الجمهوريـــــة الجزائريـــــة الديمقراطيـــة الشعبيـــة
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE MOHAMMED KHIDER - BISKRA -
FACULTE DES SCIENCES ET DES SCIENCES DE L'INGENIEUR
DEPARTEMENT GINIE ELECTRIQUES

Mémoire de magister
En Electrotechnique

Option : Electricité industrielle

Thème

Elaboration d'un Générateur de Données pour les Outils d'Evaluation de la Fiabilité des Réseaux Electrique

Présenté par

NANI youcef
Ingénieur en Electrotechnique, Option : Réseau Electrique

Devant le Jury composé de :

Président	:	MOUSSI Ammar	Pr	Univ. Biskra
Rapporteur	:	ABOUBOU Abdennacer	M. C	Univ. Biskra
Examinateur	:	SRAIRI Kamel	Pr	Univ. Biskra
Examinateur	:	MIMOUNESOURI Mohamed	Pr	Univ. Biskra
Examinateur	:	BENSALEM Ahmed	M. C	Univ. Batna

1. Motivation

Dans le domaine de réseaux électriques, à cause de l'alimentation 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 Technique Orientée Objet « TOO » pour développer les outils informatiques pouvant satisfaire ces nouveaux besoins.

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. Dans beaucoup de cas, le changement d'une partie spécifique du code produit de grands effets sur d'autres routines du programme 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 réécriture de grandes parties de codes.

En dépit du succès et des caractéristiques 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 structures 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.

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.

2. 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.

3. Structure du mémoire

- ✓ 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.
- ✓ Le Chapitre 2 ; décrit les aspects généraux de l'application de la Modélisation Orientée Objet « MOO » pour la conception du logiciel. Ce chapitre présente les grandes abstractions des réseaux électriques adaptées.
- ✓ Le Chapitre 3 ; présente notre algorithme d'analyse et calcul de la fiabilité, notons que la fiabilité n'est qu'un exemple pour application.
- ✓ Le Chapitre 4 ; Contient les détails sur le principe de fonctionnement de notre logiciel ainsi que les tests et validations nécessaires, sur des réseaux radiaux réels de distribution, appartenant à la Société Nationale d'Electricité et de Gaz (SONELGAZ).
- ✓ Enfin, la conclusion qui résume brièvement ce travail et les contributions majeures ainsi qu'une discussion des travaux futurs conclut la présentation.

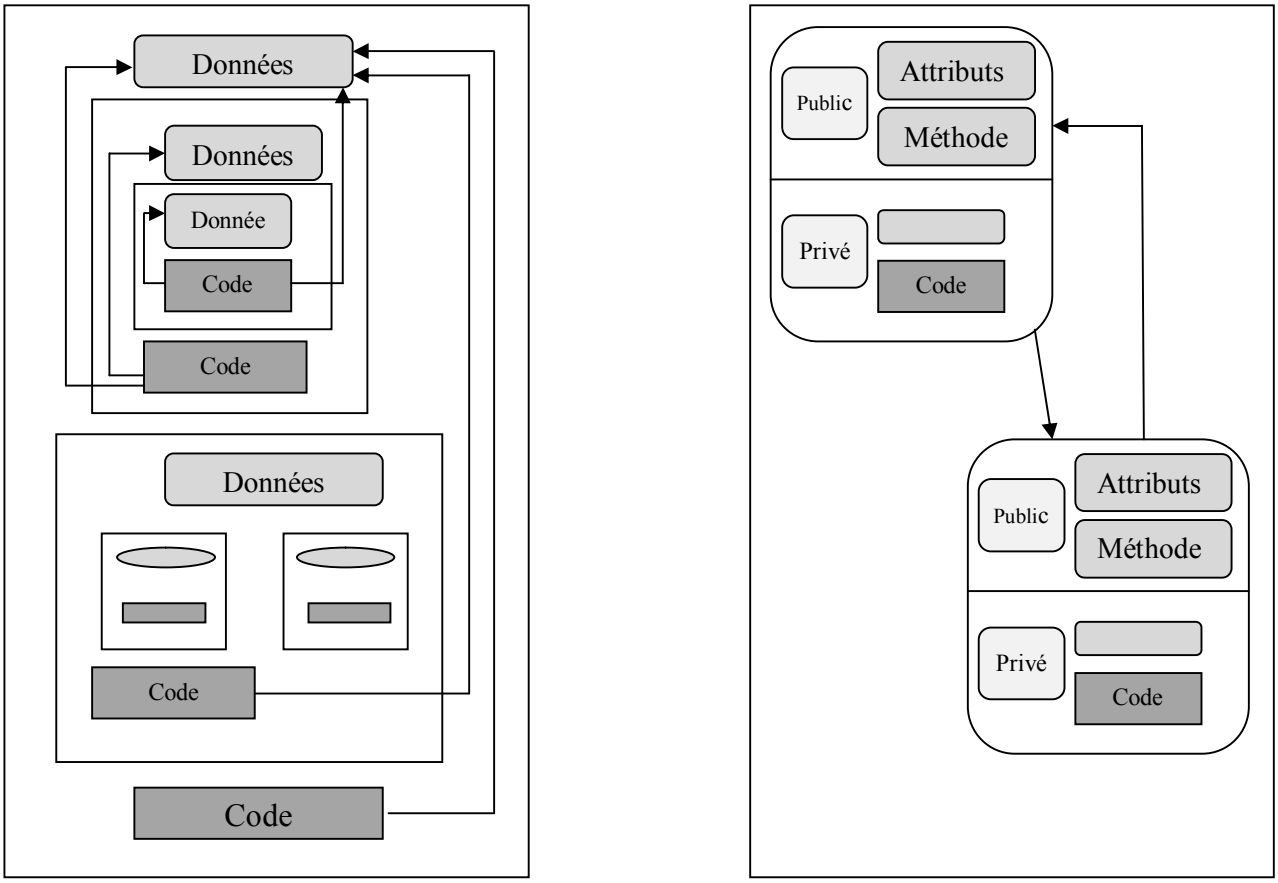
I.1 Introduction [1]

La programmation classique ou procédural telle que le débutant peut la connaitre à travers des langages de programmation comme Pascal, C etc.… traite les programmes comme un ensemble de données sur les quelles agissent des procédures. Les procédures sont les éléments actifs et importants, le donnés devenant des éléments passifs qui traversent l'arborescence de programmation procédurale en tant flot d'information.

Cette manière de concevoir les programmes reste proche des machines de Von Neuman et consiste en dernier ressort à traiter indépendamment les données et les algorithmes (traduit par des procédures) sans tenir compte des relations qui les lient.

En introduisant la notion de modularité dans la programmation structurée descendante, l'approche diffère légèrement de l'approche habituelle de la programmation algorithmique classique. Nous avons défini des machines abstraites qui ont une autonomie relative et qui possèdent leurs propres structures de données, la conception d'un programme relative dés lors essentiellement de la description des interactions que ces machines ont entre elles [1].

La programmation orientée objet relève d'une conception ascendante définie comme des « **messages** » échangés par des entités de base appelées objets.



Programmation procédure **Programmation objet**

Figure I.1 Comparaison des deux topologies de programmation

Les langages objets sont fondés sur la connaissance d'une seule catégorie d'entité informatique : l'objet.

Dans un objet, traditionnellement ce sont les données qui deviennent prépondérantes. On se pose d'abord la question : « de quoi parle-t-on ? » et non pas la question « que veut-on faire ? », comme en programmation algorithmique. C'est en ce sens que les machines abstraites de la programmation structurée modulaire peuvent être considérées comme pré-objets.

En fait la notion TAD (type abstrait de données) est utilisée dans cet ouvrage comme spécification d'un objet, en ce sens nous nous préoccupons essentiellement des services offerts par un objet indépendamment de sa structure interne.

