques.

2.3.1 Complexité des logiciels de réseaux électriques

Les problèmes que les logiciels de réseaux électriques doivent résoudre comportent souvent des éléments extrêmement complexes et qui cachent de multitudes d'exigences. Aujourd'hui, les réseaux d'énergie électrique sont de plus en plus complexes alimentant des charges elles même de plus en plus exigeantes. En parallèle, la maîtrise des régimes de fonctionnement perturbés et la conception de protections sûres et sélectives contribuent à augmenter cette complexité. Donc, le fonctionnement des réseaux électriques est déjà difficile à cerner, et pourtant il faut y ajouter des exigences (non fonctionnelles) telles que la facilité d'utilisation et la maintenance de ses logiciels. Cette complexité du problème lui même entraîne donc une complexité du logiciel.

La complexité du logiciel provient généralement de la façon dont les utilisateurs et les développeurs voient les choses. Les gens de réseaux électriques trouvent des difficultés à fournir une expression précise de leurs besoins sous une forme que les développeurs (informaticiens) peuvent comprendre. Cette incompréhension n'est due ni aux utilisateurs ni aux développeurs, mais plutôt au fait que chacun de ces groupes manque d'expertise dans le domaine de l'autre. En plus, cette complexité est augmentée par le fait que les spécifications d'un logiciel changent souvent en cours de développement. C'est pourquoi, d'après la littérature, une grande majorité de logiciels de réseaux électriques sont développées par les gens de réseaux électriques eux mêmes.

2.3.2 Acheter ou développer les logiciels à utiliser

La grande question souvent posée dernièrement se résume à ceci : vaut-il mieux investir et acheter l'outil de simulation ou développer son propre outil ? Les deux options sont valables, et le choix se fait généralement selon les contraintes de temps, des ressources et le but visé.

L'achat de l'outil entraîne souvent des coûts imprévus. De plus, vu que la désuétude des ordinateurs est de plus en plus rapide, le choix de dépréciation devrait être de cinq ans sans quoi la technologie risque d'être périmé. Une entreprise peut se permettre de gros investissement dans cette direction contrairement à un établissement académique où développer convient le mieux.

2.4 Structure générale de l'outil développé

Afin de tirer parti des avantages des TOO, une modélisation orientée objets selon OMT à été développée pour la conception des composants logiciels intervenant dans le processus de développement. La stratégie retenue quant à l'architecture des simulateurs de réseaux électriques a conduit à quatre grandes parties. La figure 2.1 montre ces principales parties qui sont :

- Un éditeur graphique est spécialement développé pour visualiser les diagrammes unifilaires des réseaux électriques avec fenêtres de boites de dialogue. Il utilise des symboles graphiques pour représenter les éléments du réseau électrique tels que les jeux de barres, les lignes de transmission, les charges, les générateurs, etc.
- Une base de données visuelle est développée pour que l'utilisateur puisse faire entrer et modifier les données avec souplesse sur écran. Les données sont liées au diagramme unifilaire et aux applications à exécuter.
- Les applications qui simulent le fonctionnement d'un réseau électrique, les applications réalisées actuellement dans cet outil sont l'écoulement de puissances et le calcul des courants de court circuit.
- Les facilitées de calcul telles que les opérations sur des matrices et l'utilisation des matrices creuses.

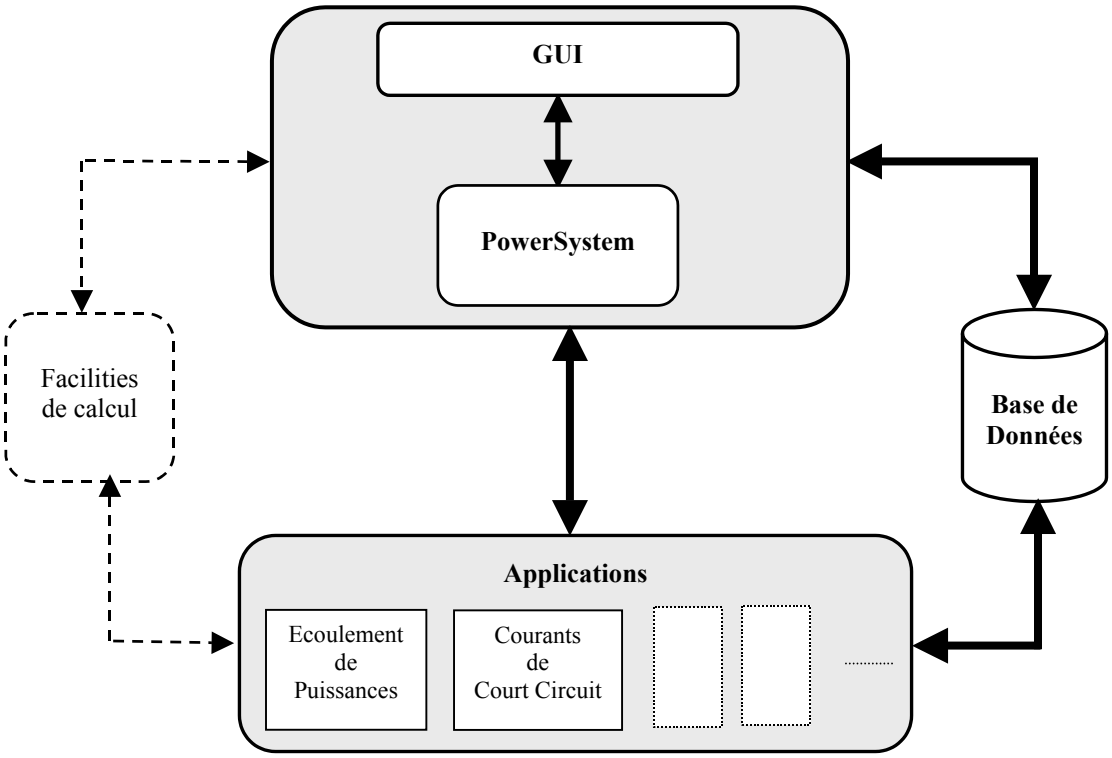


FIGURE 2.1 Structure générale

Toutes les parties sont développées en utilisant Borland C++ Builder version 5.0[25] et le langage de programmation C++ [26]. Le matériel utilisé est un PC Intel II. Pour implémenter les fonctionnalités de la GUI, l'outil développé utilise deux hiérarchies d'objets, l'une dérivant de l'objet TForm de Builder C++ pour représenter les fenêtres elles mêmes et l'autre dérive de l'objet TGraphicControl de C++ Builder pour représenter les éléments graphiques sur ces fenêtres.

2.5 Interface graphique usager GUI

La construction des interfaces usagers graphiques est devenue une partie importante du génie logiciel. Au cours des dernières années, nous avons vu apparaître de nombreuses méthodes et outils permettant de réduire la charge de travail des développeurs et des concepteurs et permettant de produire des interfaces de meilleure qualité. Depuis leur apparition, les interfaces graphiques n'ont pas cessé d'évoluer en apportant plus de souplesse à l'utilisateur, en réduisant sa charge de travail et en répondant efficacement à ses besoins. L'acceptation de ces nouvelles interfaces n'est pas uniquement due à

l'avancée technologique de l'informatique que ce soit au niveau matériel ou logiciel, mais elle revient aussi aux résultats des sciences cognitives qui s'intéressent à l'étude du comportement humain. Par l'intégration des règles ergonomiques (facilité d'utilisation, concision, cohérence, flexibilité, etc.), ces interfaces ont mérité l'appréciation de l'utilisateur. Une autre raison principale de ce succès provient du fait que l'utilisateur est maître de l'interaction avec l'application pendant toute la session de travail [27, 28].

L'avancée technologique au niveau hardware et software a fait bondir les applications interactives au premier plan. Actuellement le développement de la GUI prend une grande place dans le développement des applications informatiques. Certaines évaluations [22] ont révélé que plus de 50% du temps de développement est consacré à la réalisation de la GUI, et que plus de 70% du coût d'un produit logiciel s'accumule dans sa maintenance où la maintenance de la GUI représente la grande partie.

Pour des systèmes industriels complexes, où les interactions homme-machine sont nombreuses, la spécification de la GUI par une approche orientée objets offre des avantages indéniables en terme de génie logiciel. En effet, cette technique repose sur les notions de classe et d'héritage. L'utilisation et la combinaison d'un ensemble de techniques facilitent la maîtrise de la complexité, l'extensibilité et la réutilisabilité des objets pour la construction des GUI, et assurent la cohérence et l'homogénéité dans les modes de présentation et les dialogues. Le marché informatique actuel offre un éventail varié d'environnements graphiques facilitant la réalisation et la modification des GUI en réduisant l'écriture de code et en permettant la réutilisabilité des programmes.

Les phénomènes physiques ainsi que les données d'un réseau électrique sont mieux assimilées si l'information est représentée sous forme graphique contrairement à la forme numérique. Plusieurs outils graphiques dans le domaine de réseaux électriques ont été développés pour l'enseignement, la recherche et l'entraînement [8,9,10,11,23]. Dans ce travail de thèse, on essaye de combiner la MOO, la simulation des réseaux électriques et la GUI dans un seul outil [29]. L'interaction principale entre l'utilisateur et la simulation est faite à travers la GUI. Pour que l'outil développé soit une plate forme pour l'étude des réseaux électriques, il est important que la GUI puisse fournir à l'utilisateur des accès faciles à toutes les informations concernant le réseau électrique, même pour de grands

systèmes. Pour accomplir ceci, la GUI est conçue en utilisant la TOO. La partie la plus importante de la GUI concerne la représentation des diagrammes unifilaires ou l'éditeur graphique. La TOO permet à l'utilisateur d'interagir avec tous les objets dans les différentes fenêtres. Les dernières années, le monde du développement de logiciels s'est trouvé profondément modifié par le succès des outils de développement rapide d'applications (RAD) tels que C++ Builder, Delphi, Java et autres et l'arrivée de la TOO.

2.6 Modélisation orientée objets des réseaux électriques

La phase de modélisation est une étape très difficile, c'est celle qui fait le plus appel au « sens de l'ingénieur », elle suppose que l'on ait préalablement fourni une réponse claire à ce qu'on veut représenter et avec quel objectif. Dans le domaine de réseaux électriques, on constate que l'application de la MOO a beaucoup d'avantages vue que la structure physique d'un réseau électrique est adaptable à une structure de classes [2,4,18].

La MOO d'un réseau électrique ne vise pas seulement la construction d'une bonne structure de classes de ses éléments, mais également des méthodes d'analyse (applications ou fonctions de calcul) de ce système électrique et son GUI. Il serait bien normal de penser que les méthodes d'analyse devraient être implémentées comme méthodes (fonctions membres) dans la structure représentative du système. Cependant, cette vision limite la flexibilité de la structure. Actuellement la MOO fait sortir deux grandes structures de classes : la structure des éléments physiques du système électrique et les méthodes d'analyse appliquées au système ou sur ses éléments. Une troisième structure utilitaire concerne les facilitées de calcul (matrices, algorithmes, etc.).

La structure des éléments physiques est représentée en premier lieu, ensuite la structure des méthodes d'analyse est formée indépendamment de la première structure représentative du réseau électrique. La deuxième structure agit sur la première en l'utilisant comme données d'entrée et de sortie pour ses méthodes. La représentation des méthodes d'analyse par une telle structure indépendante de la structure physique apporte de la flexibilité à l'outil informatique. Dans la construction de la structure des éléments physiques on ignore les détails caractérisant les méthodes d'analyse du réseau électrique, et vice versa. Ce qui permet d'avoir un projet plus modulaire.

La définition des structures de classes est le point de départ dans un projet orienté objets. Le modèle objet statique qui représente les éléments du monde réel et ses relations est la base des prochaines étapes du projet [4,29,30]. Par conséquent, une attention particulière doit être adressée à cette phase du projet qui n'est pas une tâche facile [17]. En effet, dans des problèmes complexes, le cas des réseaux électriques, la possibilité d'avoir plusieurs représentations de classes est largement évidente. Donc le choix d'une forme ou d'une autre pour la représentation de la structure, selon la philosophie de la MOO, peut impliquer des dissimilitudes dans l'exécution des applications bien déterminées.

D'une manière générale, la conception d'un bon logiciel orienté objets n'est pas une tâche facile. Un bon logiciel doit également avoir de bonnes performances sur le plan temps de calcul. Les travaux les plus récents [1,2,3,18] ont donné des résultats satisfaisant pour des réalisations orientées objets (en utilisant le langage de programmation C++). Ces améliorations dans le temps de calcul sont dues aux développement de compilateurs qui supportent l'orientée objets et qui ont évolué suffisamment ces dernières années. D'un autre coté une grande évolution sur le plan matériel a permis de disposer de machines de grande capacité de traitement de données.

Dans ce travail, le processus de création des structures de classes représentatives des entités diverses du réseau électrique dans son ensemble sont divisées en abstractions distinctes et représentées par des packages. La figure 2.3 montre ces abstractions et leurs dépendances. On considère trois abstractions principales : le réseau électrique (classe PowerSystem), les fonctions de calcul (classes des Applications) et les facilitées de calcul (classe ComputationalFacilities). Cette dernière n'est pas décrite dans ce travail parce qu'elle n'est pas vraiment implémentée orientée objets, mais elle est utilisée dans les différents calcul.

En principe, l'outil informatique est constitué par un objet réseau électrique et une ou plusieurs applications peuvent être exécutées sur cet objet. Les abstractions ne sont pas fermées, elles admettent l'inclusion de nouvelles entités dans leurs propres limites.

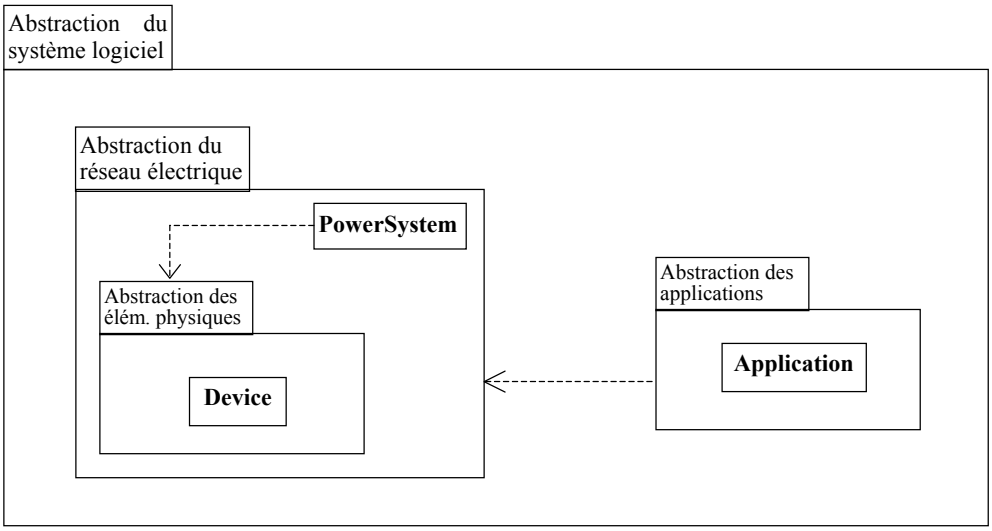


FIGURE 2.2 Abstractions du système logiciel

L'abstraction du système électrique délimite la modélisation des éléments physiques du réseau électrique. Cette abstraction est illustrée par la figure 2.3.

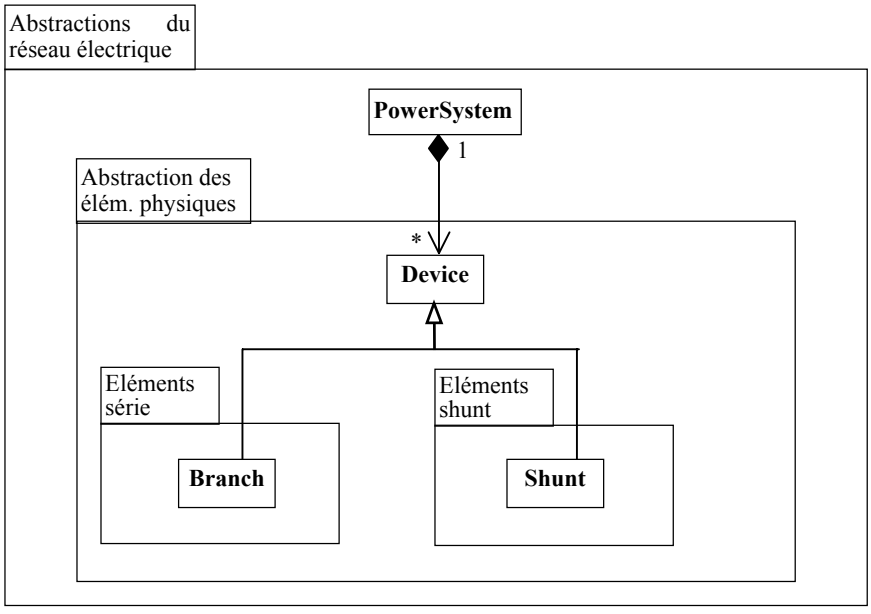


FIGURE 2.3 Abstraction du réseau électrique

Dans cette abstraction deux entités sont identifiées : la classe PowerSystem et la classe Device. La classe PowerSystem forme le réseau électrique dans son ensemble, elle englobe les classes qui forment ce réseau et qui représentent les éléments physiques. Cette abstraction contient les classes des jeux de barres, des lignes de transmission, des transformateurs, des charges, des générateurs etc., qui sont représentées dans la figure 2.4 par la classe de base Device. Cette classe est la classe de base de tous les éléments physiques du réseau électrique. Dans cette abstraction apparaît la classe constitué par la composition de ses éléments physiques divers.

2.6.1 Modélisation des éléments physiques

Un point clef dans la conception d'une structure qui représente le réseau électrique est le critère de classification hiérarchique des classes représentatives de ce système. On entend par élément physique tout dispositif qui relié d'une manière ou d'une autre au réseau électrique fait partie de sa constitution. Dans ce travail actuel de thèse, la classification des éléments physiques est basée sur la structure réelle du système électrique. Cette classification prend en compte dans sa limite inférieure le nombre de connections aux différents jeux de barres que chaque élément présente, identifiant là trois classes :

- Jeux de barres .
- Eléments en série (ou branche, avec deux connections).
- Eléments en dérivation (ou shunt, avec une connexion).

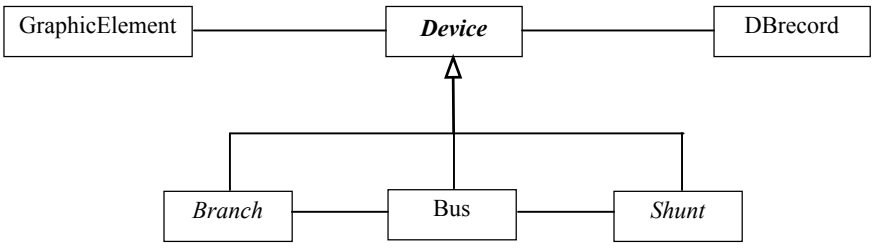


FIGURE 2.4 Eléments physiques

2.6.2 Modélisation des applications

L'abstraction des applications représente les méthodes d'analyse des réseaux électriques. Ces applications dérivent toutes de la classe PowerSystem. Les applications dépendent du système électrique, donc les algorithmes d'analyse seront exécutés sur la base des données représentant le système à étudier.

L'abstraction des applications définit les classes représentant les différentes méthodes d'analyse et de synthèse appliquées aux réseaux électriques. Cette abstraction est illustrée par la figure 2.5. Dans cette abstraction la classe PowerSystem sert de base dans la structure hiérarchique des classes représentant toutes les applications qui peuvent exister.

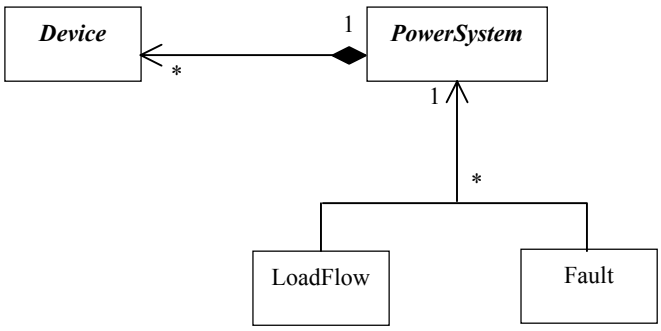


FIGURE 2.5 Eléments physiques, réseau électrique et applications

2.6.3 Modélisation des facilitées de calcul

Généralement toutes les applications réseaux électriques ont besoin de certaines structures de données bien précises tels que les complexes, les vecteurs, les matrices, les matrices creuses, la factorisation, les listes chaînées etc. Ainsi plusieurs auteurs [6,7,21] ont pensé à réaliser une sorte de bibliothèque mathématique spéciale pour les réseaux électriques qui englobe ces structures et en utilisant la TOO.

Ici également la question se pose : faut-il développer ses propres structures ou utiliser les bibliothèques standards qui existent sur marché ? Ces dernières ont une taille énorme et sont d'un usage général. Autrement dit laquelle de ces deux situations est la plus juste :

— Utiliser des « boites noires » avec une garantie totale de leur bon fonctionnement sans comprendre les détails internes des algorithmes utilisés.
— Développer les structures spéciales pour des applications particulières même si ces programmes ne sont pas assez performant que ceux sur marché.

