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

UNIVERSITE DE BATNA
FACULTE DES SCIENCES
Département d'informatique

جامعـــة باتـنة
كليـــة العلـــوم
قسم الإعلام الآلي

Mémoire

Pour obtenir le diplôme de
Magistère en informatique
Option : Systèmes informatiques Intelligents et Communicants (SiIC)

CA-SPL : Ligne de Produits Logiciels Pour Systèmes Sensibles au Contexte

Par : **FERRAH Abdelhafid**

Composition du jury

Dr. **Rachid SEGHIR**	MCA	Président	Université de Batna
Dr. **Ammar LAHLOUHI**	MCA	Rapporteur	Université de Batna
Dr. **Abdelhak BOUBETRA**	MCA	Examinateur	Université de Bordj-Bou Arreridj
Dr. **Foudil CHERIF**	MCA	Examinateur	Université de Biskra

Année 2014/2015

Remerciements

Un mémoire est constitué de moments de joie et de doute. Mais elle est en premier lieu constructive pour soi. Les remerciements exposent son coté positif car ce sont les seuls souvenirs qui méritent de rester à sa surface.

Ma profonde gratitude va tout d'abord au bon dieu de m'avoir donné le courage d'achever le présent travail.

Mes premiers remerciements sont donc destinés à mon encadreur : M. Lahlouhi Ammar, qui a été le premier à me faire confiance. C'est un plaisir de travailler avec vous et j'ai eu la chance de beaucoup apprendre à vos cotés.

Je remercie les membres de mon jury.

Je remercie tous les membres de ma famille.

Enfin, je dédie cette thèse à ma mère car c'est aussi la sienne.

Abdelhafid, le 16 novembre 2014

Résumé

Les applications mobiles s'exécutent dans des environnements où le contexte change d'une façon continue. Donc, il est nécessaire de fournir un support d'auto-adaptation dynamique, afin de permettre aux systèmes d'adapter leurs comportements selon le contexte de l'exécution. Ce support est actuellement accompli par des plateformes d'intergiciels, qui offrent un service de reconfiguration dynamique sensible au contexte. Cependant, le défaut principal d'approches existantes soit qu'elles ne sont pas convenables pour les appareils mobiles ou elles utilisent un algorithme génétique (AG) au moment de l'exécution pour générer un plan de reconfiguration. L'exécution d'un AG sur un appareil mobile provoque le disfonctionnement de certains services avec des configurations produites presque justes (86-90%). Dans ce mémoire, on présente une approche combinée des réseaux de neurones (RN) multicouches et des algorithmes génétiques, afin de permettre la génération automatique d'un plan de reconfigurations au moment de l'exécution, sans diminuer les performances de l'appareil. L'utilisation de RN lors de l'exécution afin de produire un plan de reconfiguration est achevé par : (1) avoir les informations sur la variabilité du contexte et d'applications, (2) l'utilisation d'AG pour générer des configurations optimales pour chaque situation de contexte, dont le but de produire des exemples d'apprentissages, (3) l'exécution de l'algorithme d'apprentissages de RN au moment de la conception afin d'adapter ses poids à l'ensemble d'exemples générés, et finalement (4) le RN est utilisé au moment d'exécution dans le but de générer un plan de reconfigurations dans un temps optimal. Nous avons spécifié un cas d'étude et l'évaluation de notre approche, les résultats montrent que l'exécution de notre approche est efficace.

Mots clés : Reconfiguration Dynamique, Contexte, Lignes de produits logiciels, Algorithme Génétique, Réseau de neurones, informatique autonome.

Table des matières

Sommaire

Chapitre VI : Conclusion Générale

.

Liste des figures

Liste des tableaux

Introduction Générale

L'évolution technologique et la baisse des prix du matériel informatique ont facilité l'accès de tous à ce matériel. Les appareils informatiques spécialisés deviennent abondants et leur utilisation devient aisée et généralisée à tous les domaines de notre vie quotidienne. Ce phénomène a amené au paradigme de l'informatique ubiquitaire (ubiquitous computing paradigm [Weiser, 1990]) qui assure la disponibilité des services partout et à tout moment. Cette évolution a incité les développeurs à intégrer les terminaux mobiles dans leurs applications qui sont utilisées souvent dans des environnements différents. Ces derniers produisent une variété de nouvelles situations dont lesquelles une application peut être utilisée. Une hypothèse, qu'un ensemble de chercheurs en informatique ubiquitaire partagent, est de permettre aux appareils et aux applications de s'adapter automatiquement au changement produit par leurs entourages (physique et / ou virtuel) ; ce qui permettra d'améliorer l'expérience de l'utilisateur.

Les informations de l'entourage (l'environnement physique et virtuel) créent un *contexte* de communication entre l'utilisateur et la machine. Dans [Dey-b, 2001],Dey définit un contexte comme toute information qui caractérise une situation associée à l'interaction entre les utilisateurs, les applications et le milieu environnant. L'utilisation du contexte dans les applications est un domaine de recherche d'actualité connu sous le nom de « sensibilité au contexte » (ou context-awarness en anglais). L'adaptation des applications présente un nouveau type de systèmes qui utilisent intelligemment toutes les informations disponibles à la fois au moment de la conception, pour remettre et minimiser l'impact des décisions des utilisateurs, et au moment de l'exécution, pour profiter de l'information disponible dans l'environnement de l'application, où il y a beaucoup de situations à considérer.