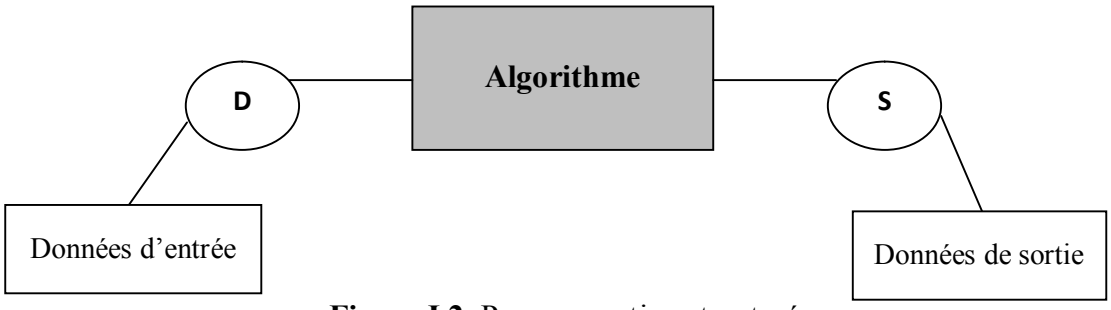


Figure I.2 Programmation structurée

I.2 Eléments de l'approche orientée objets

I.2.1 Objets

I.2.1.1 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 [2] :

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 [2] :

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**

I.2.1.2 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 [2] :

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 propriété

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.

I.2.1.3 Comportement d'un objet

Les objets n'existent pas isolément mais interagissent plutôt avec d'autres objets en démontrent un comportement donné. Cette constatation conduit à la définition suivante [2] :

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 [2] :

- 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.

I.2.1.4 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 [2] :

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.

I.2.1.5 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) ;
- **serveur :** objet qui ne peut jamais agir sur les autres objets mais sur qui les autres objets peuvent agir (il est la cible de messages) ;
- **agent :** 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.

I.2.2 Classes

I.2.2.1 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 [2] :

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 [3,2] :

- **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.

I.2.2.2 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 [3,2].

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-classe**. 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.

I.2.3 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 [2].

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).

I.2.4 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 [2].

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) [2] .

I.3 Modélisation orientée objets

L'orientée objets est une technique de modélisation des systèmes. Ainsi, si une modélisation orientée objets porte ce nom avec succès, 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.

I.3.1 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.

I.3.1.1 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"[3,2].

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 coïncidence) :** 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** [2]**.**

I.3.1.2 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 [2].

I.3.1.3 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 [3,2,4]**.**

Dans des langages comme le C++ et C#, 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.

I.3.1.4 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 [2].

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

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.

I.3.2 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.

I.3.2.1 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. 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).

I.3.2.2 Conception orientée objets

La **conception orientée objets** (COO ou OOD : Object Oriented Design) signifie que le modèle d'analyse est 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 [2]**.**

I.3.2.3 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 [4]**.**

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 [3,2,4]:

- 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).

I.3.3 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.

I.3.3.1 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).

I.3.3.2 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'analyse**, 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) [3, 2,4].

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.

I.3.3.3 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.

I.3.3.4 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.

I.3.4 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 [3,2] :

- ✓ 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.

I.3.4.1 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 [3,2]. 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.

I.3.4.2 Méthode d'analyse OOA

La technique d'analyse par objets proposée par Coad et Yourdon est une tentative d'incorporation de meilleures 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 [3]:

— Trouver les classes et les objets.

— Identifier les structures.

— Identifier les sujets.

— Définir les attributs.

— Définir les services.

I.3.4.3 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 [3]. 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 [3].

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.

I.3.4.4 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 [3]:

— modèle des besoins ;
— modèle d'analyse ;
— modèle de conception ;
— modèle d'implémentation ;
— modèle de test.

I.4 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]. On rencontre également l'application d'interface graphique utilisateur, la simulation dynamique et la restauration des réseaux électriques. Récemment, quelques travaux ont abordé le calcul générique des systèmes linéaires pour les réseaux électriques [7].

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).

II.1 Introduction

Dans le domaine de réseaux électriques, on constate que l'application de la MOO (Modélisation orientée objets) à beaucoup d'avantages vue que la structure physique d'un réseau électrique est adaptable à une structure de classes [8]. 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 ,1]. 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.

II.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 [4].

II.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 désirer.

II.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.

II.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 que 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.

II.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.

II.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 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, 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.

II.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 périmée. Une entreprise peut se permettre de gros investissement dans cette direction contrairement à un établissement académique où développer convient le mieux.

II.4 Structure générale de l'outil développé

Afin de tirer parti des avantages des TOO (Technique Orientée Objets), une modélisation orientée objets selon OMT (Object Modeling Technique) 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 II.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 la matrice de générateur de donnée et calcul la fiabilité.