Les deux situations sont justes. Mais récemment, dans le domaine des réseaux électriques, chacun à tendance à développer ses propres outils tant qu'il peut [1,2,6,7,21]. La raison est que les structures de données à usage général ont comme majeure conséquence un gaspillage de performances quant elles sont appliquées à un problème particulier.
Actuellement, l'utilisation des templates semble répondre à toutes les exigences vu qu'ils ont un aspect général et particulier en même temps.

Dans ce travail de thèse, cette partie a été entamée dans ce but décrit en dessus mais nous n'avons pas pu terminer son implémentation avec la TOO [30]. C'est pourquoi elle n'est pas présentée en tant qu'une abstraction du logiciel développé. Cependant, plusieurs parties sont utilisées aux nivaux des applications tels que les complexes, les matrices creuses, les listes chaînées, la factorisation et autres. Elles seront présentées au chapitre 4.

2.7 Conclusion

Dans ce chapitre, nous avons présenté les grandes abstractions pour la plate forme logicielle de l'outil développé. Une grande partie des fonctionnalités de ces abstractions ont été mises en œuvre dans une architecture orientée objets (mise à part la partie base de données). L'objet de cette étude que nous détaillons dans les chapitres suivants, vise à fournir des composants logiciels pour la représentation, la simulation et la manipulation de données des réseaux électriques.

Chapitre 3

Modélisation et Implémentation des éléments physiques

3.1 Introduction

Ce chapitre décrit la MOO et l'implémentation des différents éléments physiques considérées dans ce travail. La majorité des travaux apparus ces dernières années ont classifié les classes des éléments physiques du réseau électrique selon une hiérarchie tirée de sa structure réelle [1,4,29,30]. Cette classification est basée en premier lieu sur le nombre de jeux de barres de connexion que chaque élément possède. D'un autre coté, la MOO de chaque élément est basée sur son modèle algébrique.

Puisque la MOO des éléments physiques est basée sur le monde réel de ces éléments le modèle objet développé doit permettre :

- L'utilisation générale par différentes applications, et donc la possibilité d'intégrer des applications dans le futur basée sur le même modèle.
- La maintenance et la perfection des différentes parties en ajoutant ou en supprimant du code dans les différentes classes.
- La facilité de rajouter des éléments nouveaux dans le modèle.
- Le bon calcul des applications d'analyse , objectif principal à ne pas oublier.

3.2 Classe de base : Device

La classe abstraite Device de la figure 3.1 sert de base pour tous les éléments du système. On identifie dans la hiérarchie trois principales classes descendant directement de la classe Device :

- Classe Bus, représentant les jeux de barres.
- Classe Branch, représentant les éléments séries (deux connections).
- Classe Shunt, représentant les éléments en dérivation (une connexion).

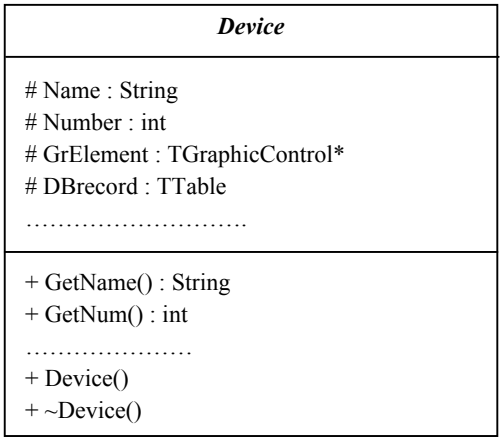


FIGURE 3.1 Classe Device

Les principaux attributs et méthodes de cette classe sont :

- Name : identification du nom de l'élément ;
- Number : identification du numéro de l'élément ;
- GraphicElement : élément graphique correspondant ;
- DBrecord : liaison avec la base de données ;
- Device() : constructeur ;
- ~Device() : destructeur.

Chaque élément possède une identification du type string (attribut Name), qui identifie le nom de l'élément. Cette identification est toujours attribué à la classe propre de l'élément au moment de sa création dans la méthode de construction (Constructeur de la classe). Chaque élément reçoit également toujours un numéro d'identification stocké dans Number.

L'implémentation de ces éléments dans la GUI est faite par héritage à partir de la classe graphique GraphicControl de C++ Builder. Ces éléments sont représentés par des boutons dans la barre d'outils (figure 3.2). La création de n'importe quel élément se fait en cliquant sur le bouton correspondant.

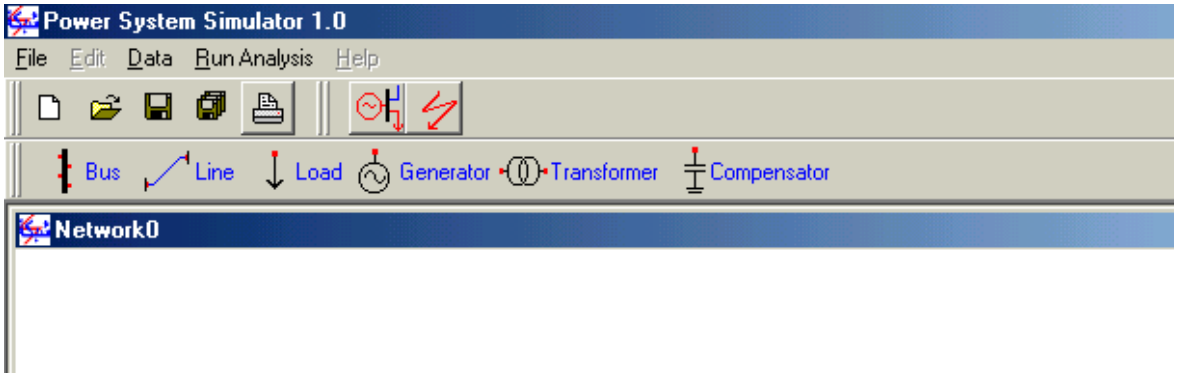


FIGURE 3.2 Barre d'outils des éléments physiques

3.3 Elément jeu de barres :classe Bus

La classe Bus représente un jeu de barres d'un réseau électrique. Ces objets représentent les points de raccordement de la structure d'un réseau électrique. La figure 3.3 montre cette classe en détails. Elle dérive directement de la classe abstraite Device.

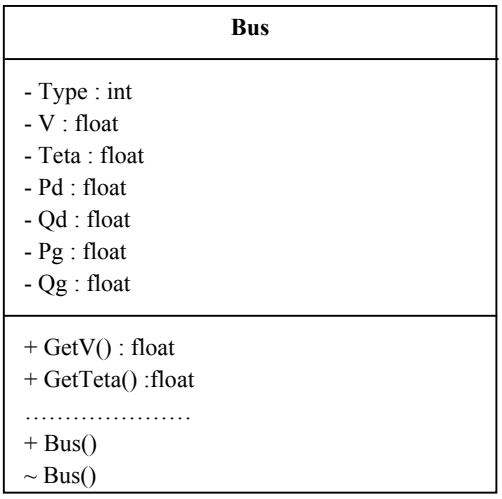


FIGURE 3.3 Classe Bus

Les principaux attributs et méthodes de la classe Bus sont :

- Type : type de jeux de barres (de charge, de génération ou de référence) ;
- V, Teta : module et phase de la tension du jeu de barres ;
- Pd, Qd : puissance active et réactive de charge au jeu de barres ;
- Pg, Qg : puissance active et réactive de génération au jeu de barres ;
- GetV() : fonction membre pour accéder à l'attribut V ;
- GetTeta() : fonction membre pour accéder à l'attribut Teta ;
- Bus() : constructeur ;
- ~Bus() : destructeur.

L'implémentation de cette classe dans la GUI est illustrée par la figure 3.4. Les données des différents paramètres de cette classe qui sont reliées directement à la base de données sont visualisées sur la boite de dialogue Data Bus. Toutes les fenêtres dérivent de la classe de base TForm de C++ Builder. Quant l'utilisateur clique sur le symbole de Bus au moment de sa création pour la première fois ou par le biais du bouton droit à n'importe quel moment, la boite de dialogue est ouverte et l'utilisateur peut spécifier les paramètres de cet élément (figure 3.5).

FIGURE 3.4 Les différentes positions du symbole de Bus

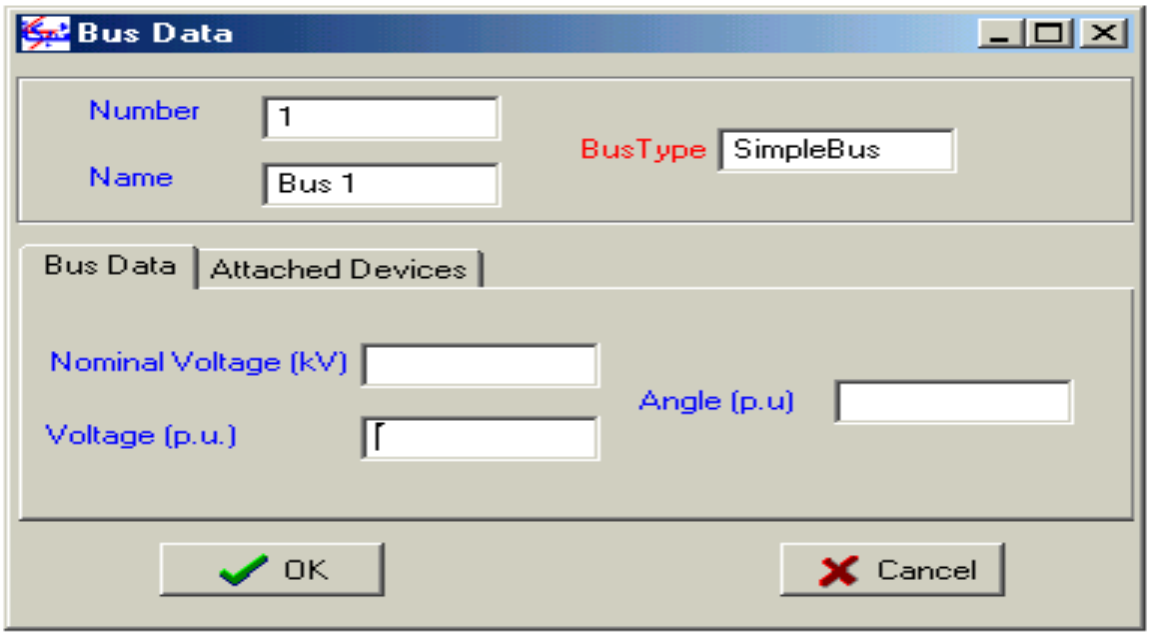


FIGURE 3.5 La boite de dialogue de Bus

3.4 Eléments en série : classe Branch

La classe Branch est une classe abstraite, elle sert de base à tous les éléments séries du réseau électrique (éléments connectés entre deux jeux de barres distincts, la ligne et le transformateur). Cette classe est illustrée par la figure 3.6.

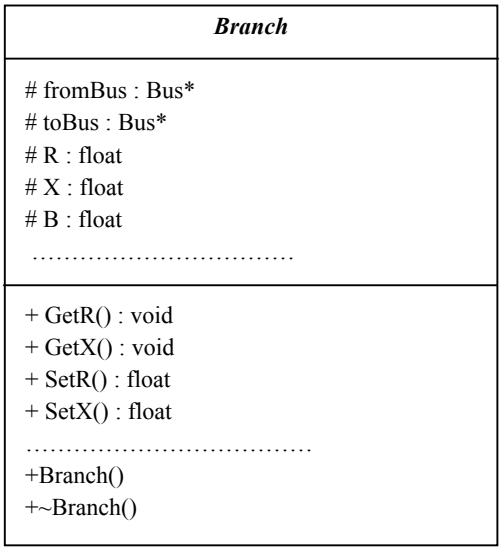


FIGURE 3.6 Classe Branch

Les principaux attributs et méthodes de cette classe sont :

- fromBus, toBus : pointeurs aux objets jeux de barres Bus auxquels l'élément est relié ;
- R, X, B : résistance, réactance et susceptance de l'élément ;
- Pflow, Qflow : écoulement de puissance dans l'élément série ;
- GetR(), GetX() : méthodes d'accès aux paramètres R et X en lecture ;
- SetR(), SetX() : méthodes d'accès aux paramètres R et X en écriture ;
- Branch() : méthode ou fonction de création de l'élément ;
- ~Branch() : méthode ou fonction de destruction de l'élément.

La connexion de l'élément à ses deux jeux de barres, aussi bien que dans le cas des éléments en dérivation, est faite automatiquement au moment de sa création dans l'éditeur graphique. C'est pourquoi, si les deux bornes n'appartiennent pas à deux jeux de barres distincts l'élément Branch ne peut pas être crée. Les jeux de barres doivent être crées en premier lieu.

3.4.1 Elément ligne de transmission : classe Line

La classe Line représente les lignes de transmission du réseau électrique. Le modèle algébrique considéré est le circuit en π. L'implémentation orientée objets de ce modèle est la

classe Line de la figure 3.7. Elle dérive directement de la classe Branch et ainsi elle hérite tous ses attributs et méthodes.

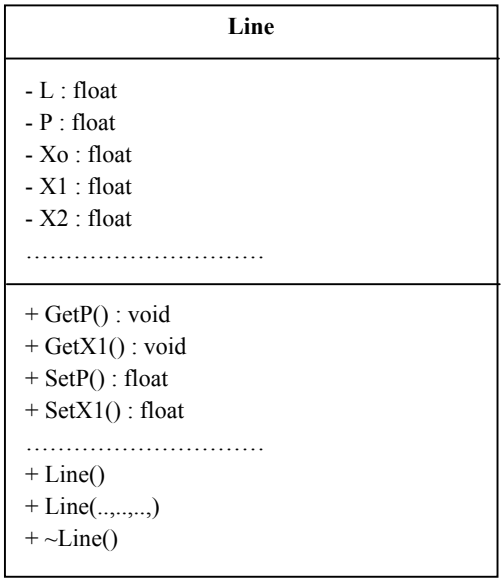


FIGURE 3.7 Classe Line

Les principaux attributs et méthodes de cette classe sont :

- L : longueur de la ligne ;
- P : puissance de la ligne ;
- Xo, X1, X2 : réactance homopolaire, directe et inverse de la ligne.
- GetP(), GetX1() : méthodes d'accès aux paramètres P et X1 en lecture ;
- SetP(), SetX1() : méthodes d'accès aux paramètres P et X1 en écriture ;
- Line() et Line(..,..,..) : deux constructeurs de line ;
- ~Line() : destructeur de line.

L'implémentation de cette classe dans la GUI est illustrée par la figure 3.8. Les différentes positions sont obtenues dans n'importe quelle direction. Après avoir cliquer sur le symbole de Line, l'utilisateur doit positionner le curseur sur le premier jeu de barres et glisser la souris pour atteindre le deuxième jeu de barres et relâcher la souris. La boite de dialogue correspondante est ouverte automatiquement pour permettre la saisie des données si l'utilisateur le désire, si non cette opération peut se faire plus tard en cliquant sur l'élément par le biais du bouton droit de la souris. La boite de dialogue de la classe Line est illustrée par les figures 3.9 et 3.10.

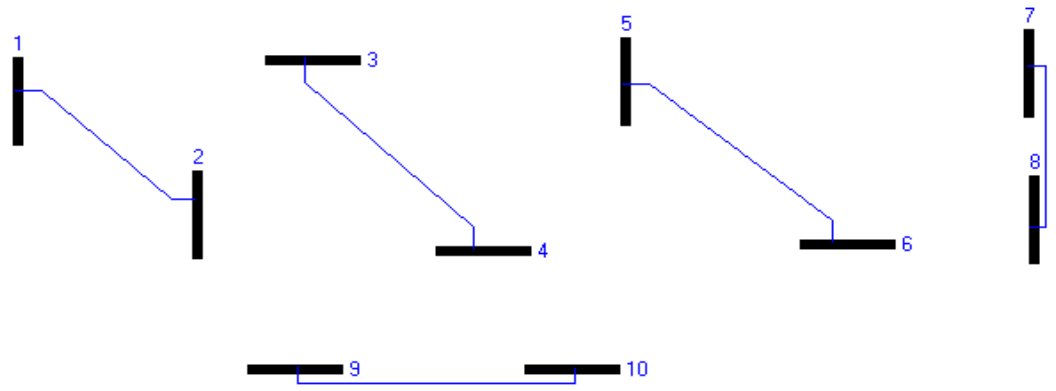


FIGURE 3.8 Les différentes positions du symbole de Line

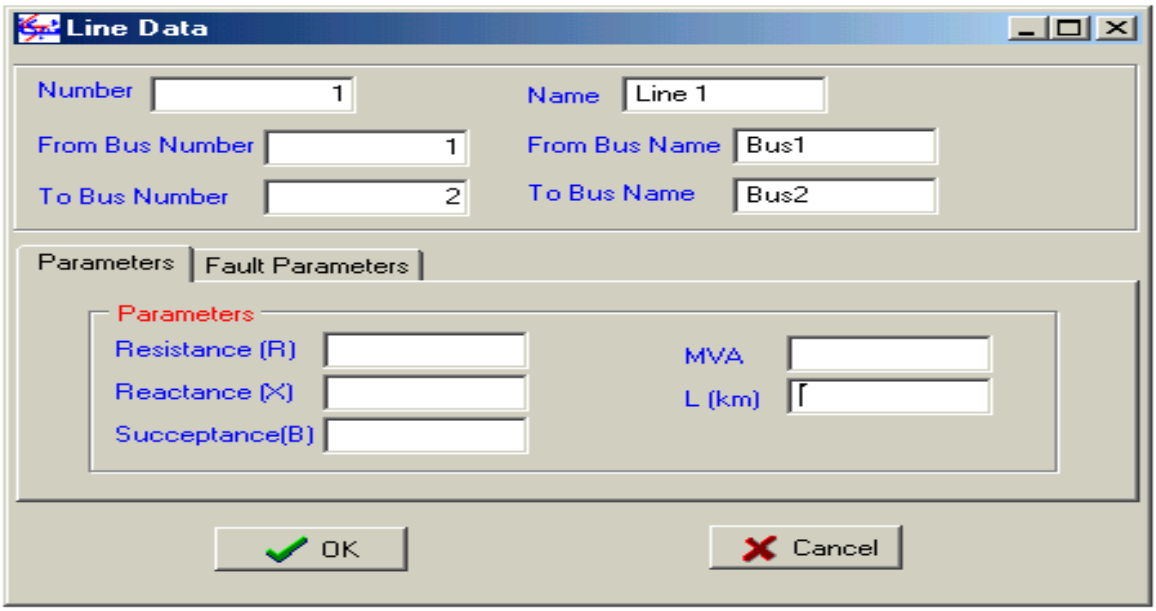


FIGURE 3.9 La boite de dialogue de Line : Parameters

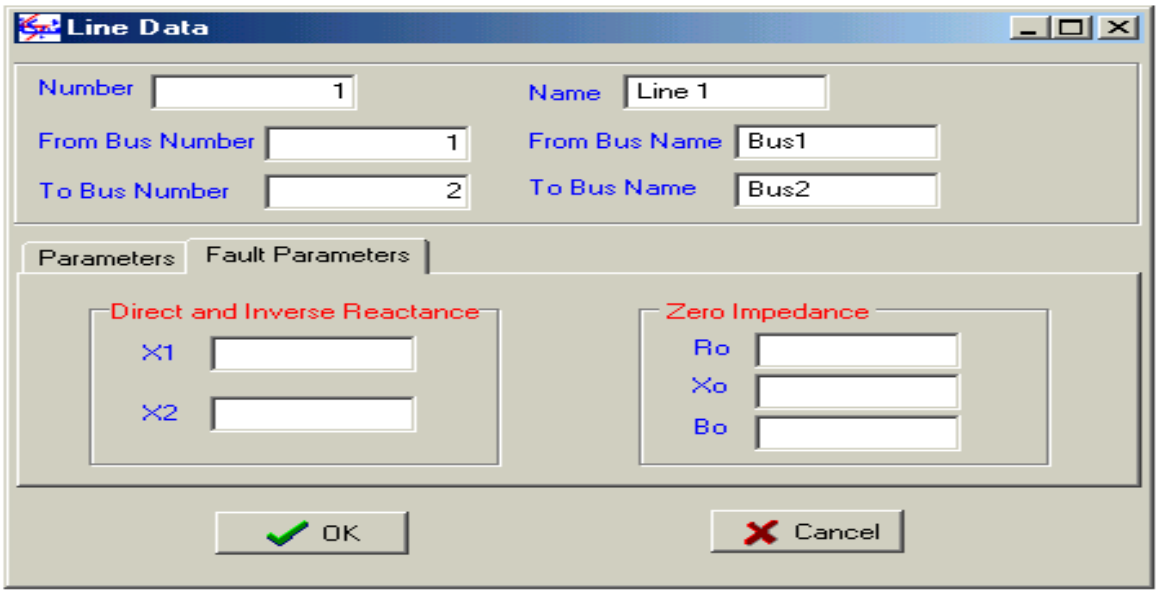


FIGURE 3.10 La boite de dialogue de Line : Fault Parameters

3.4.2 Elément Transformateur : classe Transformer

Dans le domaine des réseaux électriques une simple impédance série suffit pour la modélisation du transformateur. La classe Transformer de la figure 3.11 représente ce modèle implémenté par la MOO.