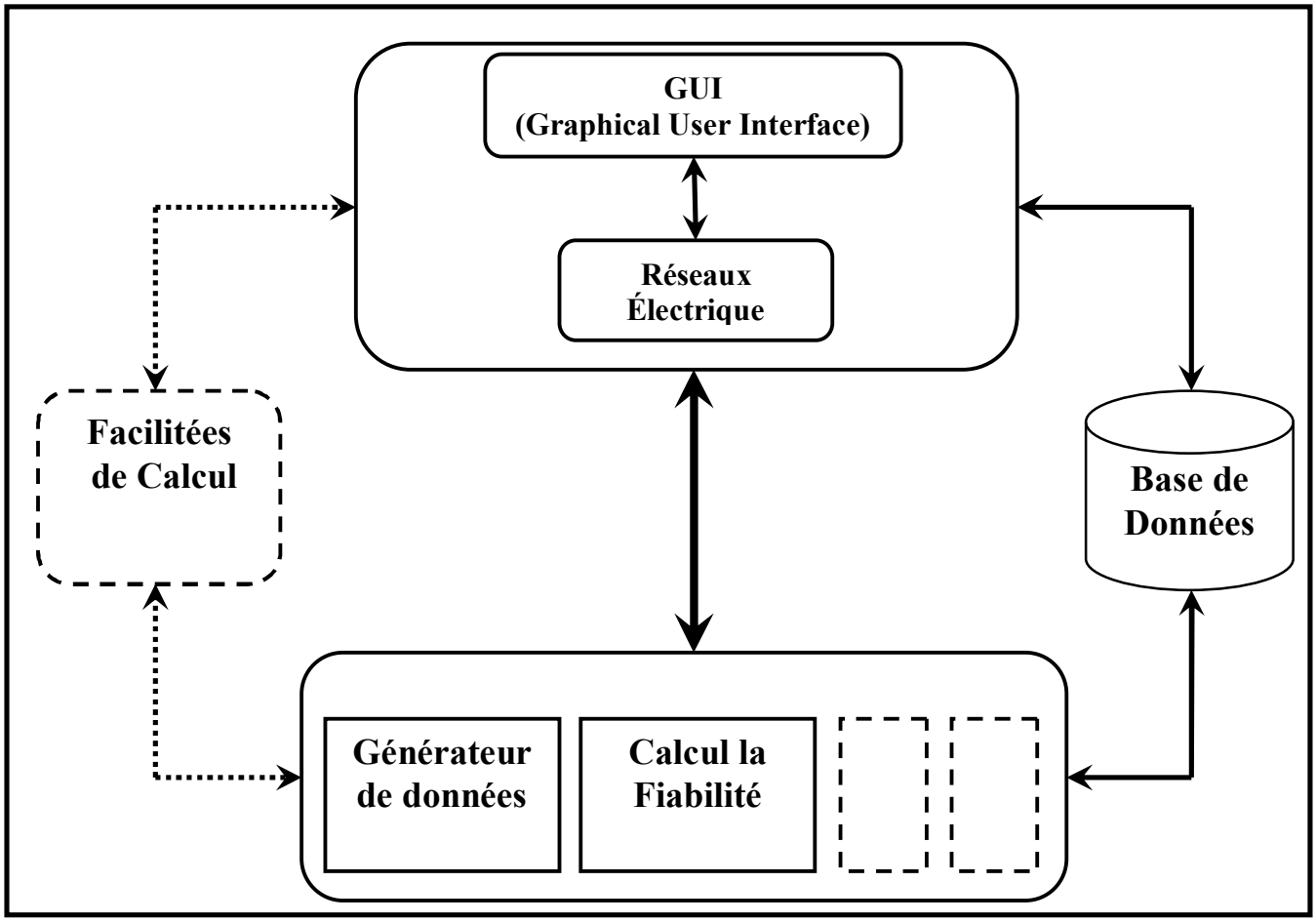


Figure II.1 Structure générale

Toutes les parties sont développées en utilisant Microsoft Visual Studio 2008 version 9.0 [10] et le langage de programmation C# [11]. Le matériel utilisé est un PC Intel IV. Pour implémenter les fonctionnalités de la GUI, l'outil développé utilise deux hiérarchies d'objets, l'une dérivant de l'objet TForm de Microsoft Visual Studio 2008 pour représenter les fenêtres elles mêmes et l'autre dérive de l'objet TGraphicControl de Microsoft Visual Studio 2008 pour représenter les éléments graphiques sur ces fenêtres.

II.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 meilleurs 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 [8].

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 [8] 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]. Dans ce travail de mémoire, on essaye de combiner la MOO, la simulation des réseaux électriques et la GUI dans un seul outil. 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 Microsoft Visual Studio 2008, C++ Builder, Delphi, Java et autres et l'arrivée de la TOO.

II.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 [2,4,8].

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 [4,8]. Par conséquent, une attention particulière doit être adressée à cette phase du projet qui n'est pas une tâche facile. 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 [1,2,3,11] ont donné des résultats satisfaisants pour des réalisations orientées objets (en utilisant le langage de programmation C#). Ces améliorations dans le temps de calcul sont dues aux développements 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 II.3 montre ces abstractions et leurs dépendances. On considère trois abstractions principales : le réseau électrique (classe Power System), les fonctions de calcul (classes des Applications) et les facilitées de calcul (classe Computational Facilities). 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 calculs.

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.

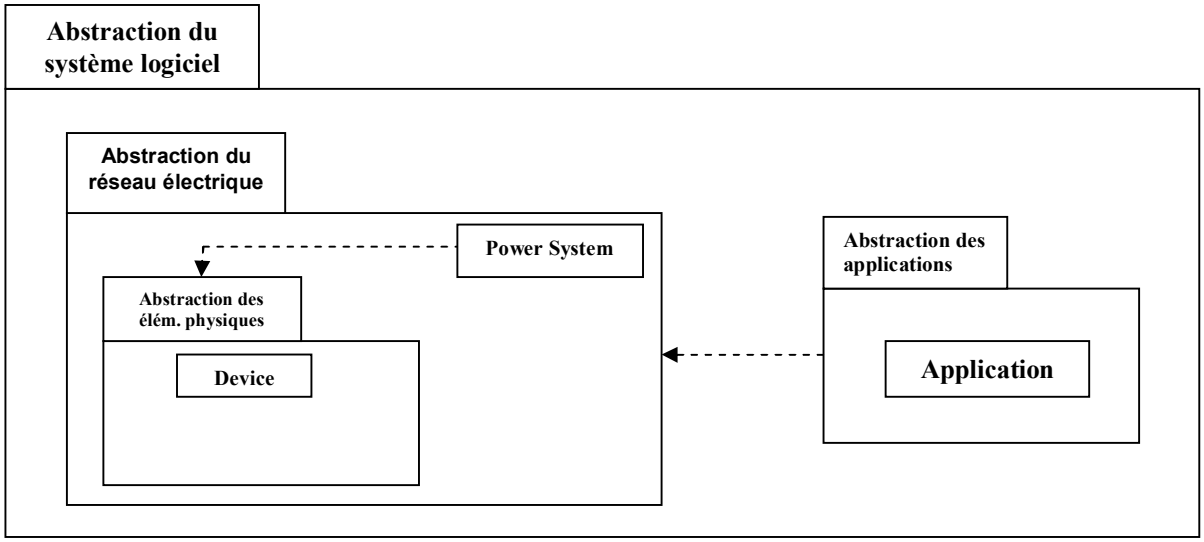


Figure II.2 Abstractions du système logiciel

L'abstraction du système électrique délimite la modélisation des éléments physiques du réseau électrique. Cette abstraction est illustrée par la figure II.3.

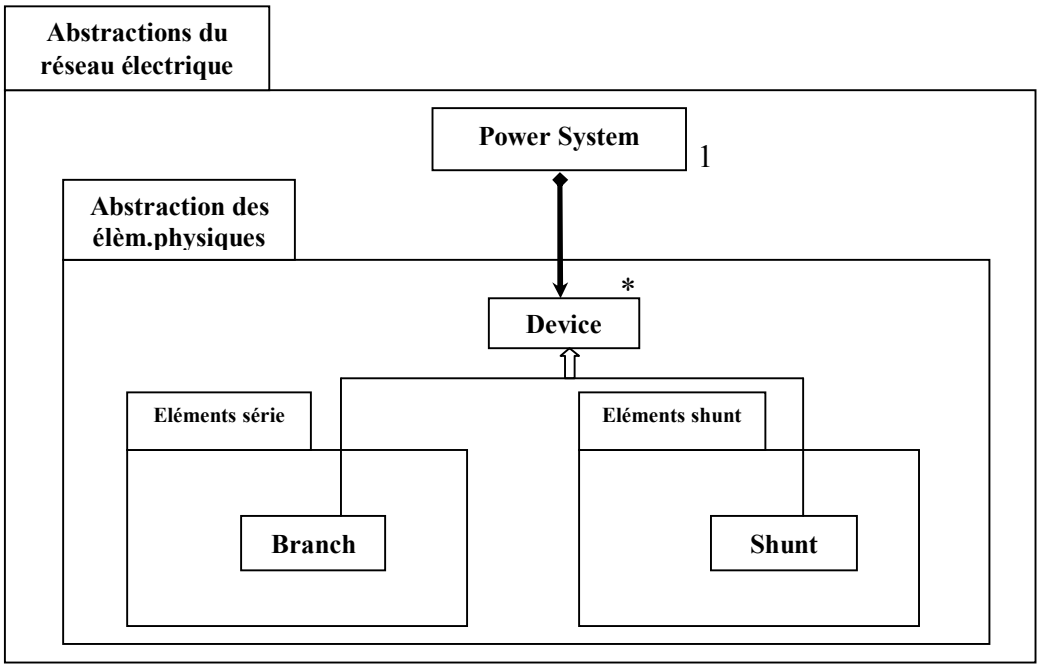


Figure II.3 Abstraction du réseau électrique

Dans cette abstraction deux entités sont identifiées : la classe (Power System) et la classe (Device).La classe (Power System) 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 II.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 constituée par la composition de ses éléments physiques divers.

II.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 mémoire, 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).

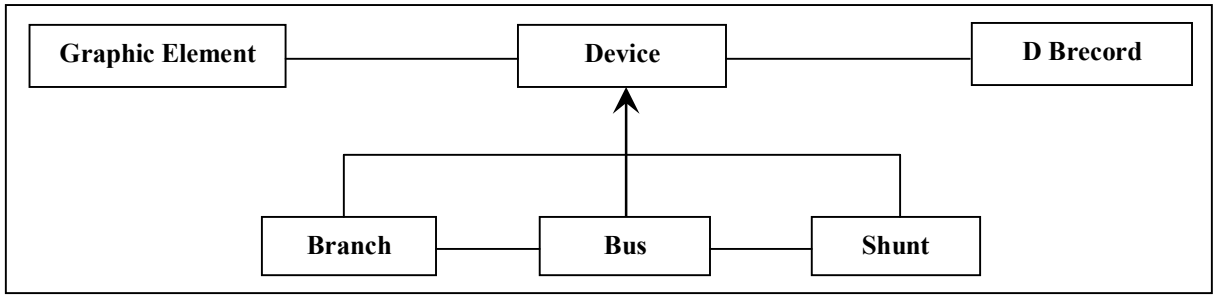


Figure II.4 Eléments physiques

II.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 II.5. Dans cette abstraction la classe PowerSystem sert de base dans la structure hiérarchique des classes représentant toutes les applications qui peuvent exister.

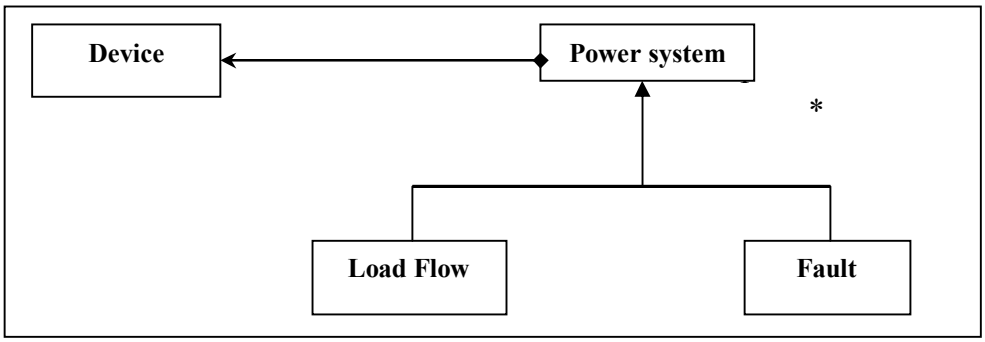


Figure II.5 Eléments physiques, réseau électrique et applications

II.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] ont pensé à réaliser une sorte de bibliothèque mathématique spéciale pour les réseaux électriques qui englobe ces structures et en utilisant la TOO.

Egalement ici la question qui 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]. 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 Template semble répondre à toutes les exigences vu qu'ils ont un aspect général et particulier en même temps.

II.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.

III.1 Introduction:

Le but de base de chaque production d'énergie électrique est de répondre à l'exigence de demande d'énergie pour le consommateur au plus bas coût possible, tout en maintenant les niveaux acceptables de la qualité et la continuité d'alimentation.

La capacité d'un réseau d'énergie électrique de fournir une alimentation raisonnable en énergie électrique est habituellement indiquée par le terme fiabilité des systèmes électriques [14].

Cette fiabilité peut être définie, d'une façon générale, comme, probabilité d'un dispositif exécutant une fonction prévue au cours de la période prévue dans les conditions de fonctionnement. Le concept de la fiabilité d'un système (réseau) électrique est extrêmement large et couvre tous les aspects de la capacité du système de répondre aux exigences des clients. L'évaluation de la fiabilité d'un système électrique, que ce soit déterministe ou probabiliste, peut être divisée en deux aspects de base (voir figure III.1.):

- ✓ Aspect d'Adéquation ou de concordance,
- ✓ Aspect de Sécurité,

Figure III.1 division de fiabilité de système

L'aspect d'adéquation est relié à l'existence des équipements de puissance suffisants dans le système pour satisfaire à la demande des charges du consommateur. L'adéquation des systèmes inclut donc les équipements nécessaires pour produire de l'énergie suffisante ainsi que les équipements associés au transport et à la distribution de l'énergie aux points de charge du consommateur [14].

L'aspect de la sécurité est relié à la capacité du système de répondre aux perturbations surgissant dans ce système. La sécurité est donc associée à la réponse du système aux perturbations.

La plupart des techniques probabilistiques actuellement disponibles pour l'évaluation de la fiabilité du système d'énergie appartiennent au domaine de l'adéquation. La technique présentée dans ce thème est également dans ce domaine.

III.2 Définitions des indices de performance:

Dans l'évaluation de la fiabilité des systèmes de distribution, la mesure de l'efficacité du service s'avère un problème de base [15].

La solution est de condenser les effets des interruptions de service dans des indices de performance qui seront employés pour prendre des décisions dans le fonctionnement des systèmes. C'est indices sont généralement des valeurs moyennes annuelles de fréquence ou de durée d'interruption.

SAIDI : (System Average Interruption Duration Index) est la durée moyenne d'interruption par client servi. Il est déterminé en divisant la somme de toutes les durées d'interruption de client pendant une année par le nombre de clients servies.

$$\text{SAIDI} = \frac{\text{somme de durée d'interruption}}{\text{nombre de clients servis}} \qquad \text{(III.1)}$$

CAIDI : Customer Average Interruption Duration Index est la durée moyenne d'interruption pour ces clients interrompus pendant une année. Il est déterminé en divisant la somme de toutes les durées d'interruption de client par le nombre de clients éprouvant une ou plusieurs interruptions pendant une période d'une année.

$$\text{CAIDI} = \frac{\text{somme de durée d'interruption}}{\text{nombre de clients interruptions}} \qquad \text{(III.2)}$$

Ces deux indices de performance expriment des statistiques d'interruption en termes de clients du système. Un client ici peut être un particulier, ferme, organisation. Si le service est fourni à un client à plus d'un endroit, chaque endroit sera compté en tant que client séparé.

III.3 Comparaison des différentes conceptions des systèmes :

L'intérêt primordial pour n'importe quelle étude de fiabilité c'est d'assurer une bonne qualité de service aux clients qui est définis comme une combinaison de la disponibilité de l'alimentation en énergie électrique et la qualité de cette énergie disponible aux clients.

Dans ce qui suit, nous allons discuter la fiabilité d'alimentation en énergie électrique pour deux configurations [14] :

III.3.1 Réseau de distribution radial simple:

Le réseau radial est la forme la plus simple, les lignes se développent en antenne en partant de la centrale ou de la station de transformation. A chaque accès (nœud) du réseau peuvent être reliés des consommateurs.

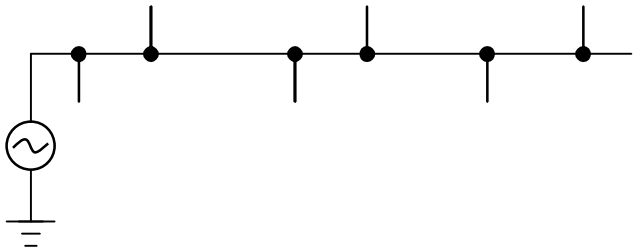


Figure III.2 Réseau radial simple.

Les dépenses pour la protection de ce type de réseau sont minimes, grâces a sa structure simple. La sécurité par contre est rudimentaire, puisque une avarie sur une ligne et l'ouverture du disjoncteur concerné entraîne une interruption de service pour tous les usagers en aval. La tenue de la tension peut également devenir difficile au bout d'une longue antenne.

III.3.2 Réseau de distribution radial avec double sources :

Dans ce type de réseau, pour plus de fiabilité en alimentation d'énergie, une deuxième source d'alimentation est placée à l'autre bout de la ligne. Sur la figure III.3 est représenté cet arrangement du système d'alimentation. Une partie de la charge est reliée à l'une des deux sources (par exemple la source S0) et l'autre partie de la charge est reliée à l'autre source d'énergie (S1).