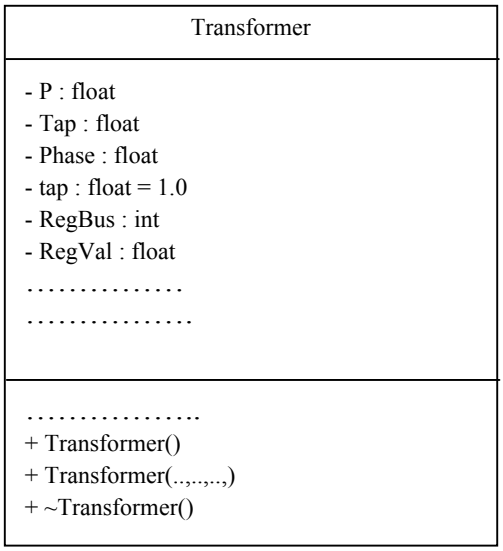


FIGURE 3.11 Classe Transformer

Les principaux attributs et méthodes de cette classe sont :

- P : Puissance du transformateur ;
- Tap : tap du transformateur ;
- Phase : angle de phase du transformateur ;
- RegBus : jeu de barres de régulation ;
- RegVal : valeur de régulation ;
- Transformer(), Transformer(..,..,..,..) : deux constructeurs ;
- ~Transformer() : destructeur.

L'implémentation de cette classe dans la GUI est illustrée par la figure 3.12. Quand l'utilisateur clique sur le symbole de Transformateur pour créer un objet Transformer, la boite de dialogue qui est liée directement à la base de donnée est ouverte automatiquement. Si l'utilisateur n'a pas envie de spécifier ses différents paramètres maintenant il peut reprendre cette opération plus tard. La boite de dialogue de Transformer est représentée par les figures 3.13 et 3.14.

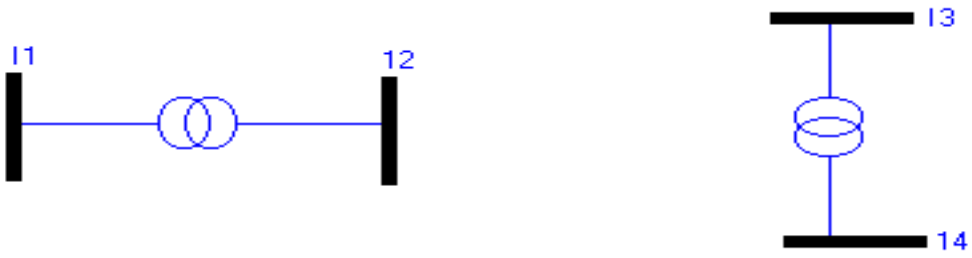


FIGURE 3.12 Les différentes positions du symbole de Transformer

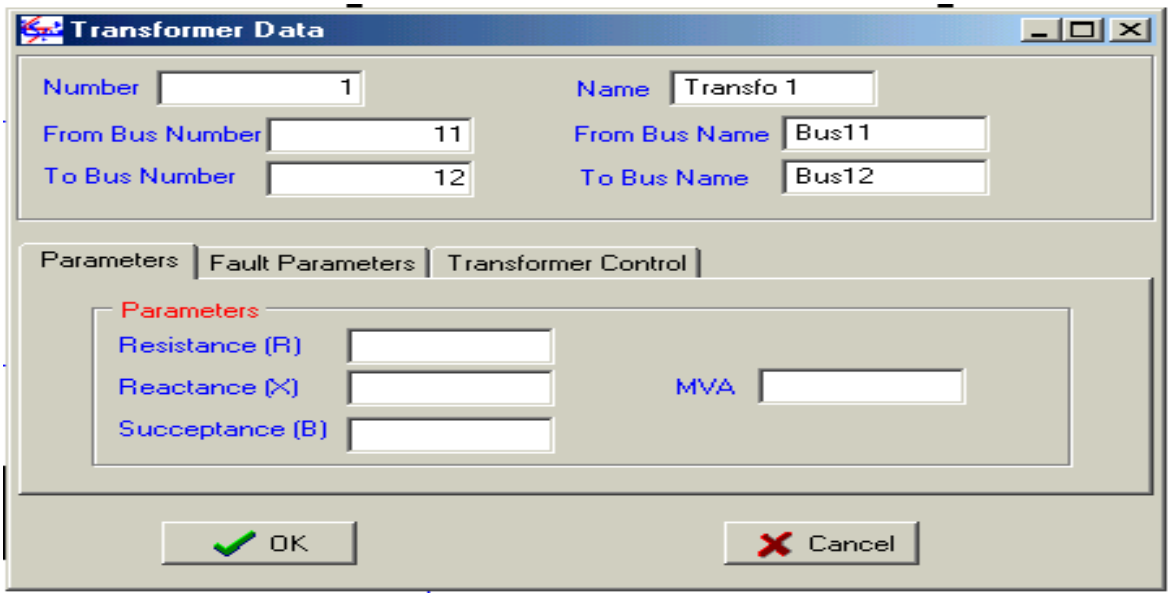


FIGURE 3.13 La boite de dialogue de Transformer : Parameters

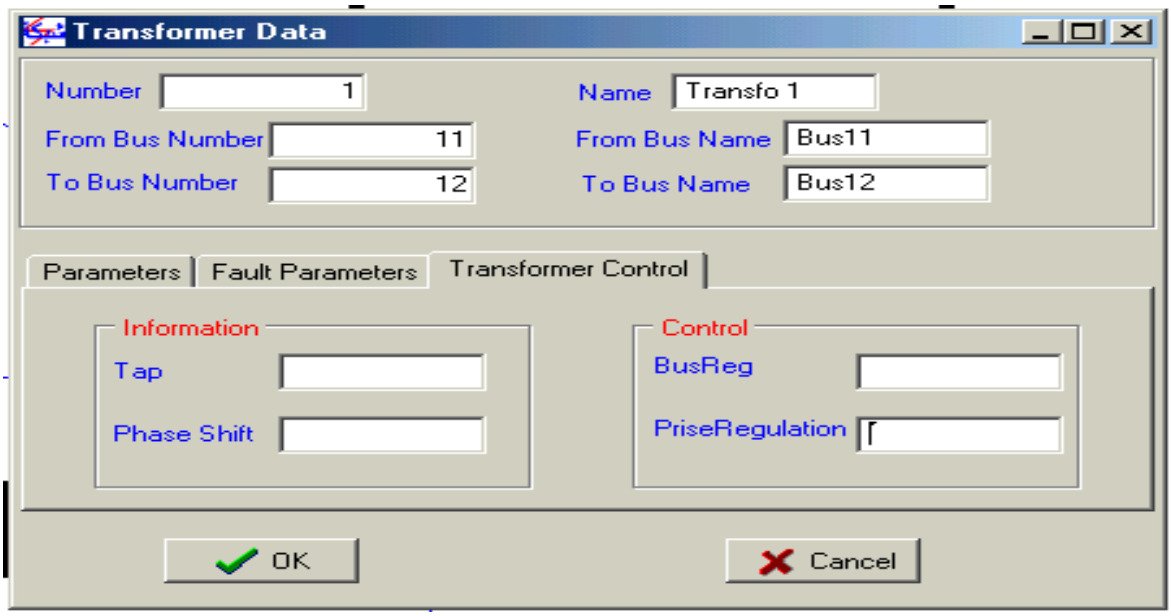


FIGURE 3.14 La boite de dialogue de Transformer : Transformer Control

3.5 Eléments en dérivation : classe Shunt

La classe Shunt est une classe abstraite, qui sert de base pour tous les éléments en dérivation dans le réseau électrique tels que le générateur, la charge et le compensateur (éléments connectés à un jeu de barres seulement). Ce sont les éléments considérés dans ce travail. Cette classe est illustrée par la figure 3.15.

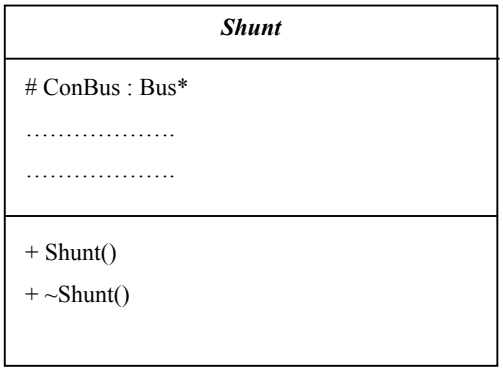


FIGURE 3.15 Classe Shunt

Les principaux attributs et méthodes de cette classe sont :

- ConBus : pointeur vers l'objet jeu de barres de connexion ;
- Shunt() : constructeur de l'élément ;
- ~ Shunt() : destructeur de l'élément.

3.5.1 Elément Charge : classe Load

Le modèle d'une charge est une admittance. La classe représentant ce modèle de charges aux jeux de barres dans un réseau électrique est celle de la figure 3.16.

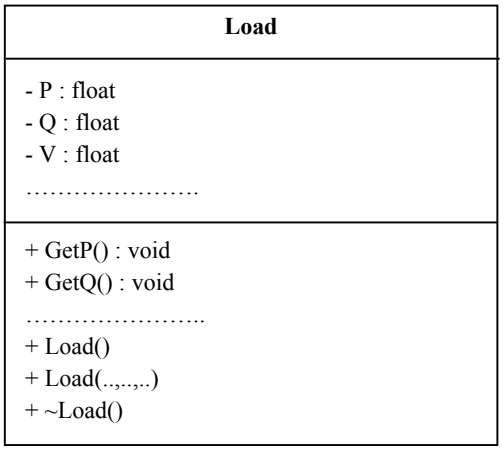


FIGURE 3.16 Classe Load

Les principaux attributs et méthodes de cette classe sont :

- P, Q : puissances active et réactive respectivement ;
- V : tension ;
- GetP(), GetQ() : méthodes d'accès aux attributs P et Q ;
- Load(), Load(..,..,..) : les deux constructeurs de l'élément ;
- ~ Load() : destructeur.

L'implémentation de cette classe dans la GUI est illustrée par la figure 3.17. La boite de dialogue de l'élément Load est présentée à la figure 3.18.

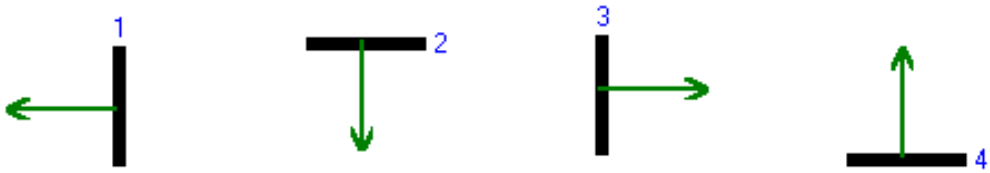


FIGURE 3.17 Les différentes positions du symbole de Load

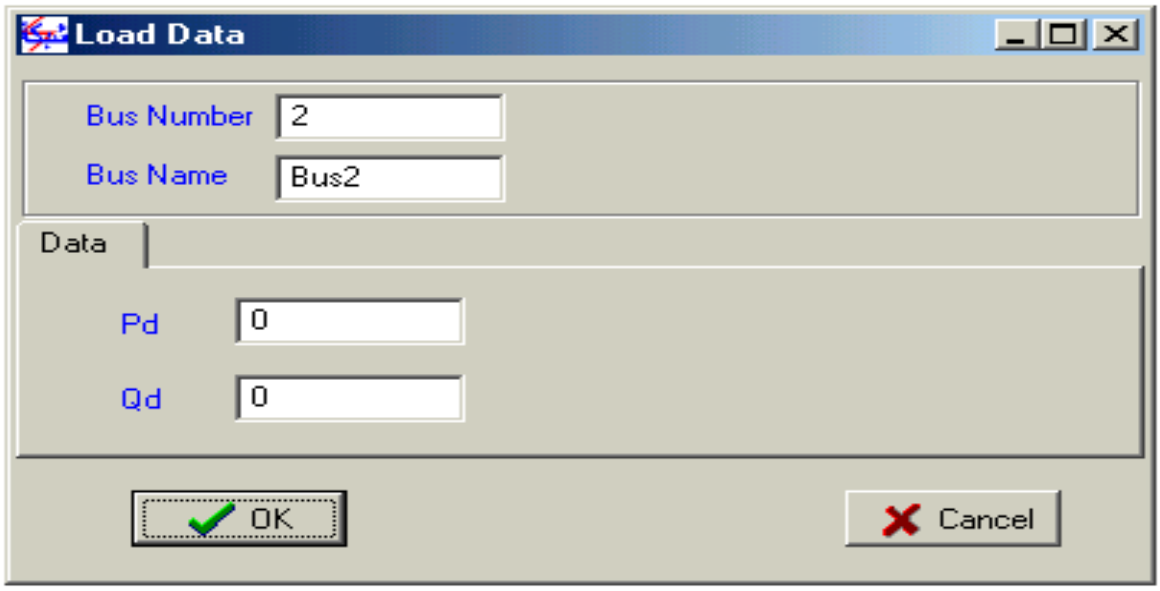


FIGURE 3.18 La boite de dialogue de Load

3.5.2 Elément compensateur : classe Compensator

Le modèle algébrique du compensateur est une impédance en série avec une capacité. Cette classe représente les compensateurs d'énergie réactive pouvant être reliés à un jeu de barres dan un réseau électrique. Il est produit par des batteries de condensateurs et/ou des réacteurs. Cette classe est illustrée par la figure 3.19.

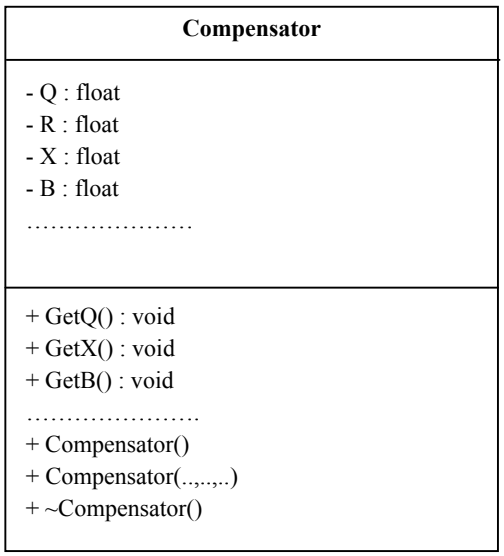


FIGURE 3.19 Classe Compensator

L'attribut principal de la classe Compensator est sa puissance réactive, représentée par Q, sa valeur sera positive si le compensateur est une batteries de condensateurs, négative si c'est un réacteur. Cette puissance réactive sera injectée au jeu de barres auquel le compensateur est relié.

- Q : puissance réactive ;
- R, X, B : résistance, réactance et susceptance respectivement ;
- GetQ(), GetX(), GetB() : méthodes d'accès aux attributs Q, X et B ;
- Compensator(), Compensator(..,..,..) : les deux constructeurs de l'élément ;
- ~ Compensator() : destructeur.

L'implémentation de cette classe dans la GUI est présentée à la figure 3.20. Quand l'utilisateur clique sur le symbole de Compensateur pour créer un objet de ce type dans l'éditeur graphique, la boite de dialogue s'ouvre automatiquement. La figure 3.21 représente la boite de dialogue de la classe Compensator.

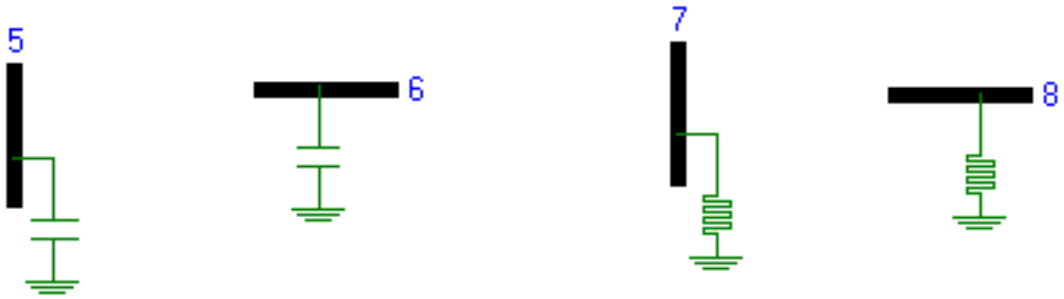


FIGURE 3.20 Les différentes positions du symbole de Compensator

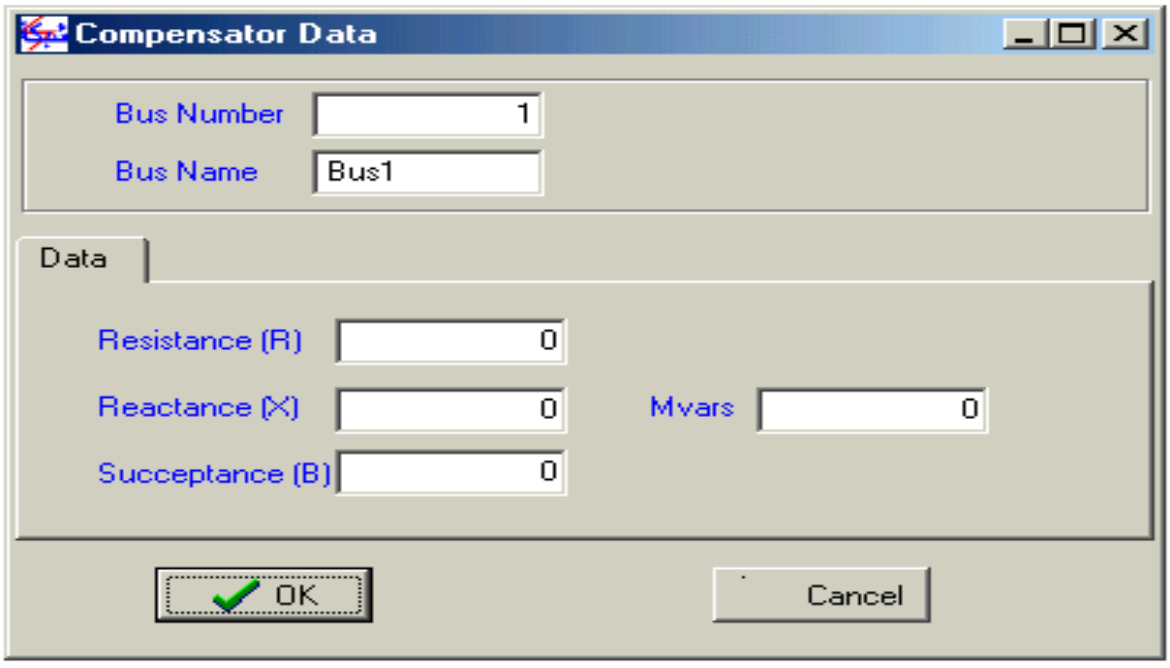


FIGURE 3.21 La boite de dialogue de Compensator

3.5.3 Elément générateur : classe Generator

Le générateur est un élément typiquement câblé en dérivation dans un réseau électrique, il est responsable de la génération de la puissance au jeu de barres auquel il est connecté. La classe Generator est illustrée par la figure 3.22.

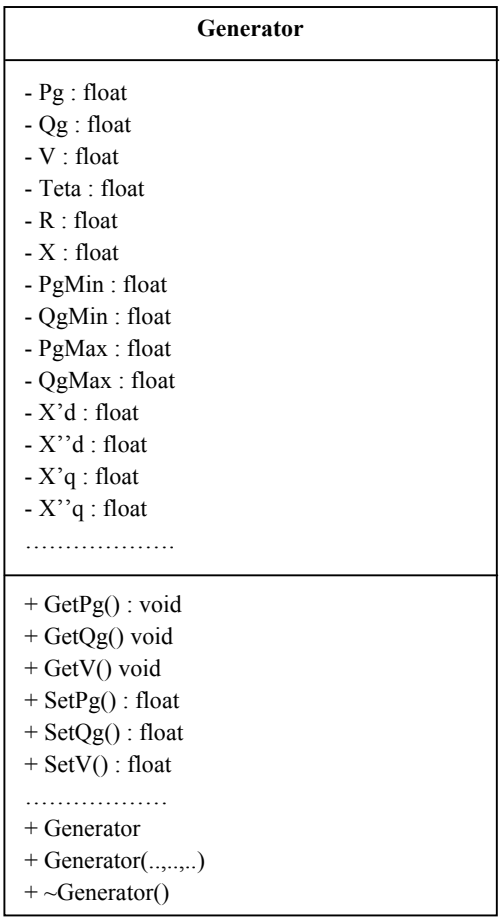


FIGURE 3.22 Classe Generator

Les principaux attributs et méthodes de Generator sont :

- Pg, Qg : puissances active et réactive du générateur;
- V, Teta : module et phase de la tension;
- PgMin, QgMin, PgMax, QgMax : Puissances active et réactive minimales et maximales ;
- X'd, X''d, X'q, X''q : réactances transitoires et subtransitoires directes et en quadrature ;
- GetPg(), GetQg(), GetV() : fonctions d'accès aux attributs Pg, Qg et V en lecture ;
- SetPg(), SetQg(), SetV() : fonctions d'accès aux attributs Pg, Qg et V en lecture ;
- Generator(), Generator(..,..,..) : deux constructeurs de la classe ;
- ~Generator() : destructeur.