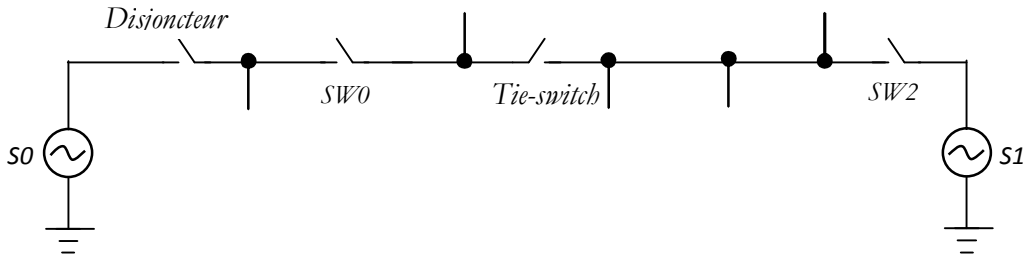


Figure III.3 système avec double alimentation

Le réseau radial (branche principale) est divisé en deux parties par un disjoncteur d'interconnexion appelé « tie switch ». La première partie est alimentée par S0, et l'autre partie est alimentée par S1, sachant que le disjoncteur d'interconnexion est ouvert en fonctionnement normal.

L'appareillage électrique est conçu pour s'adapter à 100% de sa charge. Par exemple, quand un échec se produit dans la source S0, après que le défaut ne soit isolé par le disjoncteur de la source, le disjoncteur d'interconnexion sera fermé pour permettre l'alimentation de l'ensemble du réseau à partir d'une seule source (S1) jusqu'à ce que l'avarie soit rétablie. La plupart des clients peuvent être alimentés immédiatement et ne doivent pas attendre jusqu'à ce que la source S0 soit rétablie.

III.4 Les opérations de commutation :

Il est important de noter que les opérations de commutation, aussi, mènent à une exploitation plus fiable et plus rentable ; il est à préciser que le temps d'opération du commutateur doit être moindre que le temps de réparation ; ainsi les charges qui ont été coupées de la source, peuvent être réalimentées plus rapidement par des opérations appropriées de commutation.

Il y a deux genres d'opérations de commutation [14,17].

v On isole le point d'échec ou de défaut de sorte que le point de charge, qui a perdu l'alimentation puisse être réalimenté par la source originale. Autrement dit on isole le point de défaut et après réparation du défaut, on réalimente le point de charge à partir de cette même source. Par exemple, dans la figure III.4, si un défaut se produit dans le composant *5*, le commutateur SW4 sera ouvert pour isoler le composant *5* du reste du système. La source originale S0 peut encore assurer l'alimentation à tous les clients, excepté ceux en aval du commutateur SW4.

On isole le point d'échec de sorte que le point de charge, qui a perdu l'alimentation puisse être réalimenté à partir d'une autre source ; si cette source est disponible. Donc il s'agit d'isoler le point en défaut qui cause l'interruption de la source originale et utilisé une autre source d'alimentation. Par exemple, si le composant *2* dans la figure III.4, avait un défaut permanent, SB1 et SW14 peuvent isoler le défaut. Au cas où il n'y a aucune source supplémentaire, tous segments en aval de la zone en défaut, peuvent être alimentés seulement après que le défaut soit réparé. Par contre la présence de la source supplémentaire S1 (supposant que S1 peut assurer l'alimentation), en aval de SW14 peut être utilisée par fermeture du disjoncteur normalement ouvert SW25. Le temps de rétablissement pour la présente partie est plus court avec les opérations de commutation comparé avec l'opération de réparation.

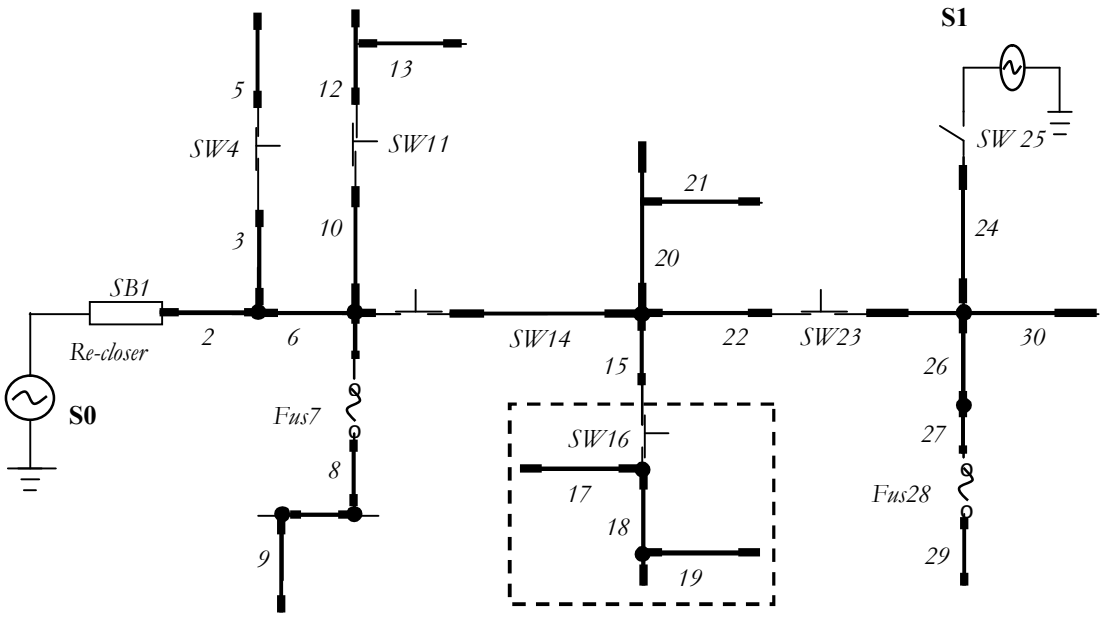


Figure III.4.Circuit d'exemple

III.5 Les composants d'analyse de la fiabilité:

Du point de vue fiabilité, un réseau électrique de distribution, est supposé diviser en deux configurations superposés [15,16,17].

La première configuration est celle qui se compose de lignes, de transformateurs, ainsi que des composants qui sont directement responsables du transport de la puissance à partir de la source ou des sous-stations de distribution vers les consommateurs (clients).

La seconde configuration est celle qui se compose de fusibles, de sectionneurs, de disjoncteurs, d'interrupteurs, etc. Ce sont les composants qui représentent le système de protection. Ce dernier est conçu d'une part, pour la détection des anomalies d'un système électrique ; et d'autre part pour isoler du reste du réseau les parties du système qui sont responsables de ces anomalies.

L'endroit des composants de protection ou d'isolement dans le système de distribution et leur réponse aux échecs peut avoir un impact important sur les indices de fiabilité. Le système de distribution est découpé dans des segments par les composants de protection et d'isolement. Dans la suite de notre travail, le système électrique n'est pas modélisé en termes de composants mais en termes de segments [16,17].

III.6 Notion de segments:

Un segment est un groupe de composants, dont le composant d'entrée est un commutateur ou un dispositif de protection. Ce dispositif de coupure (des fusibles, des sectionneurs, des disjoncteurs … etc.) isole des groupes de composants dans les sections indivisibles. Chaque segment a seulement un commutateur ou un dispositif de protection.

Dans la figure III.5, la seule protection sur le segment est le disjoncteur « D » de la source. L'échec d'un des composants dans le segment peut causer une interruption au point de la charge1. Ceci est pareil pour les autres points de charge (2, 3, 4, et 5). Aucune restauration provisoire n'est possible. Pour cette configuration, la fiabilité de toutes les charges (1,2, 3, 4, et 5) est identique.