FIGURE 3.23 Les différentes positions du symbole de Generator

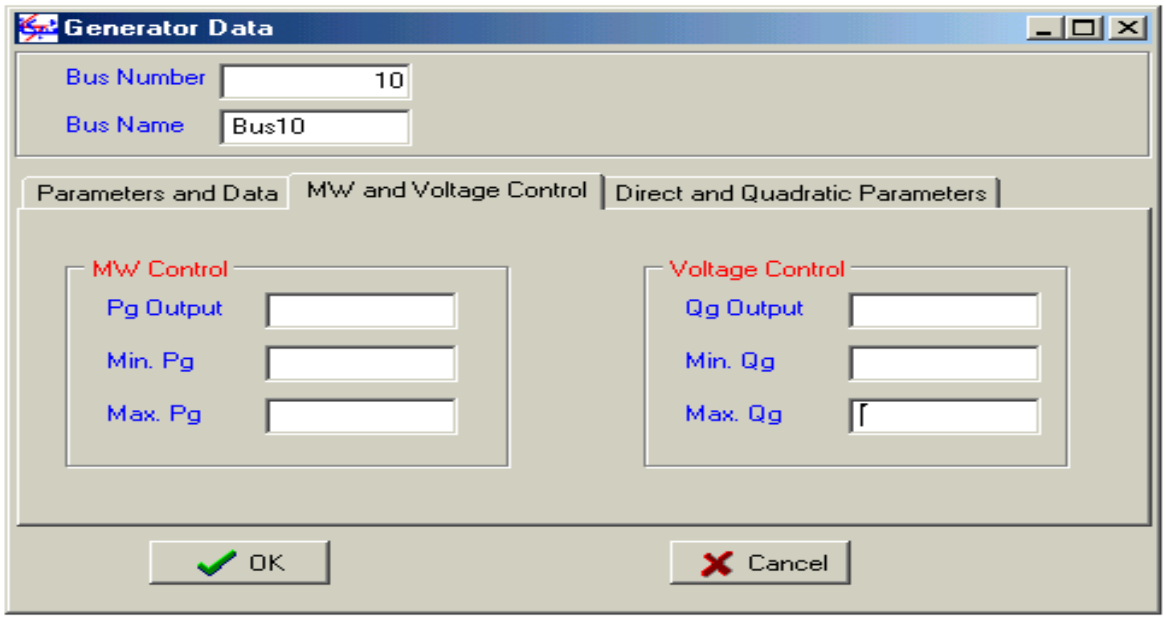


FIGURE 3.24 La boite de dialogue de Generator : MW and Voltage Control

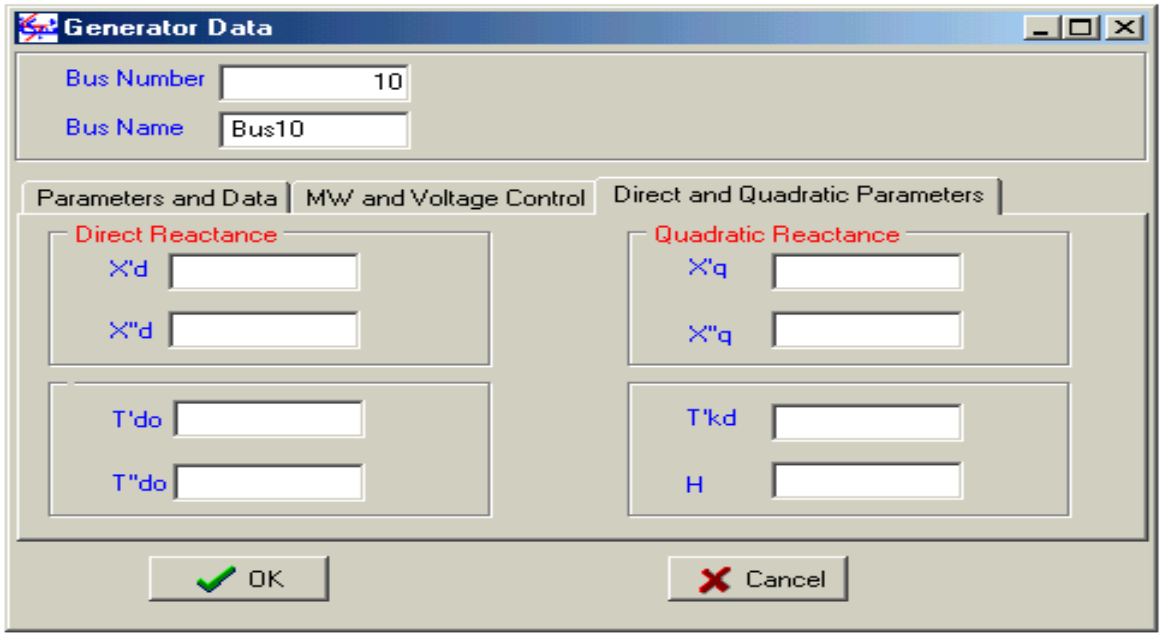


FIGURE 3.25 La boite de dialogue de Generator : Direct and Quadratic Parameters

L'implémentation de la classe Generator dans la GUI est illustrée par la figure 3.23. Quand l'utilisateur clique sur le symbole de Generator, une instance de cette classe est créée. La boite de dialogue correspondante est représentée par les figures 3.24 et 3.25.

3.6 Réseau électrique : classe PowerSystem

Le réseau électrique est représenté par une classe appelée PowerSystem, illustrée par la figure 3.26. La classe PowerSystem est très simple et ne peut exécuter aucune fonction de simulation. C'est exactement comme un laboratoire vide. Elle est responsable de la création et la destruction des éléments physiques qui constituent le réseau électrique. La classe PowerSystem est la classe de base pour toutes les applications de simulation.

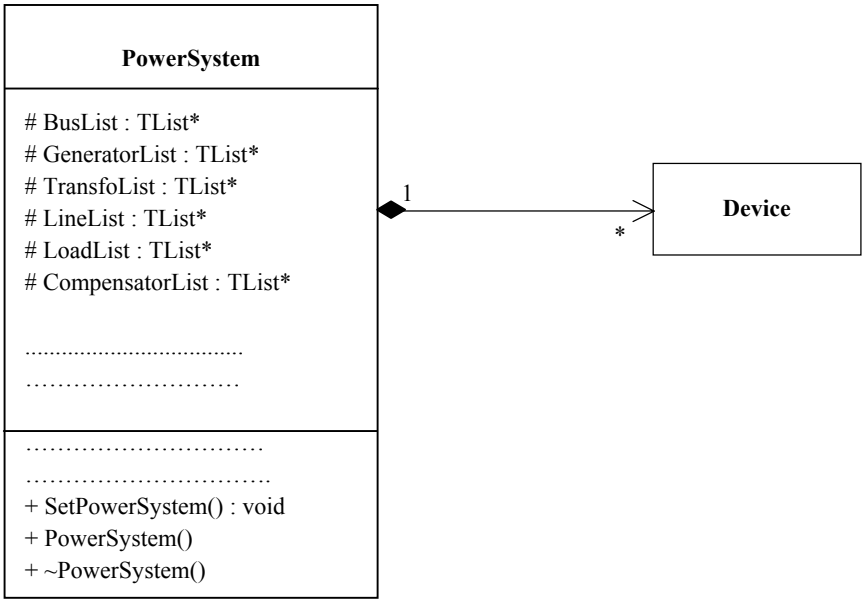


FIGURE 3.26 Classe PowerSystem

Les principaux attributs et méthodes de cette classe sont :

- BusList : liste des jeux de barres du système ;
- GeneratorList : liste des générateurs du système ;
- LineList : liste des lignes du système ;
- TransformerList : liste des transformateurs du système ;
- Compensator : liste des compensateurs du système ;
- LoadList : liste des charges du système ;
- SetPowerSystem() : mise à jour des données à partir de la base de données ;
- PowerSystem(..) : constructeur de la classe ;
- ~PowerSystem : destructeur de la classe.

Un objet réseau électrique de type PowerSystem, contient dans sa composition tous les éléments physiques du système à analyser. Ces éléments sont stockés sous forme de listes, représentées dans la classe par des attributs de type List telles que la liste des jeux de barres et la liste des lignes de transmission. N'importe quel nouveau type d'élément à ajouter dans la structure du logiciel doit être suivi par une nouvelle liste spécifique dans la classe PowerSystem. Ces listes stockent des pointeurs vers chaque élément (dans le cas du langage C++) et pas directement les éléments.

Les différentes liaisons qui existent dans cette classe PowerSystem sont représentées par le diagramme de classe de la figure 3.27

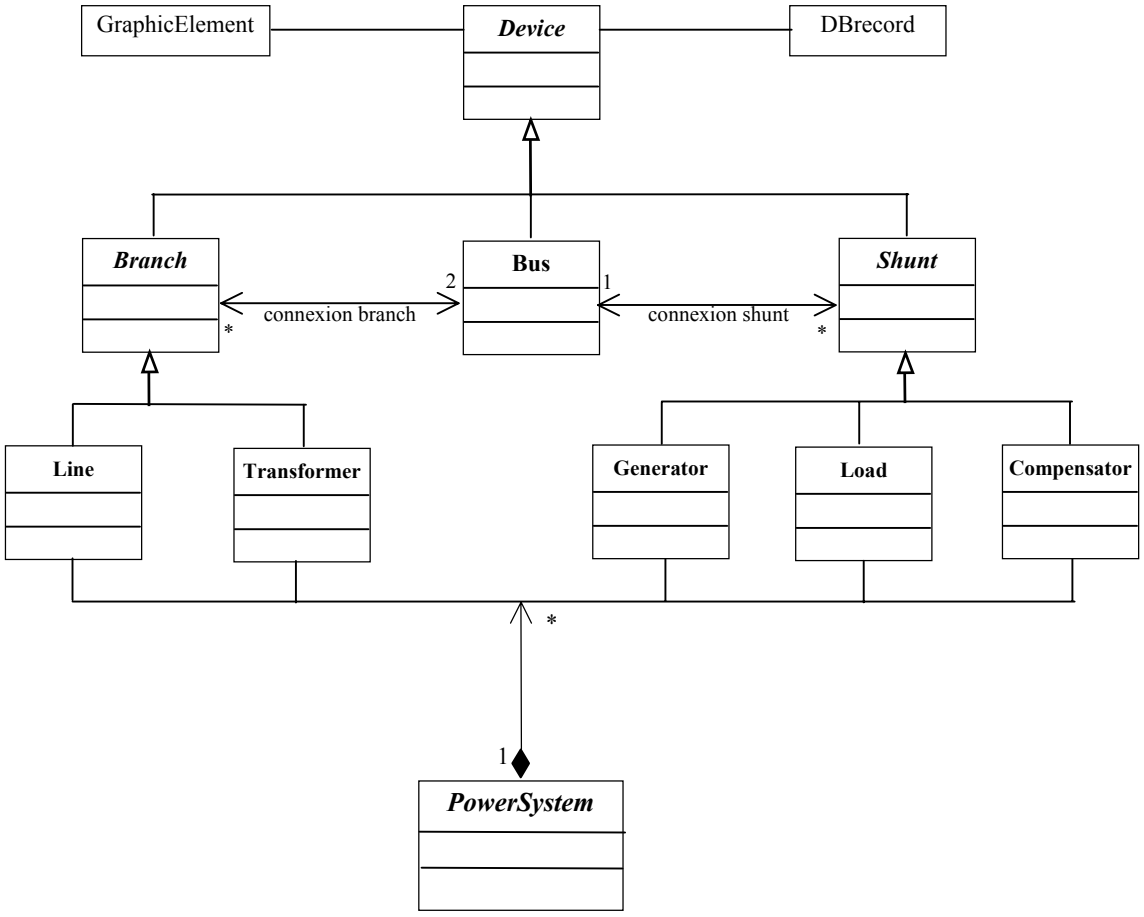


FIGURE 3.27 Diagramme de classes

3.7 Conclusion

Le produit de cette partie est le développement et l'implémentation d'un modèle orienté objets de réseaux électriques (première version) qui va servir de base pour les différentes applications. Dans la description du modèle et ses classes, on a mis l'accent sur tout ce qui est électrique et on a survolé tout ce qui est graphique (plus difficile) pour mieux présenter la TOO dans le domaine électrotechnique en général. Les classes créées sont simples et pourront être utilisées et réutilisées ce qui va permettre de se concentrer sur les spécificités de chaque application.

Les classes développées sont moyennement suffisantes pour les deux applications intégrées dans ce logiciel (Ecoulement de puissances et Courants de court circuit). Cependant les améliorations envisagées ne vont pas affecter l'architecture générale du modèle présenté à la figure 3.27, mais vont l'accroître et l'enrichir sans grandes difficultés. Au cours du développement de cette version, nous n'avons pas cesser d'apporter des changements, mais aucun de ces changements que nous avons introduis dans les dernières phases n'a ruiné l'édifice de l'architecture existante. C'est la marque d'un système orienté objets.

L'étape suivante dans ce travail qui est l'abstraction des applications n'est pas plus compliquée, premièrement parce qu'elle a un aspect beaucoup plus électrotechnique qu'informatique et deuxièmement à cause de sa structure qui sera présentée au chapitre suivant.

Chapitre 4

Modélisation des applications

4.1 Introduction

La représentation des méthodes d'analyse (ou applications) par une structure hiérarchique de classes tend à être plus simple que la représentation des éléments physiques du réseau électrique, principalement parce que ces applications sont des concepts abstraits et ne possèdent pas une vraie structure (cas physique) ce qui permet une représentation plus flexible. Donc une structure de liaison bien définie entre les applications n'existe pas ce qui élimine les connections entre leurs classes représentatives. En plus, de petites variations sur la manière de classifier les applications comme hiérarchie de classes apportera peu d'impact sur la conception de l'outil informatique.

Dans ce travail, deux applications sont considérées, l'écoulement de puissances et le calcul des courants de court circuit. Avant de décrire ces deux classes, nous présentons sans détails la technique des matrices creuses et la factorisation utilisées, nous effectuons ensuite un bref rappel de la formulation mathématique générale des deux applications pour assimiler les paramètres et les fonctions présents dans la description de ces classes.

4.2 Structure des applications

Quelques structures avaient déjà été proposées. Cependant, dans la majorité des travaux publiés dans ce domaine, peu de détails éclaircissent la structure de classification de ces applications. Puisque c'est ainsi, il est possible d'identifier depuis les structures des éléments physiques les applications convenables [1,7,18,31].

D'une manière générale, chaque application à besoin d'une série d'informations sur le réseau électrique et ses éléments constituants pour monter les structures mathématiques (matrices, systèmes linéaires, ensemble d'équations algébriques, etc.). C'est pourquoi, dans ce

travail, les applications sont représentées par une structure hiérarchique qui dérive directement de la classe PowerSystem qui représente le réseau électrique. La figure 4.1 présente la structure proposée pour la représentation des applications. Toutes les applications telles que l'écoulement de puissance (LoadFlow) et le calcul des courants de court circuit (Fault) dérivent de la classe PowerSystem.

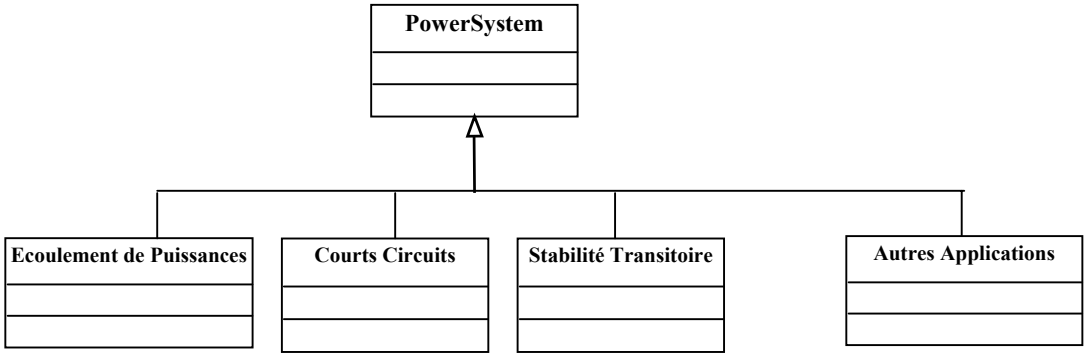


FIGURE 4.1 Diagramme de classes des applications

Chaque application représentée dans la structure ci-dessus peut être détaillée, selon les différentes méthodes qu'elle intègre. La figure 4.2 par exemple, montre les classes qui représentent les différentes méthodes de calcul de l'écoulement de puissances, telles que la méthode de Newton Raphson (NRLF), la méthode découplée rapide (FDLF) et l'optimisation de puissances (OPF). Comme autre exemple, la figure 4.3 montre la structure de classes pour représenter le calcul des courants de court circuit aux jeux de barres (CC-Bus) ou sur les lignes (CC-Line).

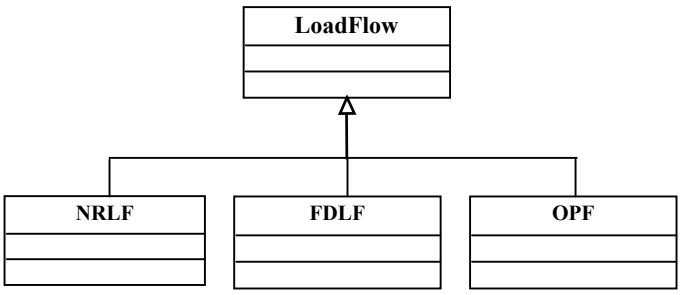


FIGURE 4.2 Structure de classes de l'écoulement de puissance (LoadFlow)

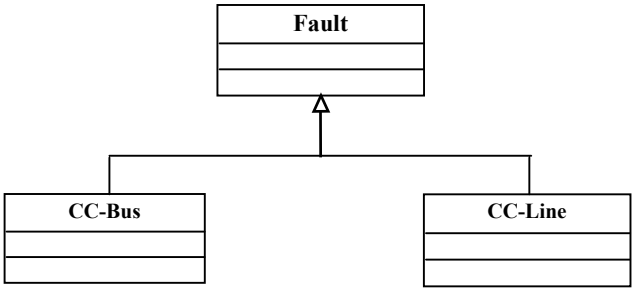


FIGURE 4.3 Structure de classes du calcul des courant de courts circuits (Fault)

Généralement, deux phases distinctes constituent ces applications : une première phase de formulation du problème par assemblage de matrices, et une seconde phase de résolution d'un système d'équations. Pour ce faire, la méthode de factorisation est choisie pour les traitements effectués sur les différentes matrices creuses telles que la matrice admittance et la matrice Jacobienne .

4.3 Matrices creuses

Dans l'analyse des réseaux électriques, beaucoup de matrices tendent à être de grande dimension et creuses telles que la matrice admittance et la matrice Jacobienne. Environ 2,8% seulement des éléments sont non nuls, pour les grand réseau ce chiffre est moins de 1%. Le stockage des éléments nuls (inutils) constitue une grande perte en espace mémoire et en temps de calcul. L'utilisation des matrices creuses permet de remédier à ce problème en ne considérant que les éléments non nuls (utils), ce qui permet un gain énorme surtout pour les réseaux de grande taille. Le seul inconvénient des matrices creuses est qu'elles sont très difficiles à implémenter comparées aux matrices pleines.

Il existe plusieurs méthodes de stockage et de manipulation des matrices creuses[32,33], la méthode utilisée dans ce travail est connue sous le nom LIFO (Last In First Out Linked List) ou les listes chaînées à plusieurs entrées. La formulation de LIFO est très simple et très puissante, elle introduit les vecteurs suivants :

- Start [i] : ce vecteur pointe au début de la liste des jeux de barres connectés au jeu de barres **i**, sa dimension est le nombre de jeux de barres ;
- Next [] : ce vecteur pointe vers la deuxième entrée de la branche, sa dimension est deux fois le nombre de branches plus le nombre de fill-ins estimé ;

— Bus [] : ce vecteur constitue la liste des jeux de barres connectés au jeu de barres **i**, sa dimension est la même que Next [].

La liste chaînée est formée juste au moment de la simulation.

4.4 Factorisation LL^T

Soit à résoudre un système d'équations linéaires, écrit sous forme matricielle :

$$\boldsymbol{A\,x = b} \tag{4.1}$$

Avec :

$\boldsymbol{A}$: matrice carrée nxn symétrique ;

$\boldsymbol{b}$: vecteur de n composantes connues.

La factorisation, qui est une méthode directe de résolution, permet d'écrire $\boldsymbol{A}$ sous la forme :

$$\boldsymbol{A = LL^T} \tag{4.2}$$

Où le facteur $\boldsymbol{L}$ est une matrice triangulaire inférieur. Le système linéaire se réécrit sous la forme :

$$\boldsymbol{LL^T\,x = b} \tag{4.3}$$

Et peut donc être traité par la résolution successive des deux systèmes triangulaires :

$$\boldsymbol{Ly = b},\ \text{puis}\ \boldsymbol{L^T\,x = y} \tag{4.4}$$

Le processus de factorisation procède par transformations successives de la matrice d'origine $\boldsymbol{A}$ pour aboutir au facteur $\boldsymbol{L}$ tel que [6,21] :

$$L_{i,j} = A_{i,j} - \sum_{k=1}^{i-1} L_{i,k} * L_{j,k} \qquad \boldsymbol{j = i\,,\,n} \tag{4.5}$$

Pour gagner en espace mémoire, la matrice originale $\boldsymbol{A}$ est détruite et la matrice $\boldsymbol{L}$ est stockée dans $\boldsymbol{A}$; donc une seule matrice est utilisée.