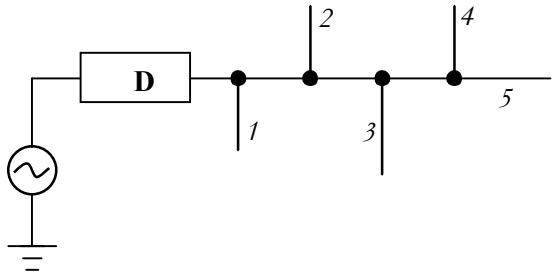


Figure.III.5-Simple Segment

Le nom d'un segment est identique à celui de son dispositif de protection (fusibles, sectionneurs, disjoncteurs etc.), dans notre cas et sur la figure III.5, il y a seulement un segment, donc c'est le segment D (disjoncteurs D), par ailleurs, les composants 1, 2, 3, 4, et 5 appartiennent donc tous au segment D.

La modélisation de notre système d'alimentation en termes de segments accélère les calculs des indices de fiabilité. L'algorithme programmé répondra plus rapidement puisque seulement les dispositifs de protection sont traités, sans les composants intermédiaires.

III.7 Ensembles d'analyse de la fiabilité :

Dans l'analyse de la fiabilité, l'échec de l'un des éléments qui peuvent causer une perte de service en un point de charge particulier doit être considéré [14,17]. (Ce point de charge sera représenté en termes de segment, qui est le segment d'intérêt S). L'échec de l'un des composants du système, qui sont placés le long du feeder entre la source et le segment d'intérêt, peut causer une interruption au point de charge.

L'échec de l'un des composants qui ne sont pas placés le long du feeder (chemin d'alimentation) peut également causer une interruption au point de charge, à moins que le composant soit séparé du chemin par un dispositif de protection qui répond automatiquement à l'échec du composant.

L'ensemble « L », représenté sur la figure III.6, contient tous les segments dans un circuit dont l'échec de l'un d'eux peut causer la perte de puissance au segment d'intérêt S. l'ensemble « L » inclut tous les segments qui ne sont pas séparés du chemin continu entre la source (sous-station, générateur, etc.) et le segment d'intérêt S par un dispositif automatique de protection.

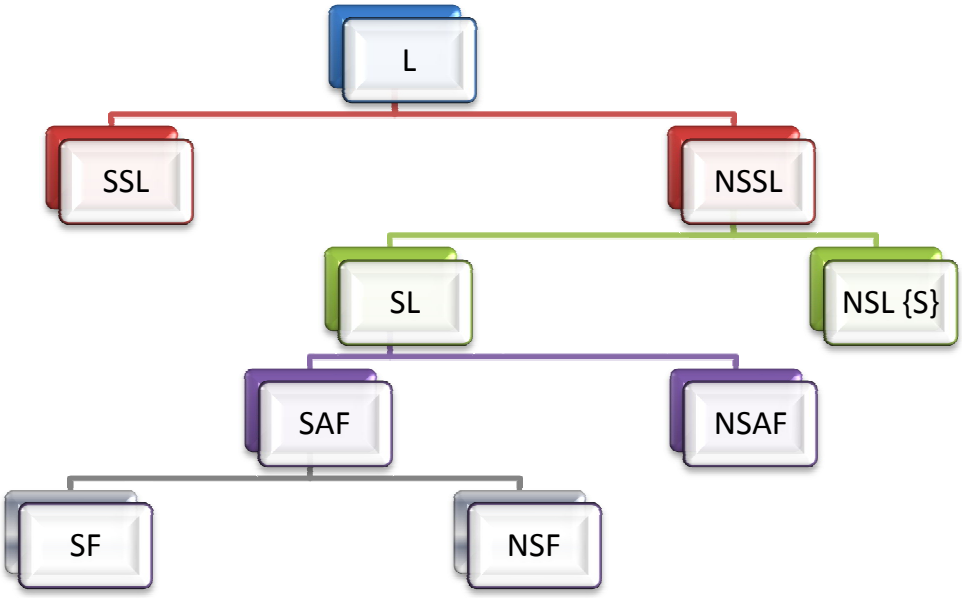


Figure III.6 Analyse de la fiabilité

Maintenant l'ensemble « L » est divisé en deux sous ensembles « SSL » et « NSSL » :

- SSL est un ensemble qui contient les segments qui peuvent être isolés du chemin continu entre S et la source originale.
- NSSL est l'ensemble qui contient les segments qui ne peuvent pas être isolé du chemin continu entre S et la source originale (segments qui n'ont pas d'influence sur le chemin d'alimentation de l'ensemble S).

L'ensemble « SSL » contient tous les segments qui peuvent être séparés du chemin continu par les commutateurs manuels. Si n'importe quel élément de cet ensemble est défaillant, le segment d'intérêt S peut être temporairement alimenté de la source originale avant que le composant défaillant soit réparé ou remplacé.

L'ensemble « NSSL » peut être partagé en deux sous ensembles « SL» et « NSL» :

- SL comprend les segments qui peuvent être isolés du segment d'intérêt S, ainsi si un échec se produit au niveau de l'ensemble « SL », l'ensemble « S » peut être alimenté par une autre source.
- NSL est un ensemble qui ne comprend qu'un seul segment qui n'a pas d'influence sur l'alimentation du segment d'intérêt « S », parce que l'ouverture ou la fermeture de ce segment reste toujours en amont, donc n'as aucune influence sur le chemin d'alimentation du segment S.

L'ensemble SL lui aussi peut être divisé en deux sous ensembles SAF et NSAF:

- SAF, L'ensemble des segments qui étant en panne, ne prive en aucun cas l'alimentation temporaire de la source de secours.
- NSAF, L'ensemble des segments qui étant en panne, prive d'alimentation temporaire le segment S à partir de la source de secours.

L'ensemble SAF peut être divisé en deux sous ensembles SF et NSF:

- SF est l'ensemble qui se compose des segments qui peuvent isoler le segment S de la source, mais permettent par ailleurs l'alimentation du segment S à partir de la source de secours.
- NSF est l'ensemble qui se compose des segments qui peuvent isoler le segment S de la source, et ne permettent pas l'alimentation du segment S à partir de la source de secours en raison de violation des contraintes de système.

Donc :

$$L=SSL \cup NSSL \quad \text{(III.3)}$$

$$L=SL \cup \{S\} \quad \text{(III.4)}$$

$$SL=SAF \cup NSAF \quad \text{(III.5)}$$

$$SAF=SF \cup NSF \quad \text{(III.6)}$$

Équations (III.3)-(III.6) rendement.

$$L= SSL \cup SF \cup \{S\} \cup NSAF \cup NSF. \quad \text{(III.7)}$$

III.8 Indices de fiabilité:

Cette analyse se fonde sur deux classes générales d'information pour estimer la fiabilité ; les paramètres de fiabilité et la structure (composants) du système [15]. En utilisant la structure du système et les données de performance des composants, la fiabilité des points de charge spécifiques ou le système entier de distribution peut être évaluée. L'information de la structure est obtenue par les traces de circuit présentées précédemment. Dans les paragraphes suivants les données de performance sont discutées.

Les techniques prédictives de fiabilité souffrent des difficultés pour collecter les données.

III.8.1 LES paramètres de fonctionnement:

Chaque élément est caractérisé par les paramètres de fonctionnement suivant [14,15]:

- Taux d'échec annuel = la fréquence moyenne annuelle de l'échec,
- Temps de panne annuel = la durée annuelle de panne sentie à un point de charge.

Le taux d'échec pour le segment i, FR_i (failure rate), et la somme des taux d'échec de tous composants contenu dans le segment i comme donné ci après.

$$FR_i=\sum_{j=1}^{n} FRj \qquad (III.8)$$

Où : FRj = le taux d'échec pour le composant j,

n = le nombre de composants dans le segment i.

Le temps de réparation moyen pour un segment i, REP_i, qui peut être calculé par :

$$REP_{i=} \sum_{j=1}^{n}(FRj \times REP_{i)} \qquad (III.9)$$

FRj = le taux d'échec pour le composant j,

REP_i = le temps de réparation moyen pour le composant j, et

n = le nombre de composants dans le segment i.

Ces indices sont calculés pour chaque segment dans le système électrique. On suppose dans notre algorithme que tous les charges dans un segment ont le même taux d'échec et seront affectées par le même temps de panne.

Dans le programme d'analyse de fiabilité, les taux d'échec et les temps de réparation des données de champ sont choisis. Quand ces données ne sont pas disponibles, des valeurs par défaut sont recherchées dans une base de données relationnelle, qui a des taux d'échec et des temps de réparation moyens génériques pour chaque type de dispositif.

III.8.2 Calcul des indices de fiabilité :

Après que les ensembles de fiabilité sont déterminés pour le segment d'intérêts S, les indices de fiabilité peuvent être calculés pour s'en assuré qu'il n ya qu'un seule défaut dans le système. Le temps d'indisponibilité sera pour le segment S :

$$DT_S = \sum_{\substack{i \in NSL \\ NSAF \\ NSF}} (FR_i * REP_I) + \sum_{\substack{i \in SSL \\ SF}} (FR_i * SOT)_i \quad \text{(III.10)}$$

Ou : sot : Switch opération time (temps d'opération de commutation)

III.9 Conclusion :

Dans cette partie de notre mémoire, nous avons donné la définition de la fiabilité, en expliquant les différentes parties de notre algorithme de calcul, afin de l'introduire dans notre application en mettant l'accent sur la fiabilité sachant que cette dernière est la partie qui demande le plus de données détaillées et ceci dans le but de valider notre générateur de données.

Par ailleurs nous avons donné les définitions des différentes matrices qui peuvent analyser et calculer la fiabilité et ses différents indices.