4.5 Ecoulement de puissances : classe LoadFlow

L'écoulement de puissances est l'un des problèmes classiques des réseaux électriques. Généralement les charges du réseau sont contrôlées au niveau des sous stations pour la majorité des consommateurs et des compagnies électriques. L'objectif d'un programme d'écoulement de puissances est de calculer les tensions à tous les jeux de barres (nœuds) ainsi que le flux des puissances actives et réactives dans les lignes et les transformateurs (branches) [34,35].

4.5.1 Formulation mathématique générale

Du point de vue écoulement de puissances, à chaque jeu de barres il y a quatre paramètres : la tension V, l'angle θ, la puissance active P et la puissance réactive Q. Suivant le type de jeux de barres, deux de ces paramètres sont spécifiés, les deux autres sont calculés.

Pour un réseau électrique dans un état statique équilibré et dont les éléments physiques ont des caractéristiques linéaires, la formulation du problème de l'écoulement de puissances est donnée par les équations ci dessous [6] :

$$P_i = \sum_{j=1}^{n} |Y_{ij}| \cdot |V_i| \cdot |V_j| \cdot Cos(\theta_{ij} - \gamma_{ij})$$
$$Q_i = \sum_{j=1}^{n} |Y_{ij}| \cdot |V_i| \cdot |V_j| \cdot Sin(\theta_{ij} - \gamma_{ij}) \qquad (4.6)$$

Avec :

i, j : jeux de barres ;

P_i, Q_i : puissances actives et réactives du jeu de barres i ;

V_i, V_j, θ_i, θ_j : modules et phases des tensions des jeux de barres i, j ;

Y_{ij}, γ_{ij} : module et phase de l'admittance de la branche i-j.

Le problème de l'écoulement de puissances est divisé en deux parties : la première est la formulation des équations (4.6), et la deuxième est la résolution de ces équations non linéaires. La méthode de Newton Raphson largement utilisée pour la résolution de ce problème donne :

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} H & N \\ J & L \end{bmatrix} \cdot \begin{bmatrix} \Delta\theta \\ \Delta V \end{bmatrix} \tag{4.7}$$

La matrice Jacobienne ***J*** est une matrice asymétrique creuse formée de sous matrices ***H***, ***N***, ***J*** et ***L***. Dans chaque itération de la méthode de Newoton Raphson, un grand système d'équations linéaires et creuses sont résout. Et puisque ***J*** est modifiée à chaque itération, l'opération de factorisation est répétée à chaque fois également. Pour réduire ces calculs, des approximations son faites. Pour un réseau électrique équilibré on observe un grand couplage entre ***P*** et $\boldsymbol{\theta}$ d'un coté et entre ***Q*** et ***V*** d'un autre coté, autrement dit :

Une variation $\Delta \boldsymbol{P}$ entraîne une variation $\Delta\boldsymbol{\theta}$ et n'affecte pas beaucoup la tension ***V*** ($\Delta \boldsymbol{V}$=0) ;
Une variation $\Delta \boldsymbol{Q}$ entraîne une variation $\Delta \boldsymbol{V}$ et n'affecte pas beaucoup la phase $\boldsymbol{\theta}$ ($\Delta\boldsymbol{\theta}$=0).
De ce fait la Jacobienne est découplée (***N* = 0** et ***J* = 0**).

De plus, le besoin de résoudre ces équations le plus rapidement possible à conduit à introduire des approximations supplémentaires (***V***= 1 p.u, $\boldsymbol{\theta}$=0) et le système d'équations (4.7) devient [6] :

$$\begin{bmatrix} \Delta P/V \\ \Delta Q/V \end{bmatrix} = \begin{bmatrix} B' & 0 \\ 0 & B'' \end{bmatrix} \cdot \begin{bmatrix} V\Delta\theta \\ \Delta V \end{bmatrix} \tag{4.8}$$

Où la matrice Jacobienne découplée rapide est constituée des sous matrices ***B'*** et ***B''*** qui sont symétriques et constantes. Par conséquent, la factorisation est nécessaire une seule fois quelque soit le nombre d'itérations. Cette méthode découplée rapide du problème d'écoulement de puissances (FDLF) donne presque les mêmes résultats que la méthode de Newton Raphson sans approximations, c'est pourquoi elle est largement utilisée, elle a été appliquée dans ce présent travail. Plus de détails dans [6].

4.5.2 Implémentation orientée objets de FDLF

L' application d'écoulement de puissances (FDLF) implémenté dans ce travail est représentée par la classe LoadFlow. Cette classe est illustrée par la figure 4.4.

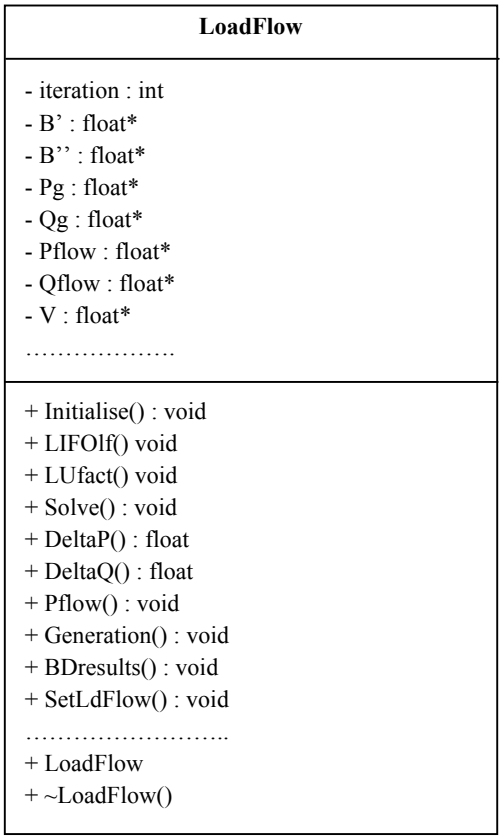


FIGURE 4.4 Classe LoadFlow

Les principaux attributs et méthodes de LoadFlow sont :

- Iteration : nombre d'itérations maximales ;
- B' : première matrice creuse de la Jacobienne ;
- B'' : deuxième matrice creuse de la Jacobienne ;
- Pg, Qg : vecteurs de puissances actives et réactives des jeux de barres de génération;
- V : vecteur des tensions des jeux de barres ;
- Pflow, Qflow : vecteurs des flux de puissances actives et réactives dans les branches ;
- Initialise() : fonction membre qui initialise toutes les matrices et les vecteurs considérés ;
- LIFOlf() : méthode qui permet la construction des matrices creuses B' et B'' et leur stockage dans la liste chaînée ;
- LUfact () : permet la factorisation de B' et B'' ;
- Solve() :donne la solution du système linéaire ;
- DeltaP(), DeltaQ() : calculent les pertes actives et réactives ;
- Pflow(), Qflow() : calculent les flux de puissances actives et réactives des éléments séries ;
- Generation() : calcule les puissances actives et réactives des jeux de barres ;

— BDResult() : renvoie les résultats vers la base de données et affiche les différents résultats dans la fenêtre d'exécution de l'application écoulement de puissances ;

— SetLoadFlow() : exécute la classe de l'application ;

— LoadFlow() : constructeur ;

— ~LoadFlow() :destructeur ;

Pour illustrer les activités qu'un objet de type LoadFlow exécute, un diagramme d'activités pour le calcul d'écoulement de puissances est conçu, il est représenté par la figure 4.5.

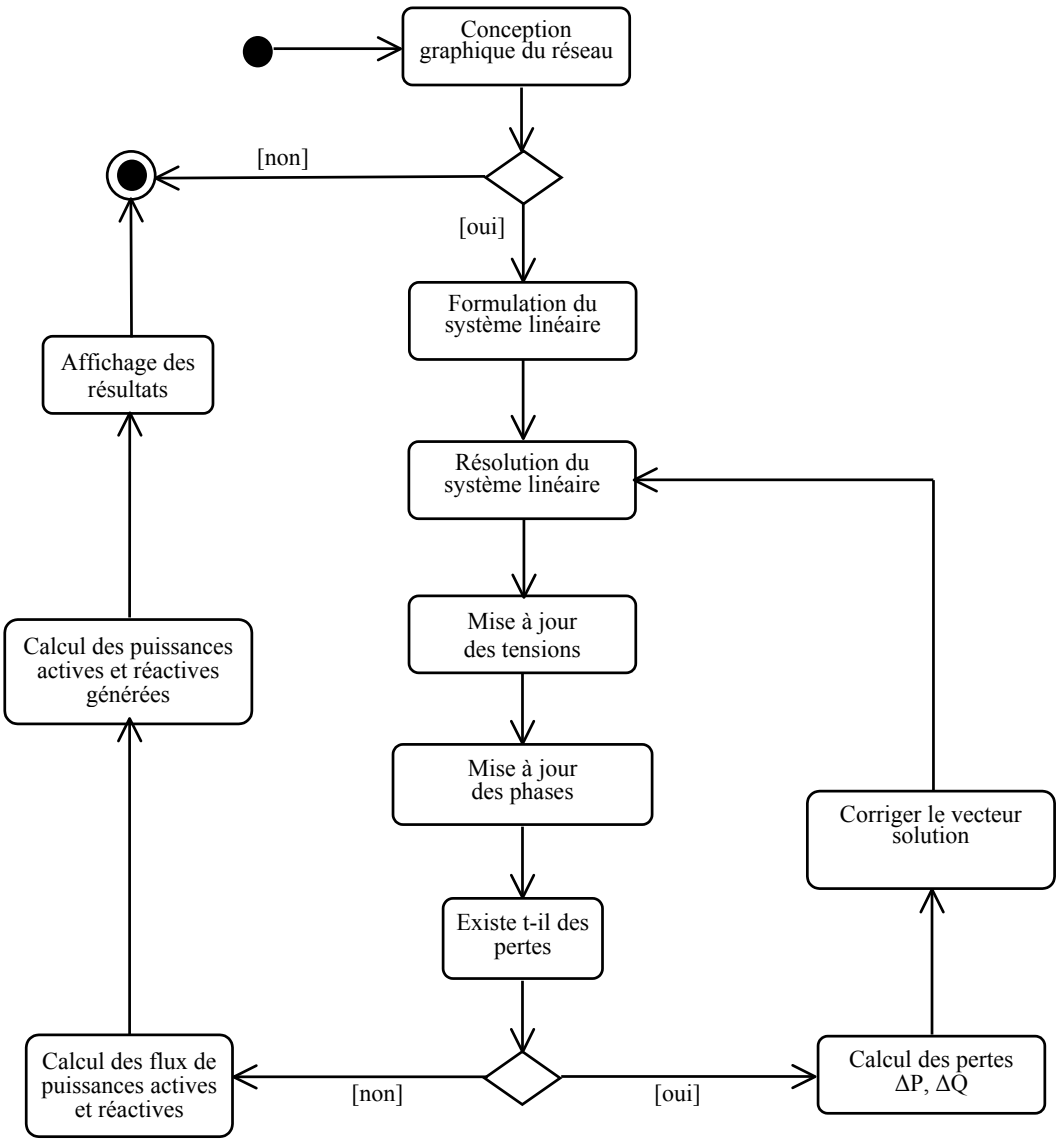


FIGURE 4.5 Diagramme d'activités de l'application LoadFlow

Les résultats de l'application seront affichés sous la forme présentée aux figures 4.6 et 4.7 se composent d'une liste de jeux de barres du réseau électrique étudié avec les tensions, les angles de phases, les puissances actives et réactives, et d'une liste des éléments série (transformateurs et lignes de transmission) avec les flux de puissances actives et réactives.

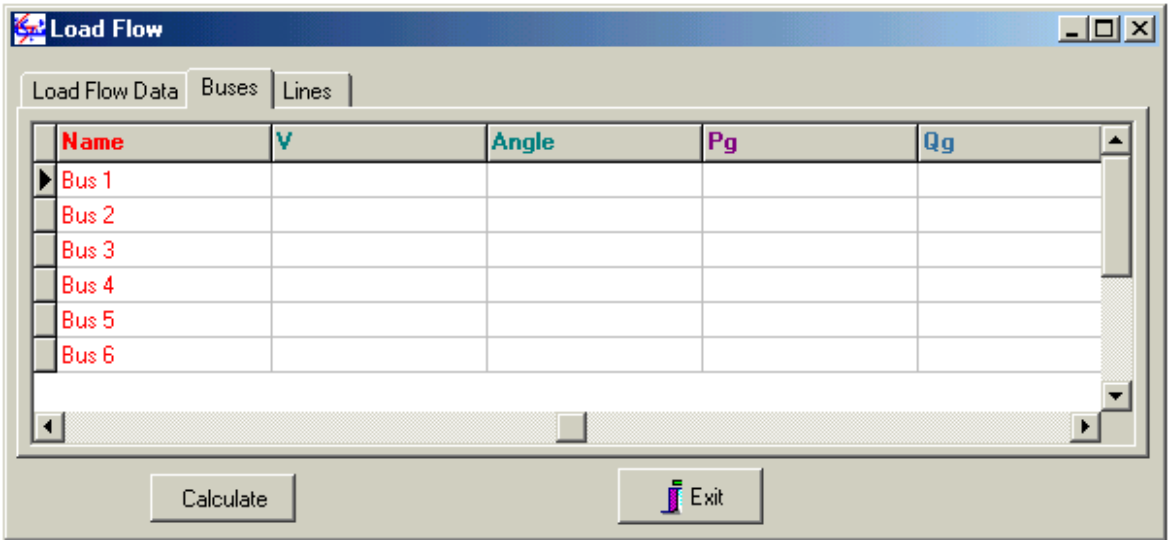


FIGURE 4.6 Présentation des résultats de l'écoulement de puissances :Buses

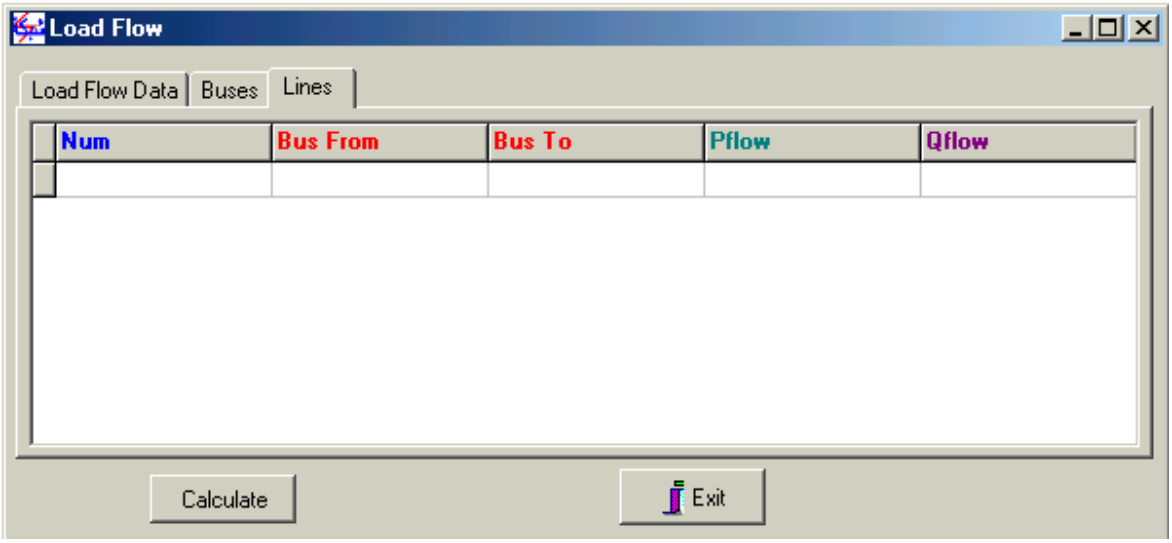


FIGURE 4.7 Présentation des résultats de l'écoulement de puissances :Lines

4.6 Analyse des courants de court circuit : classe Fault

Les réseaux électriques représentent des investissements considérables, pour des raisons techniques et économiques il est impossible que leur construction soit faite sans risque de défauts de fonctionnements. Les réseaux électriques sont donc affectés par des défauts qui peuvent mettre en cause la pérennité du matériel et la qualité de service. Parmi ces défauts très répandus les courts circuits [30,33].

Un court circuit peut être monophasé, biphasé ou triphasé. L'intensité du courant de court circuit est une caractéristique importante, elle détermine la sévérité de la contrainte appliquée au réseau électrique et au matériel en défaut (fonctionnement du réseau, tenue du matériel, qualité d'énergie). Les courts circuits sont donc des incidents qu'il faut éliminer le plus vite possible, c'est le rôle des protections qui sont avant tout choisies et dimensionnées sur la base des calculs des courants de court circuit.

4.6.1 Formulation mathématique générale

Pour un défaut symétrique (triphasé), l'étude se ramène à celle du réseau monophasé dans le système direct. Cependant, pour les défauts asymétriques, les calculs relatifs aux systèmes déséquilibrés deviennent très complexes. La méthode dite des composantes symétriques permet de les simplifier notablement.

La méthode des composantes symétriques consiste à décomposer un système de trois vecteurs quelconques en trois systèmes de vecteurs symétriques, direct, inverse et homopolaire. Appliquée aux systèmes triphasés déséquilibrés, elle permet d'avoir :

$$\begin{bmatrix} V^a \\ V^b \\ V^c \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ a^2 & a & 1 \\ a & a^2 & 1 \end{bmatrix} \cdot \begin{bmatrix} V^1 \\ V^2 \\ V^0 \end{bmatrix} \tag{4.8}$$

ou en écriture compacte :

$$V^{abc} = T \cdot V^{120}$$

où :

a, b, c représentent les indices de phases ;

1, 2, 0 représentent les indices des systèmes direct, inverse et homopolaire respectivement ;

T est appelée matrice de transformation ;

$a=-\frac{1}{2}+j\frac{\sqrt{3}}{2}$

4.6.1.1 Court circuit symétrique

Pour un court circuit symétrique à un jeu de barres **q**, les expressions de courant et de tensions sont données comme suit :

$$I_q(F)=\frac{V_q(0)}{z_{qq}+z_f} \tag{4.9}$$

$$V_q(F)=\frac{z_f}{z_{qq}+z_f}V_q(0) \tag{4.10}$$

$$V_i(F)=V_i(0)-\frac{z_{iq}}{z_{qq}+z_f}V_q(0) \tag{4.11}$$

$$I_{ij}(F)=\left[V_i(F)-V_j(F)\right]y_{ij} \tag{4.12}$$

Où :

$\boldsymbol{I_q(F)}$: courant de défaut au jeu de barres **q** ;

$\boldsymbol{V_q(0)}$, $\boldsymbol{V_q(F)}$: tension du jeu de barres **q** avant et durant le défaut respectivement ;

$\boldsymbol{z_f}$: impédance de défaut ;

$\boldsymbol{z_{qq}}$, $\boldsymbol{z_{iq}}$: éléments **qq** et **iq** de la matrice impédance du réseau **ZBus** ;

$\boldsymbol{V_i(F)}$: tensions du reste des jeux de barres **i** durant le défaut ;

$\boldsymbol{I_{ij}(F)}$: courants de branches **ij** ;

$\boldsymbol{y_{ij}}$: admittances de branche **ij**.

4.6.1.2 Court circuit asymétrique

Pour un court circuit asymétrique à un jeu de barres **q**, les expressions de courants et de tensions pour un court circuit monophasé par exemple sont données comme suit :

$$\begin{bmatrix} I_q^1(F) \\ I_q^2(F) \\ I_q^0(F) \end{bmatrix} = \frac{V_q^1(0)}{z_{qq}^1 + z_{qq}^2 + z_{qq}^0} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{4.13}$$

$$\begin{bmatrix} V_q^1(F) \\ V_q^2(F) \\ V_q^0(F) \end{bmatrix} = \frac{V_q^1(0)}{z_{qq}^1 + z_{qq}^2 + z_{qq}^0} \begin{bmatrix} z_{qq}^2 + z_{qq}^0 \\ -z_{qq}^2 \\ -z_{qq}^0 \end{bmatrix} \tag{4.14}$$

$$\begin{bmatrix} V_i^1(F) \\ V_i^2(F) \\ V_i^0(F) \end{bmatrix} = \begin{bmatrix} V_i^1(0) \\ 0 \\ 0 \end{bmatrix} - \frac{V_q^1(0)}{z_{qq}^1 + z_{qq}^2 + z_{qq}^0 + 3Z_f} \begin{bmatrix} z_{iq}^1 \\ -z_{iq}^2 \\ -z_{iq}^0 \end{bmatrix} \tag{4.15}$$

Toutes les grandeurs de phase V^a, V^b, V^c et I^a, I^b, I^c, sont ensuite déterminées en fonctions des grandeurs directes, inverses et homopolaires telles que : V^1, V^2, V^0 et I^1, I^2, I^0
L'expression du courant de défaut par exemple est :

$$I_q^a(F) = \frac{3V_q^a(0)}{z_{qq}^1 + z_{qq}^2 + z_{qq}^0 + 3Z_f} \tag{4.16}$$

D'après les expressions ci dessus, les courants de court circuit sont fonction des éléments de la matrice impédance ***ZBus*** du réseau. La méthode la plus utilisée pour la détermination de ***ZBus*** est la factorisation (ou inversion) de la matrice admittance ***YBus***, surtout pour les réseaux de grande taille où on exploite le fait qu'elle est creuse.

4.6.2 Implémentation orientée objets du calcul des courts circuits

L' application de calcul des courants de court circuit implémenté dans ce travail est représentée par la classe Fault. Cette classe est illustrée par la figure 4.8.