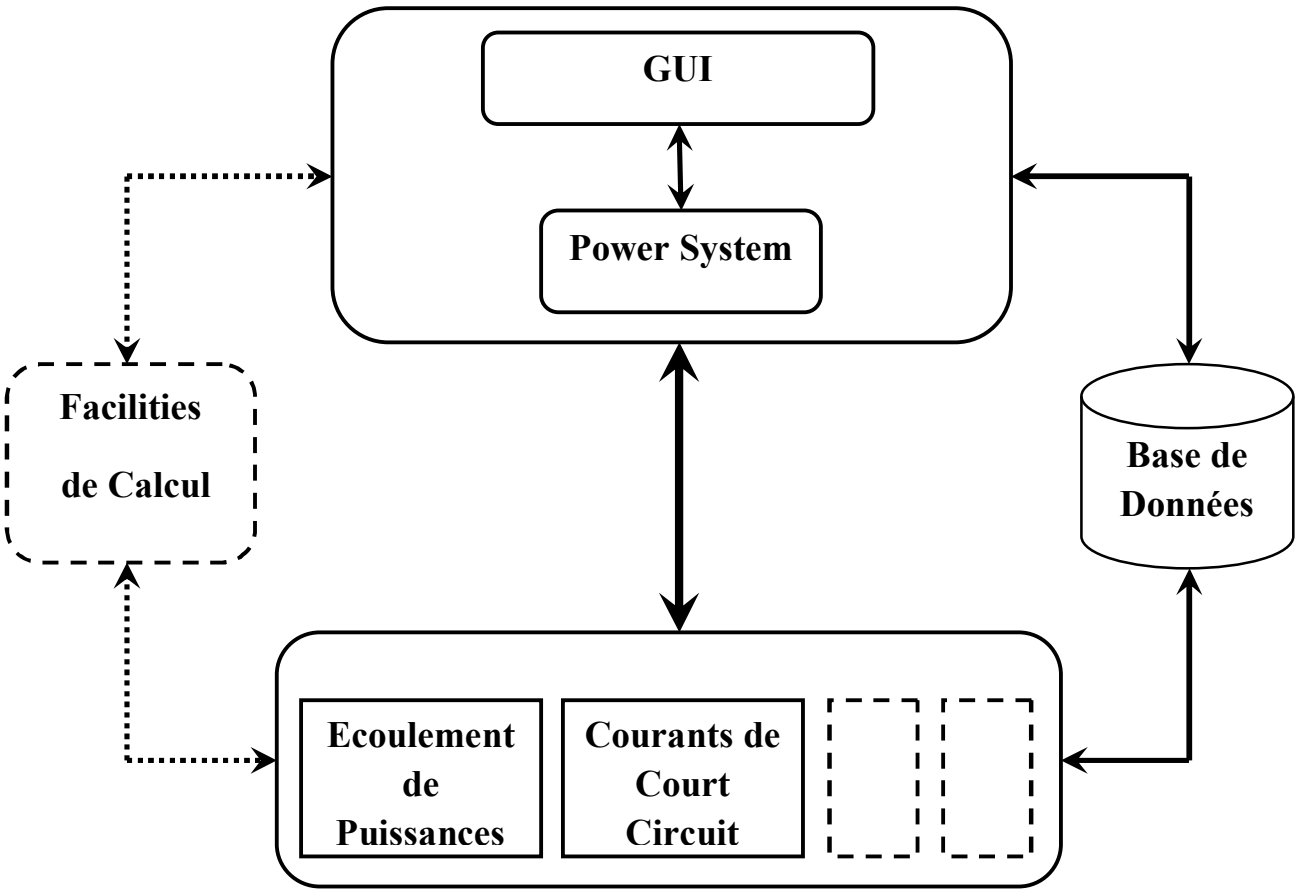


FIGURE 2.1 : Structure générale

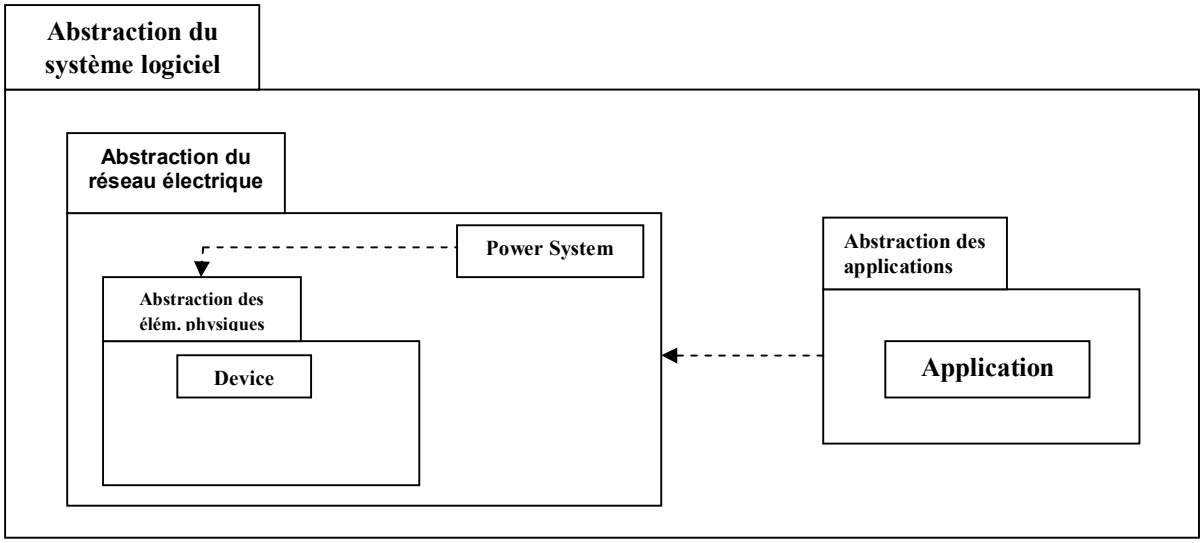


FIGURE 2.2 : Abstractions du système logiciel

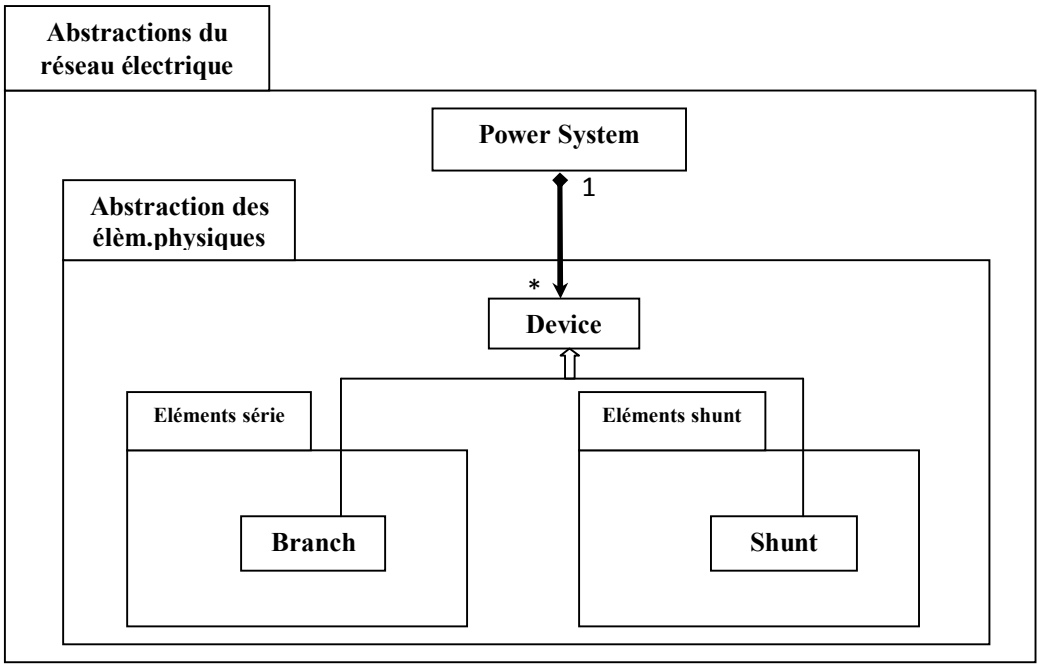


FIGURE 2.3 : Abstraction du réseau électrique

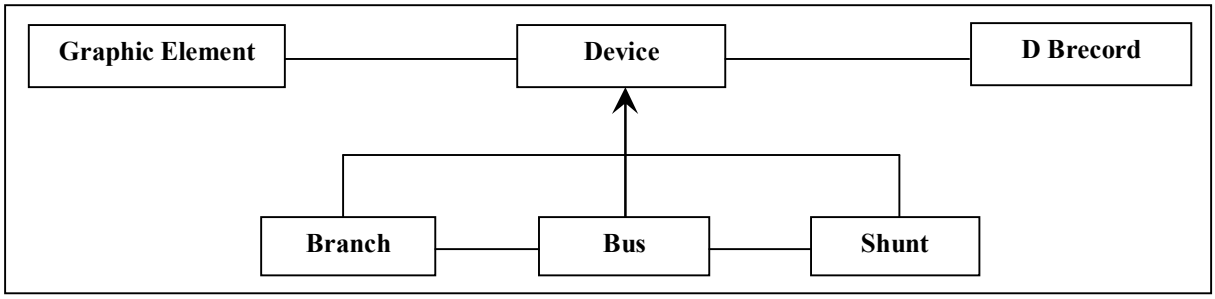


FIGURE 2.4 : Eléments physiques

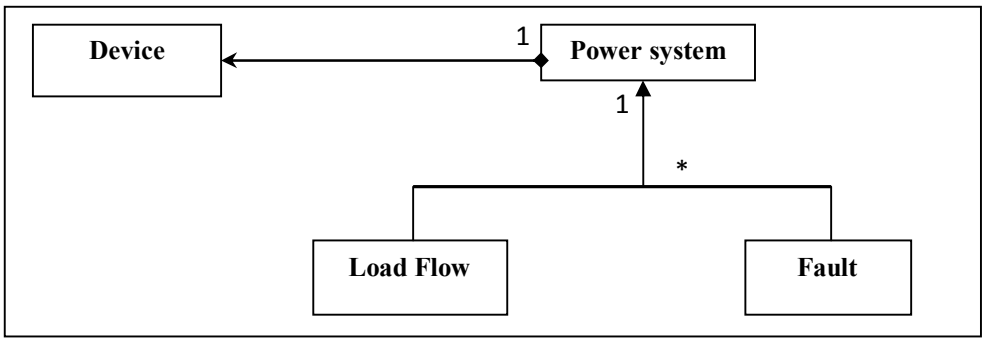


FIGURE 2.5 : Eléments physiques, réseau électrique et applications

Conclusion Générale

Le développement d'une plate forme logicielle pour la simulation des réseaux électriques en utilisant la technique orientée objets, a été abordé dans ce mémoire. Aussi les premiers composants élémentaires ayant servi à son développement ont été présentés.

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 canaliser sur les diversités de chaque application. Les trois applications intégrées dans ce logiciel sont la génération de données graphiques, la génération de données techniques et le calcul de la fiabilité.

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 de l'Electrotechnique en général.

Notre contribution se résume dans les points suivants :

1. Notre application n'a pas besoin de créée à chaque fois un nouveau modèle entre les objets pour les déférentes applications, contrairement aux travaux précédents, donc c'est une technique très ouverte et flexible pour toute application.

2. Elle contient une classe pour chaque élément de réseau électrique, autrement dit, toute les classes des objets sont séparées entre elles, ce représente un avantage primordial quant la facilité de programmation.

3. Elle facilite la modification sur la plate forme elle-même que ce soit la suppression ou l'ajout d'un objet, d'une classe ou autre application.

4. Elle fonctionne d'une manière dynamique, autrement dit, que la variation de n'importe quel élément dans le schéma du réseau électrique, crée une mise à jour automatique dans le cœur de la base de données de notre générateur.

5. Sur la partie affichage, elle assure le déplacement, rotation, suppression, ajout et possibilité d'affichage de toutes les données du générateur graphique sur l'éditeur graphique.

6. Elle permet le dessin (sur éditeur) de tout type de réseaux électrique (radial, bouclé et mixte ; transport ou distribution)

7. Elle permet l'ajout d'autres algorithmes nécessaires pour le calcul d'autres phénomènes (court-circuit, écoulement de puissance, stabilité, protection) sans toute fois modifier le programme de base.

8. Elle permet la sauvegarde de la base de données de notre générateur dans un fichier et permet d'utiliser ces mêmes données pour être utilisées dans un autre logiciel (Matlab, Delphi, Builder, C++, Pascal, etc.).

9. Elle n'a pas besoin de personne qualifiée sur la POO, pour ajouté un algorithme de calcul dans notre logiciel, Elle a besoin seulement de qualification en programmation sous C#.

Il est important de mentionner que ce logiciel est en version 0, autrement dit version test ; donc il est à remarquer qu'il comporte quelques inconvénients principaux que nous citons ci-dessous :

1. Etre qualifié en programmation C#, pour programmer avec la base de données de notre générateur. Car il y a une énorme quantité d'informations graphiques.

2. Le dessin sur l'éditeur graphique n'est pas totalement à point.

3. L'éditeur graphique présente une certaine lourdeur ; ce qui demande une carte graphique performante. Car la programmation est réalisé sur une plateforme « dotNetFramework.3.5» qui demande une certaine performance graphique.

2. Perspectives et Suggestions

1. Réaliser une première version.
2. Assouplissement de la fonction « dessin » sur l'éditeur graphique.
3. Ajout d'algorithmes pour le calcul et l'analyse d'autres phénomènes.
4. Personnaliser la base de données de chaque objet, afin de pouvoir introduire les données propre de chaque objet.
5. Acquisition on-line de données d'un réseau réel.