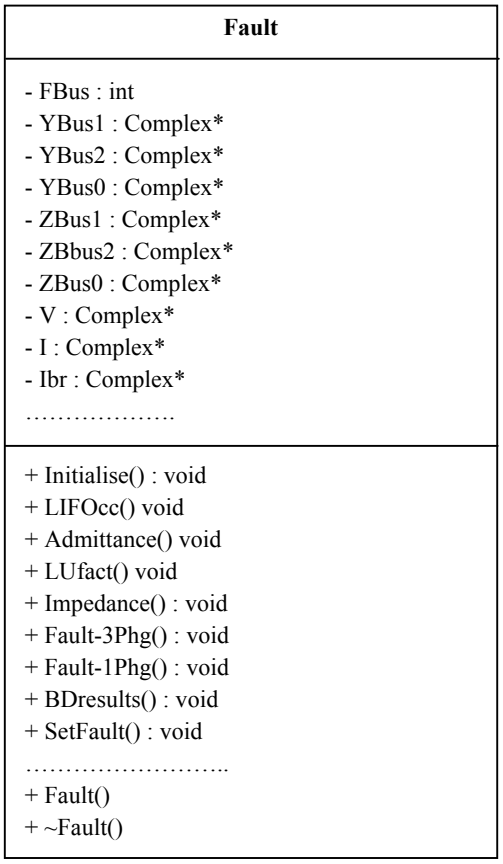


FIGURE 4.8 Classe Fault

Les principaux attributs et méthodes de Fault sont :

- FBus : jeu de barres de défaut ;
- ZBus1, ZBus2, ZBus0 : matrices impédances directe, inverse et homopolaire respectivement (en réalité ce sont des vecteurs) ;
- YBus1, YBus2, YBus0 : matrices admittances directe, inverse et homopolaire respectivement (ce sont des vecteurs également) ;
- V : vecteur complexe des tensions de défauts des jeux de barres ;
- I : vecteur complexe des courants de défaut des jeux de barres ;
- Ibr : vecteur complexe des courants de défaut des branches ;
- Initialise() : fonction membre qui initialise toutes les matrices et les vecteurs considérés ;
- LIFOcc() : fonction qui permet la construction de la liste chaînée et la formulation préliminaire des matrices admittances YBus1, YBus2, YBus0 et leur stockage ;
- Admittance() : fonction membre qui donne les matrices admittances finales ;
- LUfact () : permet la factorisation des matrices admittances ;
- Impedance() :calcule les matrices impédances ZBus1, ZBus2, ZBus0 ;

— Fault-3phg() : calcule le courant de court circuit triphasé à la terre, les tensions de défaut et les courants de branches symétriques ;
— Fault-1phg() : calcule le courant de court circuit monophasé à la terre, les tensions de défaut et les courants de branches asymétriques ;
— BDResult() : renvoie les résultats vers la base de données et affiche les différents résultats dans la fenêtre d'exécution de l'application de court circuit ;
— SetFault() : exécute la classe de l'application ;
— Fault() : constructeur ;
— ~Fault() :destructeur ;

Pour illustrer les activités qu'un objet de type Fault exécute, un diagramme d'activités pour le calcul des courants de court circuit est conçu, il est représenté par la figure 4.9.

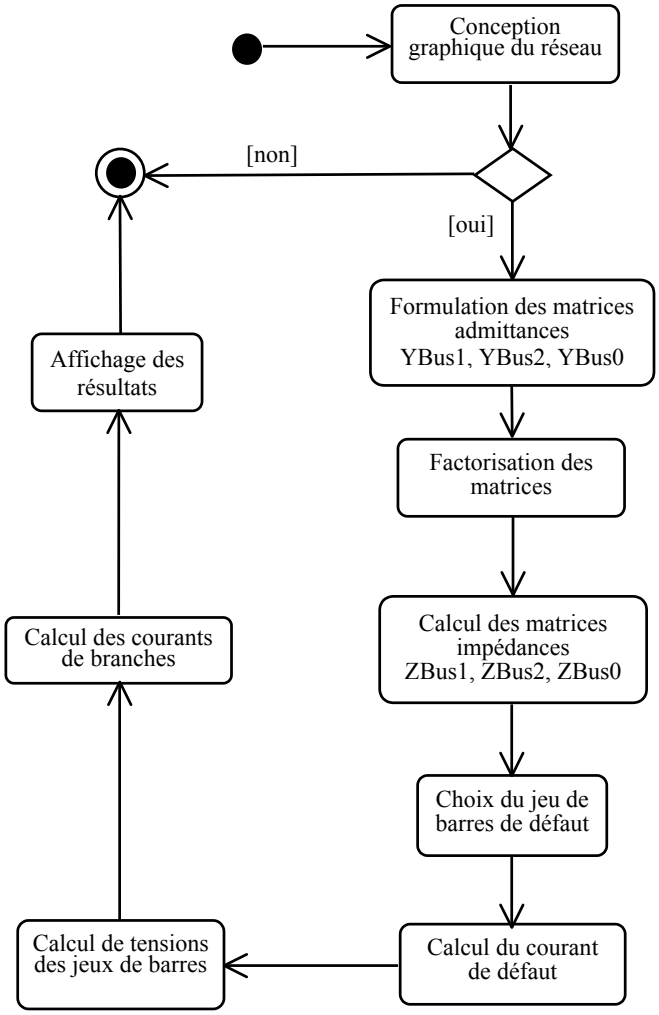


FIGURE 4.9 Diagramme d'activités de l'application Fault

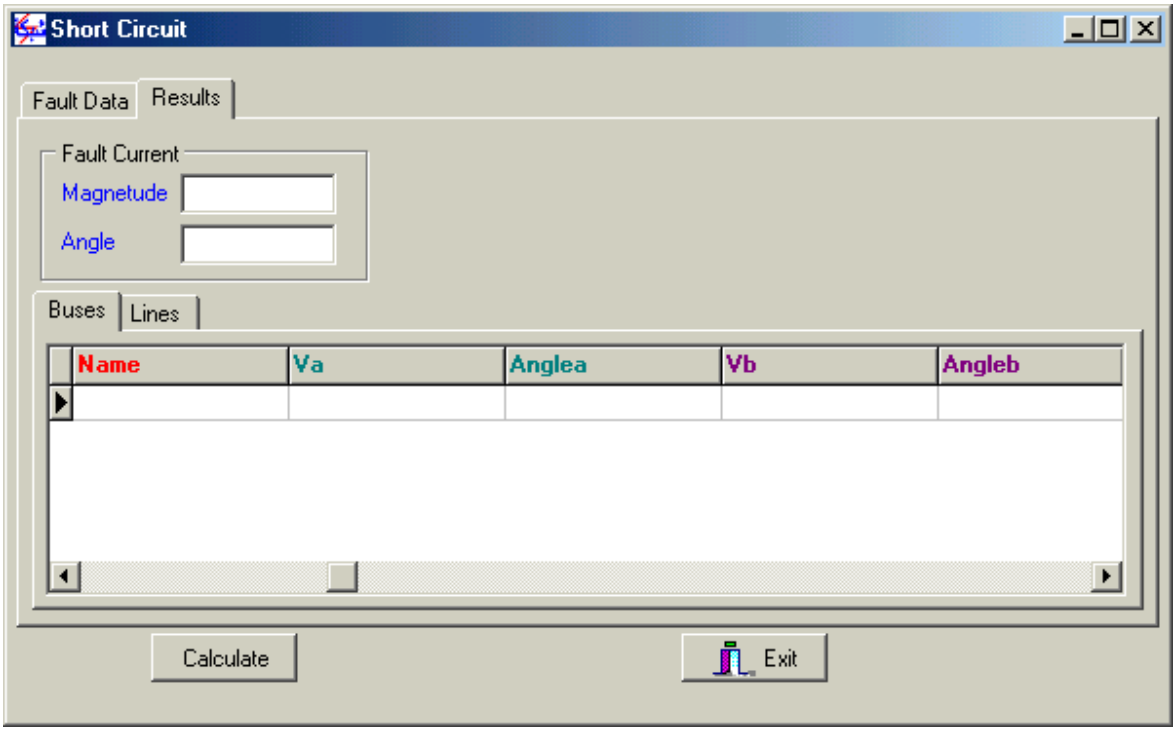


FIGURE 4.10 Présentation des résultats de court circuit : Buses

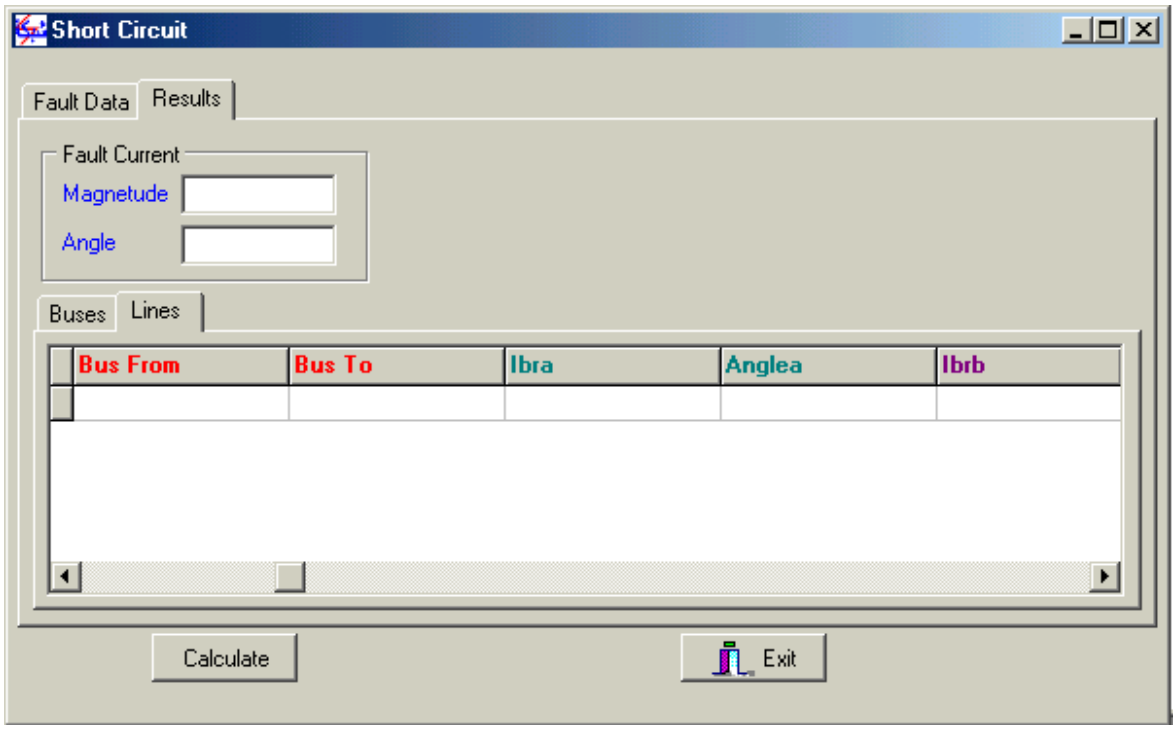


FIGURE 4.11 Présentation des résultats de court circuit : Lines

De même que pour l'application écoulement de puissances, les résultats de l'application court circuit sont affichés sous la forme présentée aux figures 4.10 et 4.11. La liste des jeux de barres du réseau électrique étudié est affichée avec les tensions de défaut y compris les angles de phases. La liste des éléments séries est affichée avec les courants de défaut de branches.

4.7 Conclusion

Les deux applications élaborées dans ce travail avec la MOO sont simples du point de vue étude et simulation des réseaux électriques. Cependant, pour ce type de projets, il faut d'abord commencer par les applications les plus simples afin de maîtriser la technique d'un coté, et d'un autre coté s'assurer que le modèle orienté objets développé fonctionne correctement (la version est stable) avant de rajouter de nouvelles applications.

Chapitre 5

Utilisation et Tests

5.1 Introduction

Dans ce chapitre, nous allons commencer par décrire les différentes étapes de l'utilisation du prototype développé pour la simulation d'un réseau électrique. Rappelons que les utilisateurs d'un logiciel de réseaux électriques ne sont pas nécessairement des experts dans les l'ordinateurs, la GUI a été alors conçue le plus simple possible.

5.2 Fonctionnement du prototype

Comme décrit dans les chapitres précédents, deux grandes abstractions sont identifiées dans le but de réaliser cet outil informatique de réseaux électriques avec la TOO : abstraction des éléments physiques du système et abstraction des applications. L'outil informatique qui englobe ces abstractions et contrôle les différentes entités qui existent est considéré également comme un objet. C'est l'objet du niveau le plus élevé dans le logiciel.

Un outil de ce type doit également contenir des objets supplémentaires (auxiliaires) qui permettent l'exécution des différents modules telle que la lecture des commandes et la gestion des fenêtres. Le diagramme d'activités simplifié de la figure 5.1 illustre le fonctionnement de l'outil.

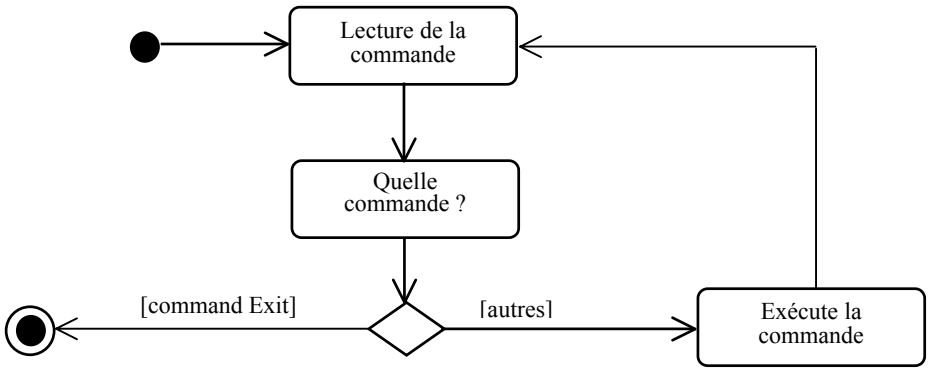


FIGURE 5.1 Diagramme d'activités simplifié de l'outil

Dans ce travail, le prototype de l'outil informatique développé pour la simulation des réseaux électriques est appelé OOShabaka. Il est représenté par une classe concrète OOShabaka créée au moment de l'exécution de l'outil. Cette classe est illustrée par la figure 5.2.

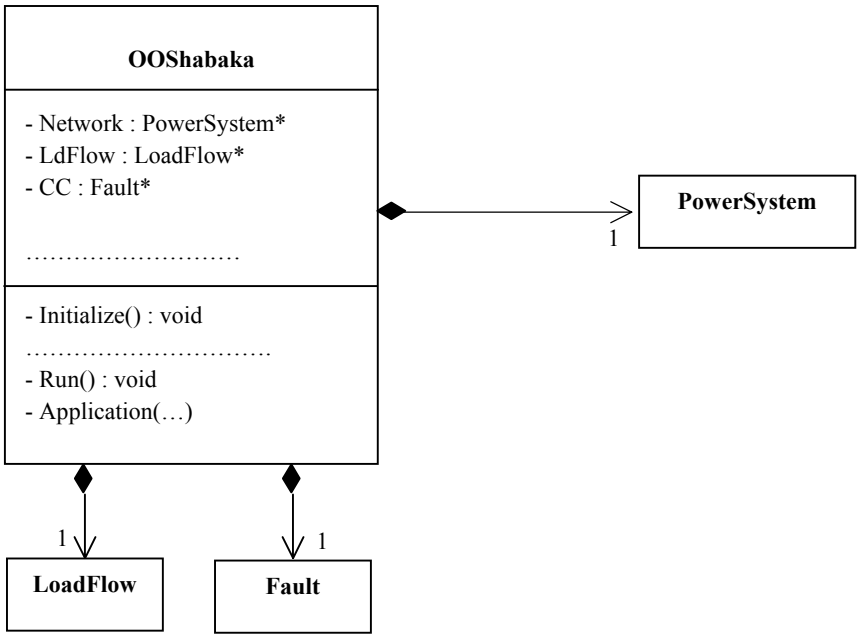


FIGURE 5.2 Classe OOShabaka

Les principaux attributs et méthodes de cette classe sont :

- Network : pointeur vers le réseau électrique ;
- LdFlow : pointeur vers l'application d'écoulement de puissances ;
- CC : pointeur vers l'application du calcul des courts circuits ;
- Initialize() : initialise l'état de l'outil ;
- Run() : contrôle l'exécution de l'outil ;
- Application(..) : crée l'application informatiquement dite et contrôle son exécution.

5.3 Utilisation du logiciel

L'utilisation du logiciel comporte les étapes principales suivantes :

- Conception graphique du schéma unifilaire du réseau à étudier par la création des objets physiques du réseau ;
- Spécifier les données des objets ;
- Choisir une application à exécuter et spécifier ses propres données ;
- Affichage des résultats dans la fenêtre d'exécution de l'application choisie.

5.3.1 Conception graphique d'un réseau électrique

Cette étape consiste à concevoir le schéma unifilaire du réseau à étudier. L'objet réseau électrique est créé à travers la création de ses objets constituants. Les objets jeux de barres qui représentent les nœuds du réseau électrique sont conçus en premier lieu, les objets lignes, charges, générateurs, etc. sont crées ensuite à condition qu'ils soient connectés à un jeu de barres pour les objets en dérivation et à deux jeux de barres pour les objets en série. Une fois créé dans l'éditeur graphique, l'objet est crée dans la base de données avec toutes ses caractéristiques graphiques et base de données.

L'utilisateur est en mesure de supprimer toutes les opérations de conception non désirées à condition que l'objet à supprimer soit déconnecté. Une fois supprimé dans l'éditeur graphique, il est détruit de la base de données et de tous les emplacements de liaison ou de connexion.

Il est également possible de modifier toutes les donnée saisies mise à part les données de liaison, par exemple le numéro d'un générateur est une données extraite de la base de données des jeux de barres, le numéro du générateur représente le numéro du jeux de barres de connexion, donc cette donnée ne peut pas être modifiée dans la fenêtre de dialogue du générateur mais dans la fenêtre du jeu de barres correspondant. Un simple clic droit fait apparaître la fenêtre de dialogue de n'importe quel composant. De même pour toutes les données de liaison.

Pour l'opération de déplacement ou glissement, elle a été réalisée uniquement pour l'objet jeu de barres parce que c'est le seul élément libre (conçu sans qu'il soit connecté). Quand l'utilisateur sélectionne un jeu de barres pour le déplacer, toutes ses caractéristiques graphiques sont modifiées dans l'éditeur graphique et dans la base de données.
La figure 5.3 représente un simple réseau de six jeux de barres conçu par le logiciel développé.

5.3.2 Spécifier les données

Chaque objet réseau crée possède une liste de jeux de barres, une liste de lignes, une listes de transformateurs, une liste de générateurs, une liste de charges et une liste de compensateurs. Ces listes sont construites par une simple liaison à la base de données.

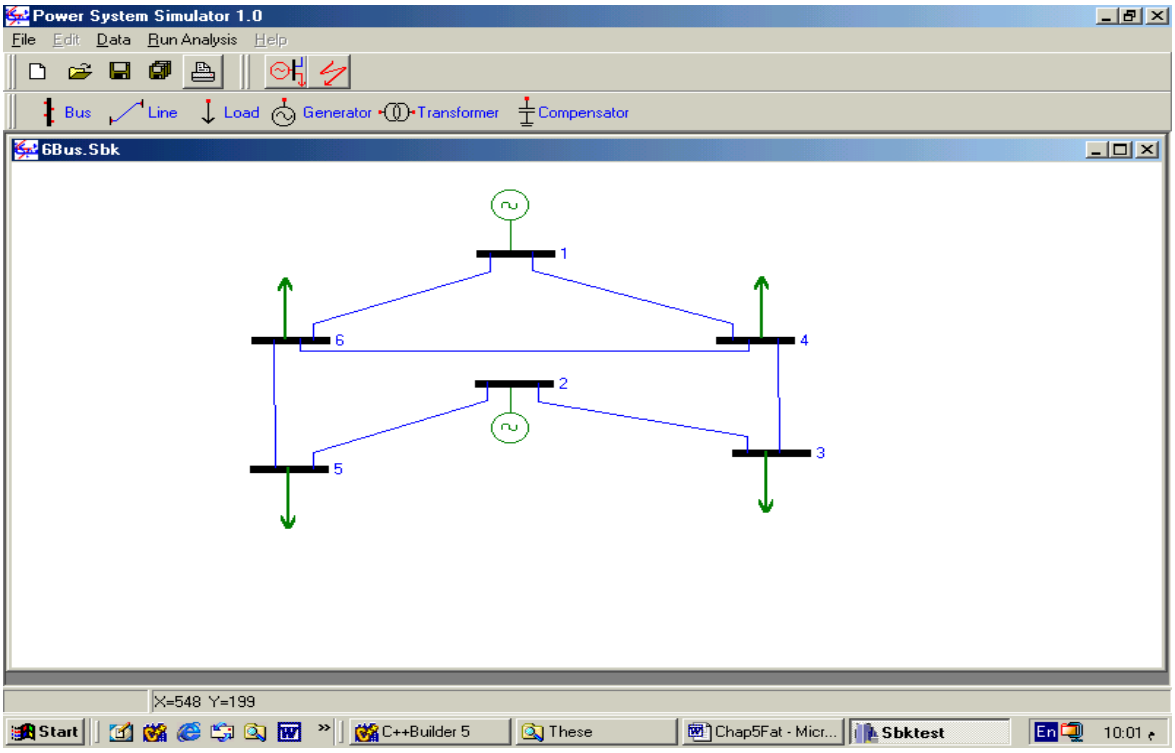


FIGURE 5.3 Réseau 6 jeux de barres

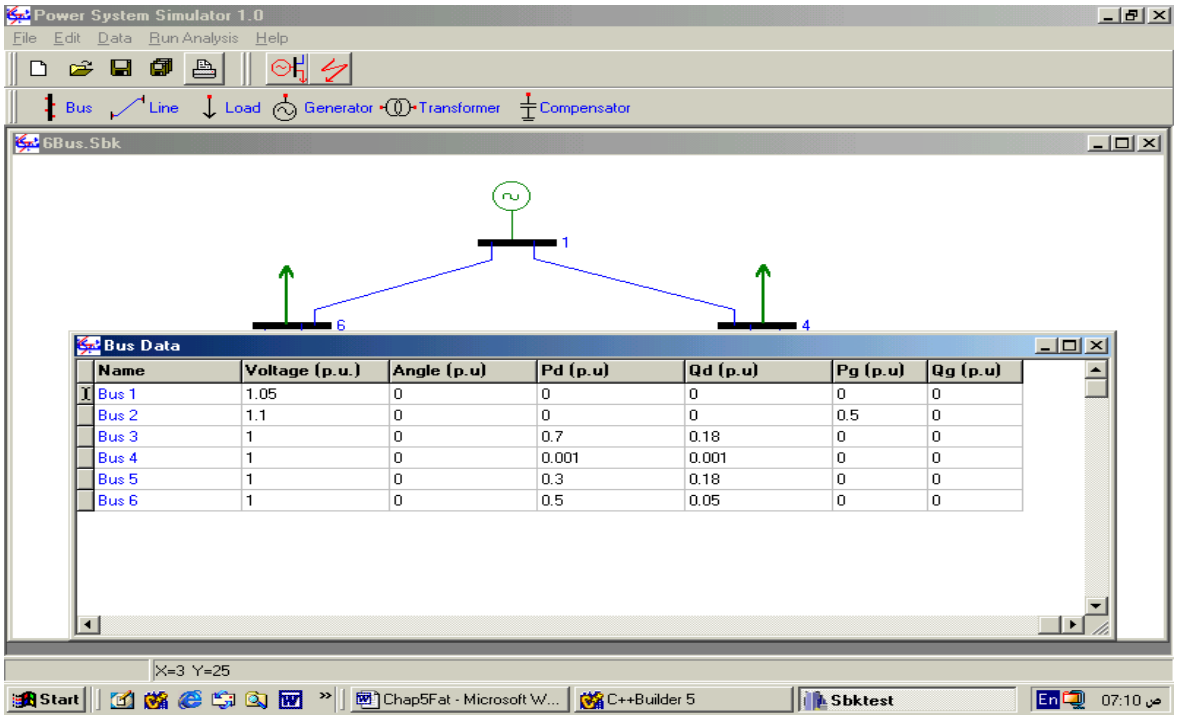


FIGURE 5.4 Liste des jeux de barres

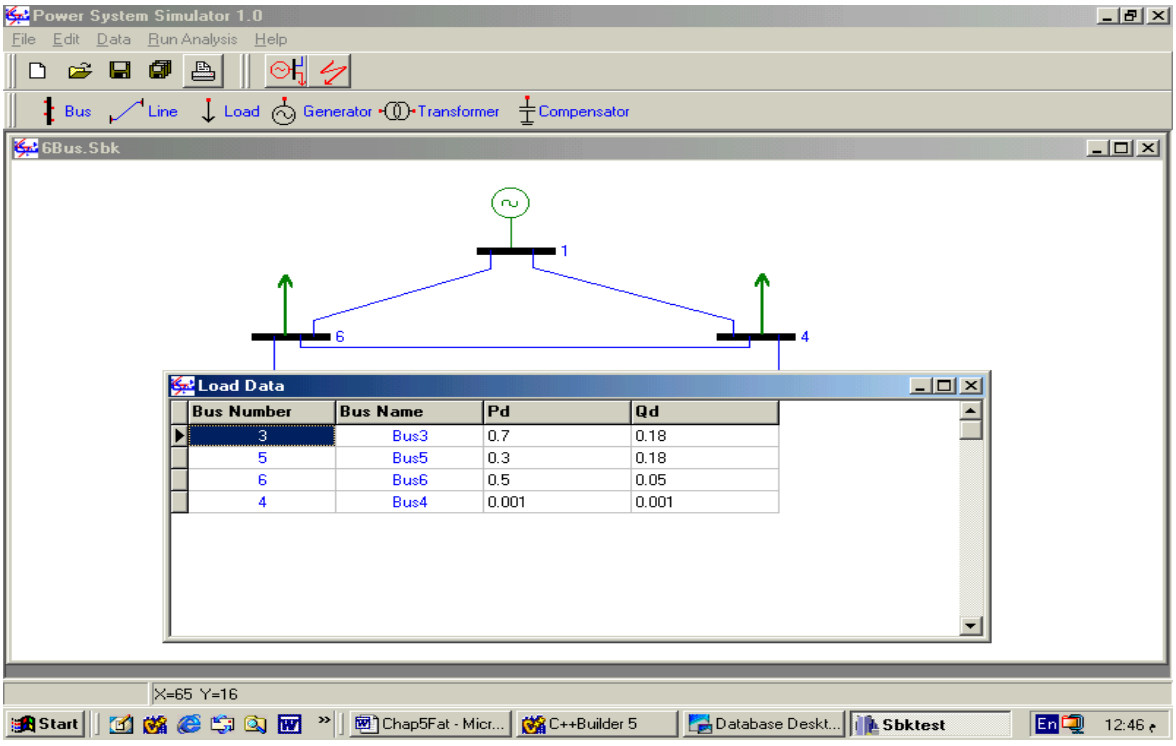


FIGURE 5.5 Liste des Charges

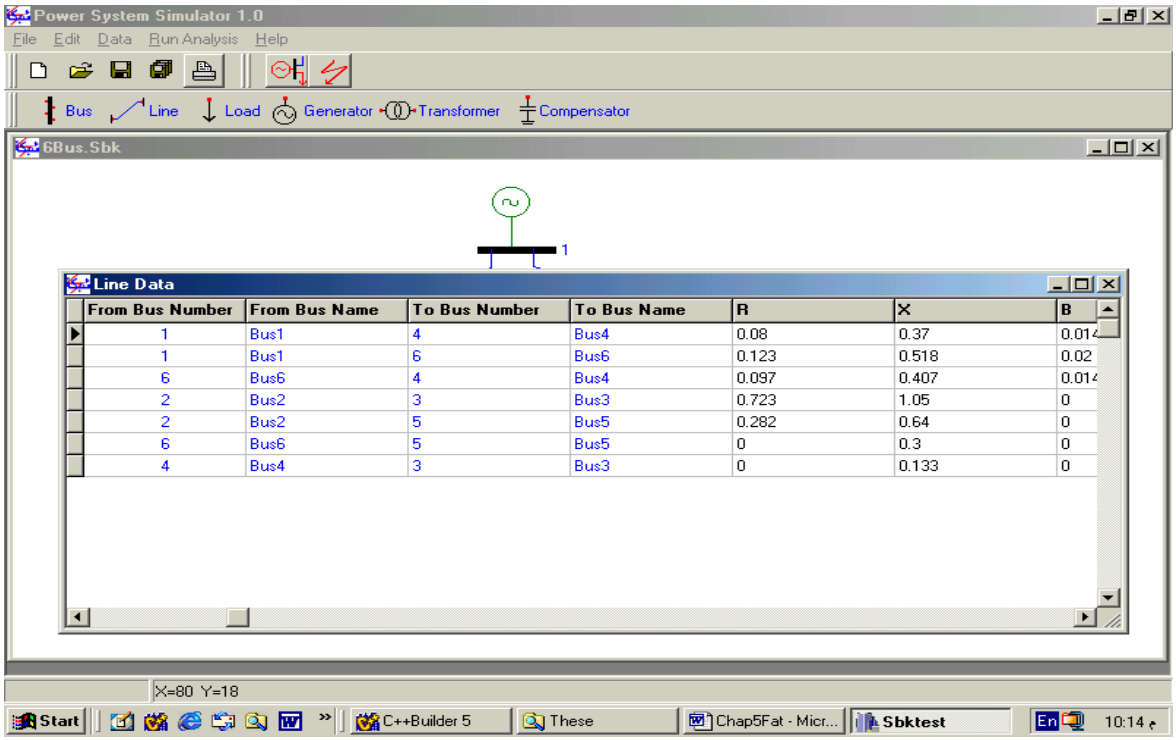


FIGURE 5.6 Liste des Lignes

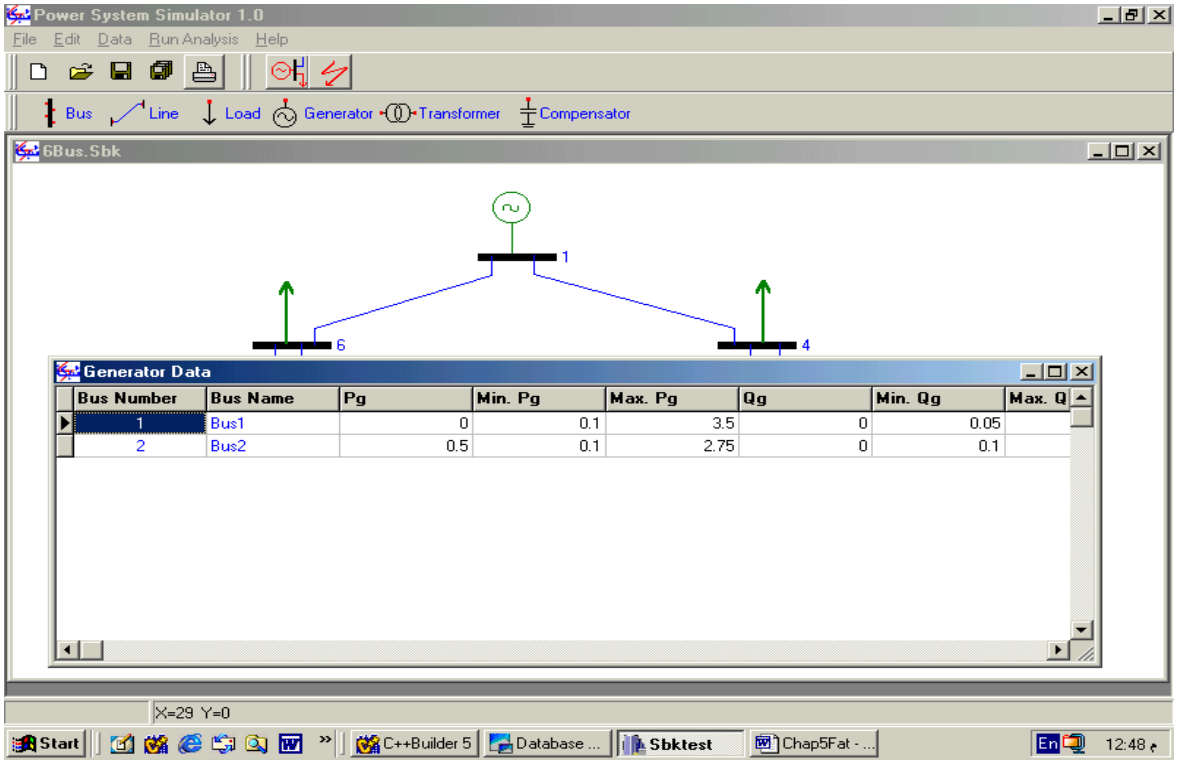


FIGURE 5.7 Liste des générateurs

En général, les données introduites dans les phases précédentes sont utilisées d'une manière ou d'une autre par les deux applications écoulement de puissances et le calcul des courants de court circuit. En choisissant une application précise à exécuter, des données supplémentaires sont nécessaires.

5.3.3 Exécution de l'application écoulement de puissances

Pour l'exécution de cette application, une fenêtre de simulation est générée (figure 5.8), sur la même fenêtre les résultats sont visualisés. Le simulateur attend les entrées de l'utilisateur, une fois ces données sont spécifiées, les résultats sont directement affichés (figures 5.9 et 5.10)

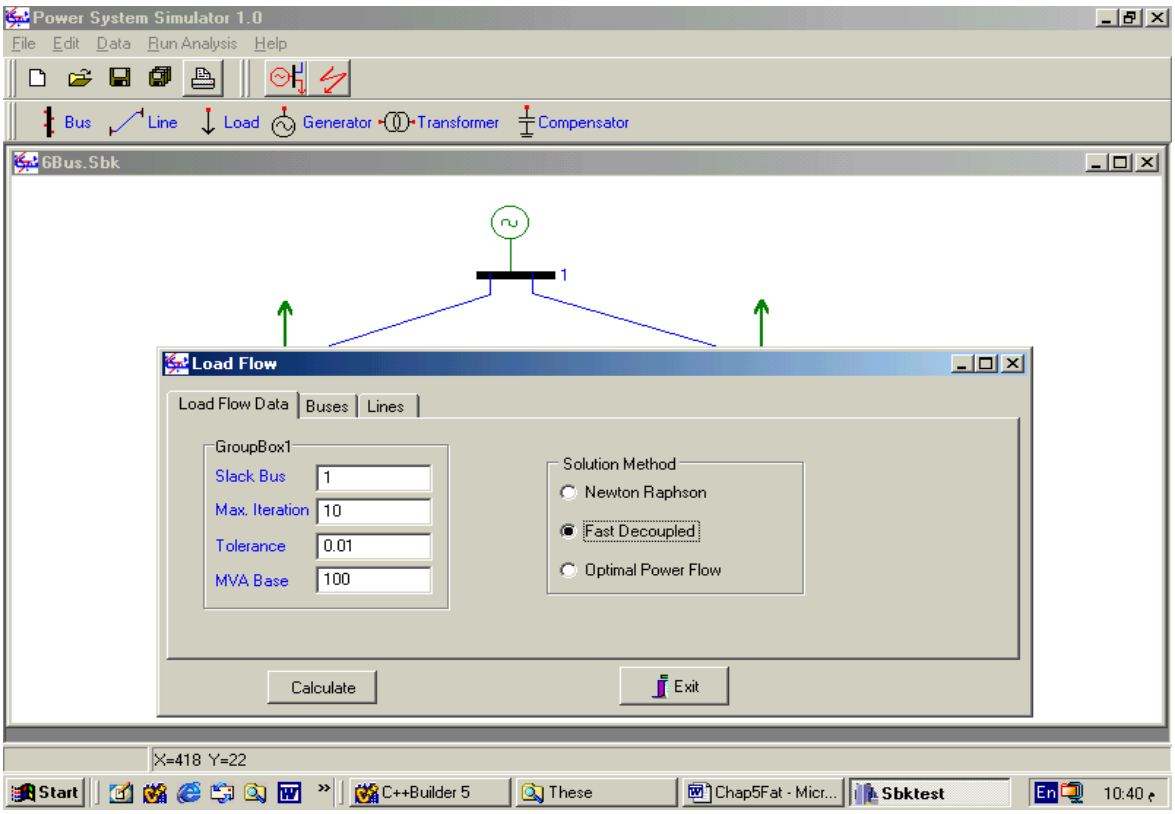


FIGURE 5.8 Exécution de l'application Ecoulement de puissances

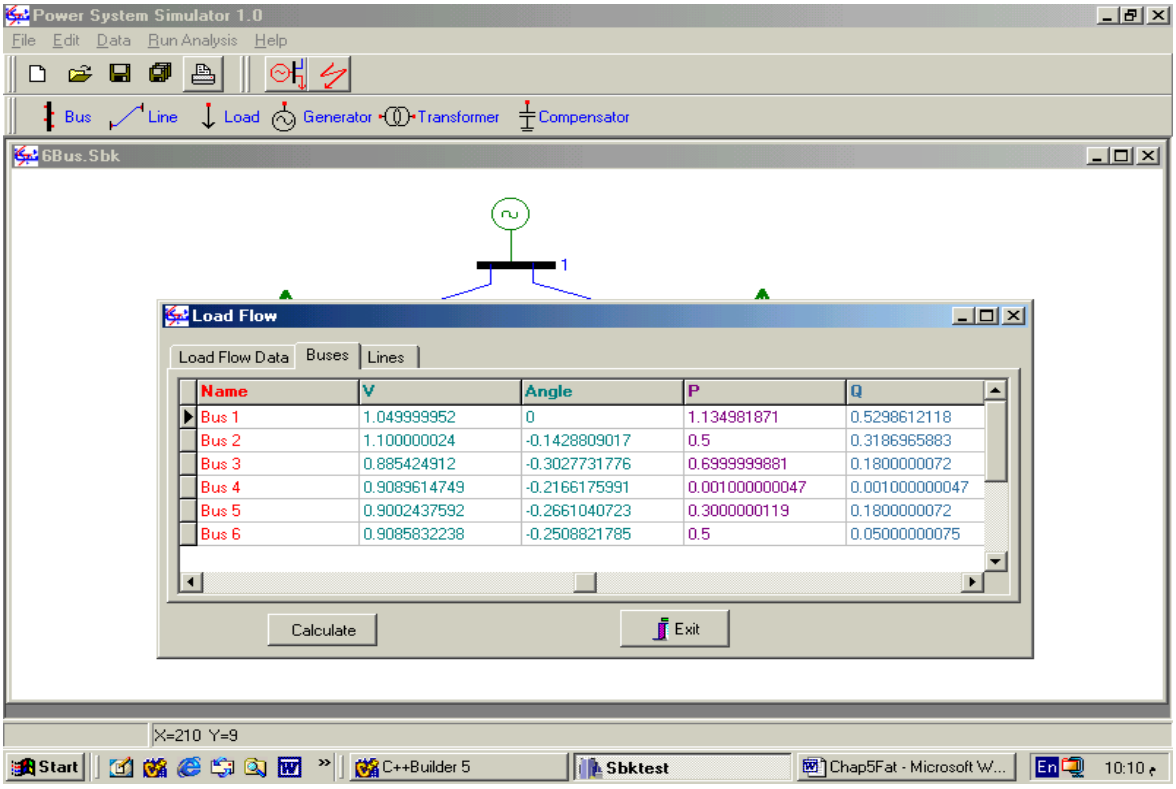


FIGURE 5.9 Résultats de l'écoulement de puissances : Buses

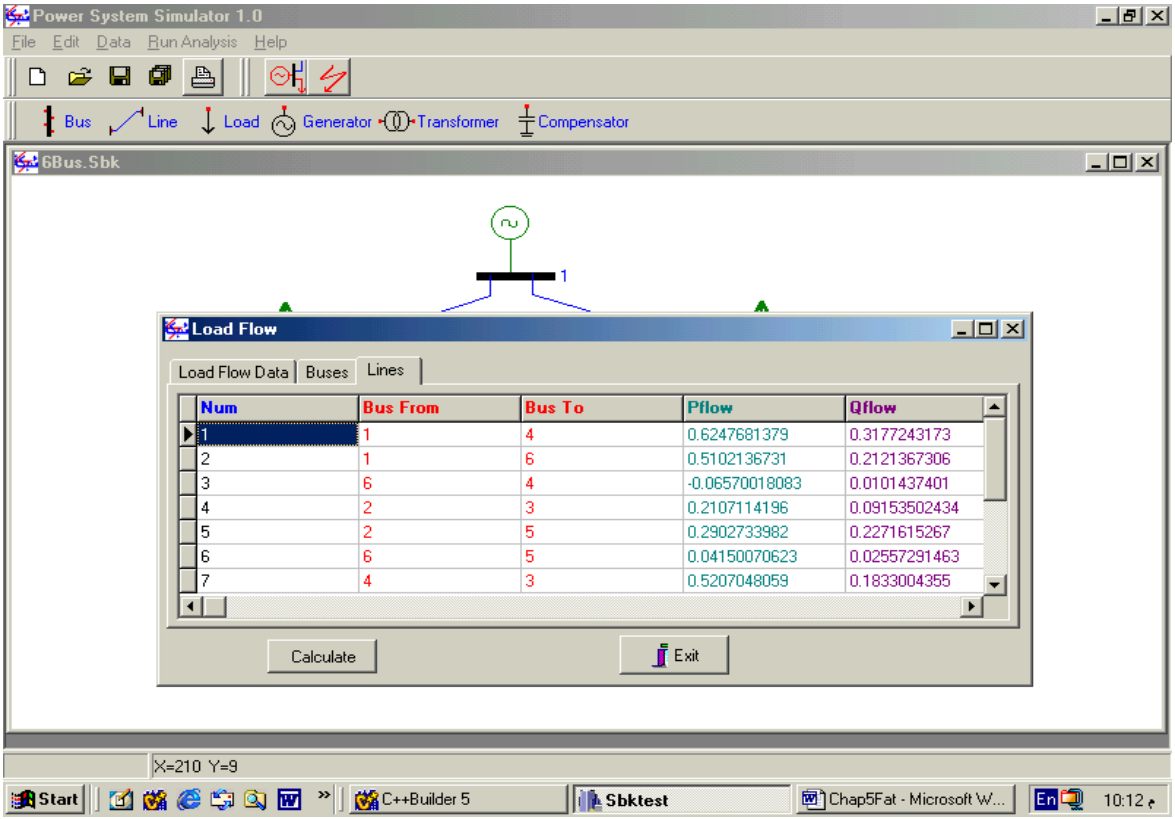


FIGURE 5.10 Résultats de l'écoulement de puissances : Lines

5.3.4 Exécution de l'application calcul des courants de court circuit

De la même manière que l'application précédente, l'exécution de cette application engendre une fenêtre de simulation (figure 5.11), sur la même fenêtre les résultats sont visualisés une fois les données propres de l'application courants de court circuit sont spécifiées et l'application est exécutée (figures 5.12 et 5.13).

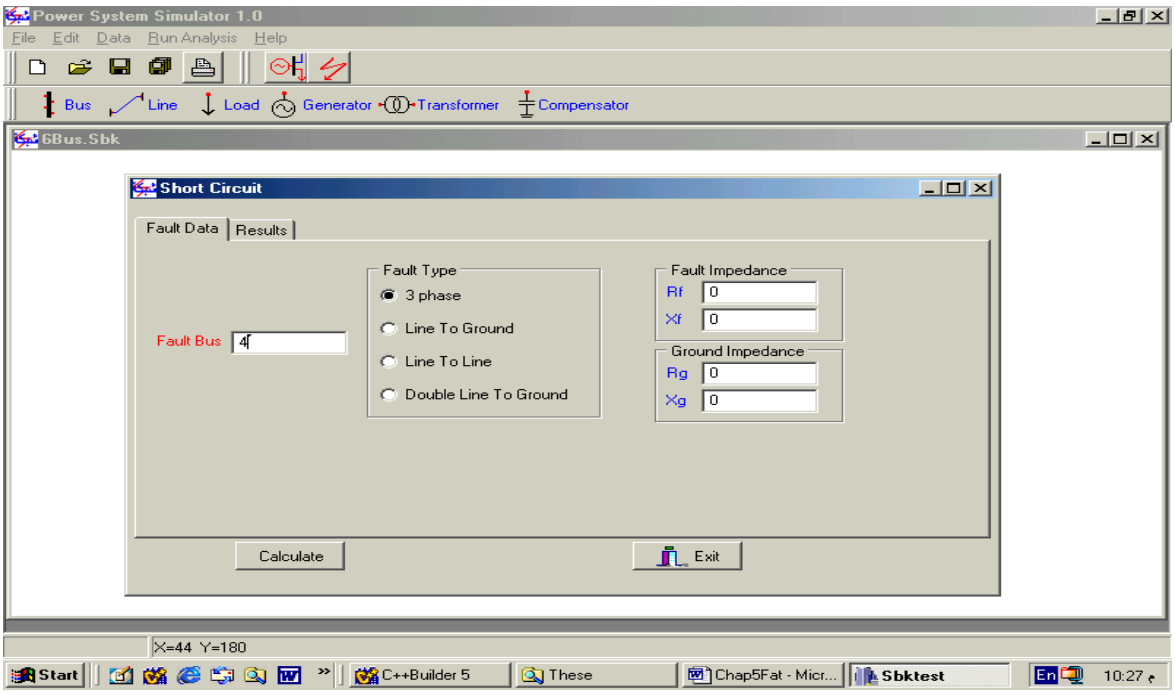


FIGURE 5.11 Exécution de l'application courants de court circuit

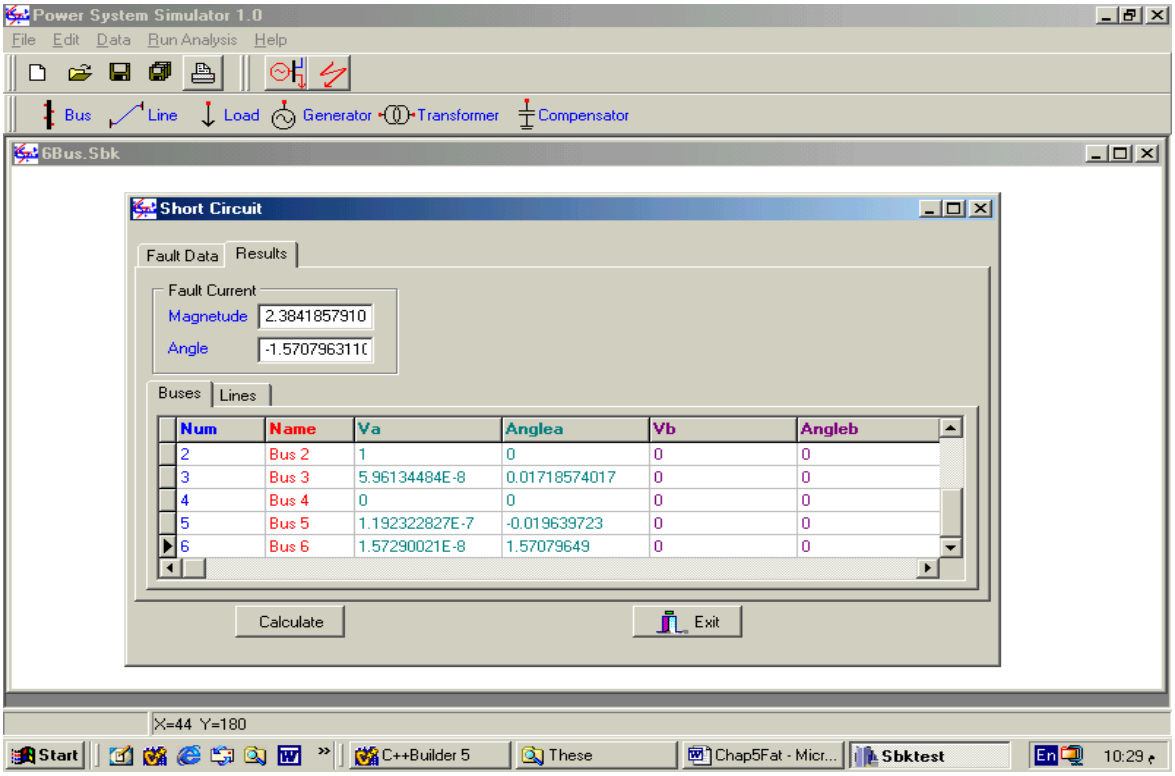


FIGURE 5.12 Résultats du calcul des courants de court circuit : Buses

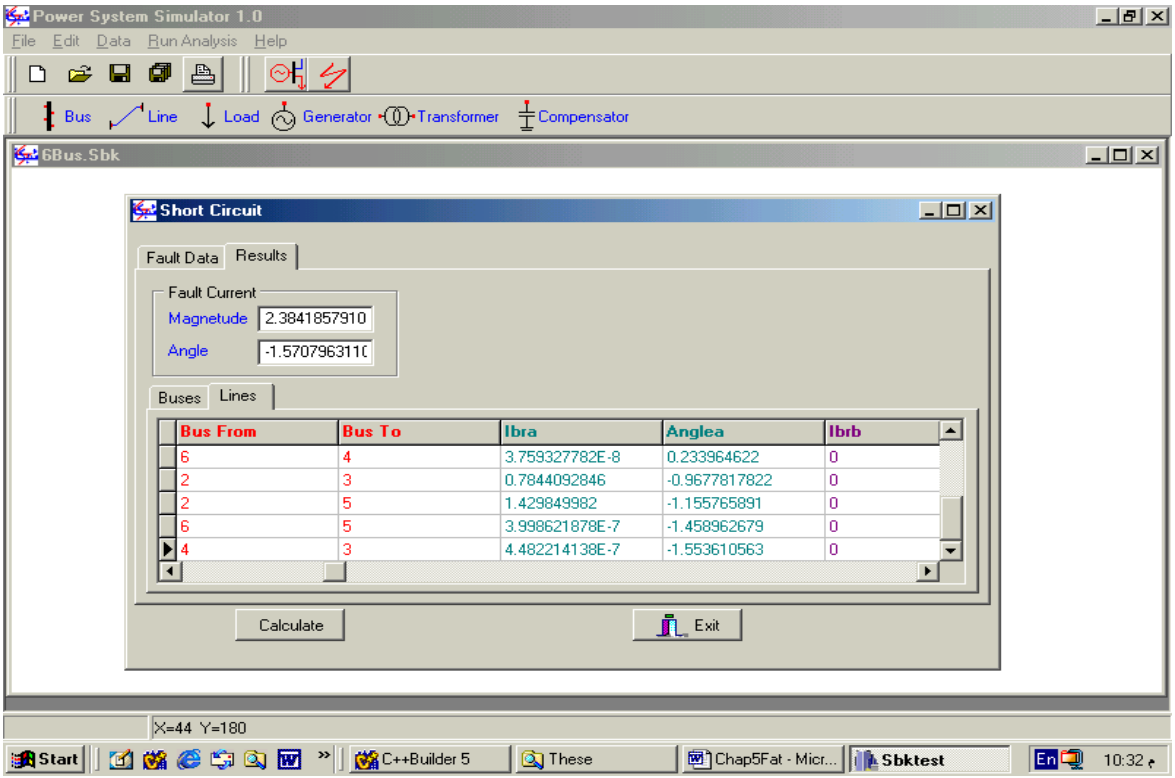


FIGURE 5.13 Résultats du calcul des courants de court circuit : Lines

5.4 Conclusion

Dans ce travail, l'objectif n'est pas l'étude de l'écoulement de puissances et le calcul des courants de court circuit c'est pourquoi nous n'avons pas discuter les résultats de ces deux applications, cependant nous avons effectué une comparaison avec les travaux précédents traitant ces deux problèmes pour s'assurer de l'exactitude de leurs implémentations.

Dans l'état actuel, le prototype que nous avons développé a été testé sur des exemples de réseaux de petites tailles, donc en terme de temps de calcul et efficacité, des réseaux de grandes tailles doivent être testé dans le futur.

Conclusion et Perspectives

Nous avons abordé dans cette thèse, le développement d'une plate forme logicielle pour la simulation des réseaux électriques en utilisant la technique orientée objets. Nous avons présenté les premiers composants élémentaires ayant servi à son développement.

La MOO est différente des modélisations traditionnelles d'analyse, de conception et de programmation. Mais cela ne signifie pas que tous les principes éprouvés dans les anciennes méthodes sont rejetés. Dans ce présent travail, la MOO des réseaux électriques réalisée, aussi modeste qu'elle soit et ayant encore loin d'être globale pour satisfaire toutes les exigences des études de réseaux électriques, elle a été séduisante en permettant une représentation réelle et naturelle de la structure physique du réseau électrique, ses éléments et ses fonctions d'analyse.

Deux grandes structures de classes sont développées : la structure des éléments physiques du réseau électrique (jeux de barres, lignes de transmission, générateurs,..) et la structure des méthodes d'analyse (applications). Les classes créées sont simples et pourront être utilisées et réutilisées ce qui va permettre de se concentrer sur les spécificités de chaque application. Les deux applications intégrées dans ce logiciel sont l'écoulement de puissances et le calcul des courants de court circuit.

La construction de l' interface usager graphique GUI est une partie importante de la plate forme logicielle développée. Cependant, dans la description de ce travail et ses classes, on a mis l'accent sur tout ce qui est électrique et on a survolé tout ce qui est graphique (plus difficile) ou purement informatique pour mieux présenter la TOO et ses avantages dans le domaine Electrotechnique en général.

1. Contributions

Développement d'une plate forme logicielle pour les réseaux électriques

Une plate forme logicielle est développée pour la simulation des réseaux électriques, elle servira de noyau pour des futurs développements dans la portée de ce projet. Elle se compose essentiellement d'un éditeur graphique, d'une base de données et de deux applications (écoulement de puissances et calcul des courants de court circuit). L'utilisateur est en mesure de concevoir graphiquement son réseau d'étude, spécifier ses données et lancer l'une des applications disponibles.

Modélisation et implémentation orientée objets

La MOO a été utilisée pour la représentation des éléments physiques ainsi que les deux applications considérés. La classification des éléments physiques a été basée sur le nombre de connections aux différents jeux de barres que chaque élément présente (c'est la classification de la majorité des travaux publiés). L'implémentation de chaque élément est basée sur son modèle algébrique. En premier lieu, trois classes de base sont développées : la classe des jeux de barres, la classe des éléments en série et la classe des éléments en dérivation. Par la suite, suivant le type d'élément sa classe est conçue par héritage de sa classe de base.

Les classes des deux applications sont conçues par héritage de la classe du réseau électrique. Cette dernière est une agrégation de tous les éléments physiques.

Utilisation du langage UML

La notation graphique est l'un des aspect du développement orienté objets, c'est pourquoi on a essayé dans la mesure du possible de représenter la MOO et l'implémentation du présent travail par le langage standard UML. Cette notation permettra de garder trace de ce qui a été réalisé et une compréhension rapide du code facilitant ainsi les développements futurs.

2. Perspectives et Suggestions

Les perspectives ouvertes par ce travail sont nombreuses et se situent dans chaque partie de ce projet du fait que la première version d'un système informatique représente une partie mineure du produit final visé. A chaque nouvelle version, on effectue les mêmes activités de développement mais les données d'entrée décrivent des besoins de modification et de perfection.

Au niveau de la plate forme logicielle, le plus grand n'est pas encore fait, citons entre autres :

- La GUI est loin d'être complète, elle a besoin d'être enrichie par de nouveaux composants graphiques, des utilitaires tels que copier, coller, agrandir, choix des couleurs et des dimensions des composants graphiques, affichage des résultats au niveau de chaque composant sur le schéma unifilaire et autres ;
- La Base de données doit être orientée objets également et mérite d'être mieux organisées et mieux adaptée à la GUI ;
- Il vaut mieux développer la partie facilitées de calcul indépendamment des autres parties par principe de la MOO (matrices creuses, matrices inverses, opérations sur les matrices et les vecteurs, résolution des systèmes présents dans les analyses réseaux électriques,..).

Au niveau de la MOO des réseaux électriques :

- Les classes réalisées ont besoin d'être enrichies et perfectionnées ;
- De nouvelles classes doivent être développées pour représenter le reste des éléments physiques présent dans un réseau électrique ;
- Généraliser les classes des applications développées dans ce travail et introduire de nouvelles applications telles que la stabilité transitoire, l'écoulement de puissances optimales, la compensation d'énergie réactive et autres ;
- Lier les différentes applications ;
- Dans ce travail, la notation graphique par UML n'a pas été détaillée, il vaut mieux dégager le reste des diagrammes.

Bibliographie

[1] E.Z. Zhou, " Object Oriented Programming, C++ and Power System Simulation ", IEEE Trans. on Power Systems, Vol. 11, N°1, pp. 206-215, February 1996.

[2] Manzoni, A.S. Silva, I.C. Decker, "Power Systems Dynamics Simulation Using Object Oriented Programming" IEEE Trans. on Power Systems, NewYork, Vol. 14, N° 1, pp. 249-255, February 1999.

[3] M.N. Agostini, I.C. Decker, A.S. Silva, , "A New Approach for the Design of Electric Power System Software Using Object Oriented Modeling" CBA pp. 1555-1561, 2002 (en Portuguese).

[4] J. Zhu, D. L. Lubkeman, " Object Oriented Development of Software System for Power System Simulations ", IEEE Trans. on Power Systems, Vol. 12, N°2, pp. 1002-1007, May 1997.

[5] A.F. Neyer, F.F. Wu, K. Imhof, " Object Oriented Programming for Flexible Software: Example of a Load Flow ", IEEE Trans. on Power Systems, Vol. 5, N°3, pp. 689-696, August 1990.

[6] S.A. Soman, S.A. Khaparde, S. Pandit, "Computational Methods for Large Sparse Power System Analysis, An Object Oriented Approach", Kluwer Academic Publishers, London 2002.

[7] S. Pandit, S.A. Soman, S.A. Khaparde, "Object Oriented Design for Power System Applications", IEEE Computer Applications in Power, pp.43-47, October 2000.

[8] P.H. Schavemaker, R. Reijntjes, L.V. Sluis, " Power System Demos: A Graphical Aid for Lecturing and Training Purposes ", IEEE Trans. on Power Systems, Vol. 16, N°4, pp. 581-586, November 2001.

[9] M.T. Tsay, S.Y. Chan, " A Personal Computer Environment for Industrial Distribution System Education, Design, and Analysis ", IEEE Trans. on Power Systems, Vol. 15, N°2, pp. 472-476, May 2000.

[10] M. Foley, A. Bose, W. Mitchell, "An Object Based Graphical User Interface", IEEE Trans. on Power Systems, Vol. 8, N°1, pp. 97-104, February 1993.

[11] J.R. Shim, W.H. Lee, D.H. Im, " A Windows Based Interactive and Graphic Package for the Education and Training of Power System Analysis and Operation ", IEEE Trans. on Power Systems, Vol. 14, N°4, pp. 1193-1199, November 1999.

[12] David R.C. Hill, "Analyse Orientée Objets et Modélisation par Simulation", Addison Wesley, France 1993.

[13] Grady Booch, “Analyse et Conception Orientée Objets”, Addison Wesley, France 1994.

[14] Ivar Jacobson, “Le Génie Logiciel Orienté Objets”, Addison Wesley, France 1993.

[15] G. Booch, J. Rumbaught, I. Jacobson, “The Unified Modeling Language Version 1.0”, Rational Software Corporation 1997.

[16] E. Gamma, R. Helm, R. Johnson, J. Vlisides, “Design Patterns: Elements of Reusable Object Oriented Software”, Addison Wesley, France 1995.

[17] J. Zhu, P. Jossman, “ Application of Design Patterns for Object Oriented Modeling of Power Systems “, IEEE Trans. on Power Systems, Vol. 14, N°2, pp. 532-537, May 1999.

[18] CR Fuerte Esquivel, E. Acha, SG Tan, J.J. Rico, “ Efficient Object Oriented Power Systems Software for the Analysis of Large Scale Networks Containing FACTS-Controlled Branches “, IEEE Trans. on Power Systems, Vol. 13, N°2, pp. 464-472, May 1998.

[19] Z. L. Gaing, C. N. Lu, B. S. Chang and C. L. Cheng, “ An Object Oriented Approach For Implementing Power System Restoration Package “, IEEE Trans. on Power Systems, Vol. 11, N°1, pp. 483-489, February 1996.

[20] M.R. Sigari, D. Kirschen. S. Silverman, “ Evaluating A Restoration Tool Using Consolidated Edison’s Training Simulator “, IEEE Trans. on Power Systems, Vol. 11, N°3, pp. 1636-1642, August 1996.

[21] E.Z. Zhou, “ Design of Generic Direct Sparse Linear System Solver in C++ for Power System Analysis “, IEEE Trans. on Power Systems, Vol. 16, N°4, pp. 206-215, November 2001.

[22] B.A. Meyers, “ User Interface Software Tools “, ACM Trans. on Computer-Human Interaction, Vol. 2, N°1, pp. 64-103, Mars 1995.

[23] S. Islam, N. Chowdhury, “ A Case Based Windows Graphic Package for the Education and Training of Power System Restoration “, IEEE Trans. on Power Systems, Vol. 16, N°2, pp. 181-187, May 2001.

[24] R.B. Palma, F.G. Flatow, N.S. Oyarce, “ Object Oriented Platform for Integrated Analysis of Energy and Transportation Network “, IEEE Trans. on Power Systems, Vol. 18, N°3, pp. 1062-1069, August 2003.

[25] Borland C++ Builder, “Guide du développeur”, 2002
http://www.borland/C++

[26] John Hubard, “ Programmation en C++ “, Mc Graw Hill
London 1997.

[27] M. Belkacemi, L. Benfarhi, “A Software Tool for Identification of Power System Networks”, UPEC’98 Conference, September 1998, Edinburg University, UK.

[28] M. Belkacemi, L. Benfarhi, “An Object Oriented User Interface Power System Simulation”, UPEC’99 Conference, September 1999, Leicester University, UK.

[29] L. Benfarhi, M. Belkacemi, L. Mokhnache, “An Object Oriented Tool for Power System Simulation”, Proceeding of the 5th International Conférence on Quality, Reliability, and Maintenance, QRM 2004, University of Oxford, UK, 1st-2nd April 2004, Professional Engineering Publishing Limited.

[30] L. Benfarhi, M. Belkacemi, A. Tolba, “Object Oriented Sparse Matrix Computation for Power System Simulation”, Archives of Electrical Engineering, vol. LIII, N° 210, pp. 369-384, April 2004.

[31] Hakavik and A. T. Holen, “Power System Modeling and Sparse Matrix Operations Using Object Oriented Programming”, IEEE Trans. On Power Systems, vol. 9, N° 2, pp. 1045-1052, 1994.

[32] W.F. Tinney, I.W. Walker, “Direct Solution of Sparse Network Equations by Optimally Ordered Triangular Factorization”, Proc. Of the IEEE, vol.55, pp. 1801-1809, November1967.

[33] M. Belkacemi, N. Aloui, L. Benfarhi, M. Ghezaili, “Sparsity Techniques Application in Power System Short Circuit Computation”, 3rd Regional CIGRE Conference, 25-27 May 1999, Doha, Qatar.

[34] T. Bouktir, M. Belkacemi, L. Benfarhi, A. Gherbi, “Object Oriented Optimal Power Flow”, UPEC’2000 Conference, UK.

[35] T. Bouktir, A. Gherbi, L. Benfarhi, M. Belkacemi, “An Efficient Object Oriented Load Flow Applied to a Large Scale Power System: Application to Sonelgaz Network” ,Conference ICEL 2000, 13-15 Novembre 2000, USTO, Oran.