Dans les premiers systèmes sensibles au contexte, tels que [Asthana, 1994, Abowd, 1997], les interactions de l'application avec son entourage (environnement) étaient réalisées en programmant ces interactions au cœur même de l'application. Le couplage entre le code applicatif et le code d'interaction avec l'environnement est fort. Ce qui ne facilite ni le changement de type d'objet communicant, ni la reconfiguration de la sensibilité au contexte. Le découplage entre le code métier de l'application et le code de gestion de la sensibilité au contexte peut-être réalisé grâce aux techniques suivantes [Taconet, 2011] :

(1) la programmation par aspects,

(2) intergiciels orientés composants,

(3) L'ingénierie dirigée par les modèles (ou IDM) [Taconet, 2011].

Les travaux dans le domaine de la sensibilité au contexte, s'intéressent aux étapes de capture, d'interprétation, de modélisation et de dissémination du contexte. Nous constatons l'absence d'une approche générique et complète pour adapter les applications au contexte. Ces travaux présentent des approches d'adaptation ad hoc spécifiques à un domaine particulier (comme la sensibilité à la localisation de l'utilisateur).

Dans ce mémoire, nous montrons l'importance du paradigme des lignes de produits logiciels (SPL) dans le développement des systèmes adaptatifs. SPL est une approche génie logiciel pour la création des applications configurables et adaptables à une variété de besoins. SPL est construit autour d'un ensemble de composants logiciels avec les points de variabilité qui permettent la personnalisation. Le développement d'une SPL comprend deux processus de développement :

(1) Ingénierie de domaine (*domain engineering*) : c'est le processus de développement des artefacts logiciels communs et variables de SPL [Pohl, 2005],

(2) Dérivation de produits (*product derivation*) : c'est le processus de configuration des artefacts logiciels réutilisables pour spécifier un ensemble de besoins des clients ou du marché.

Bien que ce dernier processus soit communément conçu pour le processus de développement des produits, il peut être aussi étendu pour couvrir l'adaptation d'un produit au moment de l'exécution [Parra-b, 2011]. Dans [Hallsteinsen, 2008], les auteurs introduisent SPL pour la dérivation des logiciels qui ont besoins de s'adapter au moment de l'exécution. Ils qualifient ce type de SPL de dynamique. Une SPL Dynamique (DSPL) est capable de produire des systèmes qui peuvent s'adapter durant l'exécution afin de couvrir dynamiquement de nouveaux besoins et un changement des conditions de l'environnement.

Chaque produit dérivé par une SPL est représenté par une combinaison unique et valide de caractéristiques. Une combinaison de caractéristiques est gouvernée par un modèle de caractéristiques FM (*feature model*) de SPL qui définit les relations entre les caractéristiques. Le modèle de caractéristiques est une architecture qui définit les relations entre les caractéristiques d'une manière hiérarchique. Malgré les bénéfices de SPLs, des études de cas industriel ont montré que la dérivation d'un produit est encore une activité coûteuse et consomme du temps [Deelstra, 2005]. Les développeurs sont en face d'un nombre de défis quand ils essaient de

dériver une configuration (sélection de caractéristiques) optimisée qui satisfait un ensemble arbitraire de besoins. Même avec un petit modèle de caractéristiques, une combinaison de caractéristiques peut produire un nombre exponentiel de configurations [Guo, 2011]. De plus, une fois la sélection des caractéristiques est faite, la configuration doit être examinée pour vérifier les contraintes définies dans FM et aussi les contraintes des ressources (les besoins non-fonctionnels). Cette vérification rend le processus de sélection des caractéristiques plus complexe.

Trouver une configuration optimale qui est conforme aux contraintes de FM et aux contraintes des ressources est un problème NP-hard (un algorithme exact ne peut pas être utilisé en un temps raisonnable parce que le problème à résoudre est non-deterministic polynomial-time hard) [White, 2009]. Des différentes approches SPL concernées par le problème d'optimisation d'une configuration, ne prennent pas en considération les contraintes de ressource (contexte de l'environnement) [Benavides, 2008]. D'autres recherches dans la communié SPL ont appliquées et présentées des différentes techniques pour résoudre le problème de sélection des caractéristiques [Benavides, 2005] [Czarnecki-a, 2007], ces recherches utilisent des techniques exactes mais présentes la complexité de temps exponentiel. D'autres approches SPL sont basées sur des algorithmes génétiques (AG). Un AG est une technique d'optimisation stochastique et une recherche globale heuristique dérivée de la génétique et de l'évolution naturelle [Mitchell, 1996]. L'objectif des de telles approche SPL, est d'adapter le AG pour optimiser la sélection des caractéristiques au moment de la conception [Guo, 2011, White, 2009, Benavides, 2005, Czarnecki-a, 2007]. Dans [García, 2013] une approche qui permet la génération automatique de configurations et de plans de reconfiguration selon le principe de l'informatique autonome (AC) au moment d'exécution. Ils ont utilisé le GA proposé dans [Guo, 2011] pour optimiser la configuration (sélection des caractéristiques).

Malgré que les techniques d'optimisation présentées fournissent des configurations optimales, le temps nécessaire à une telle génération demeure encore non raisonnable pour les applications mobiles.

Dans ce travail, nous proposons une approche SPL pour des applications sensibles au contexte qui est plus performante en termes de temps d'exécution occupé par le service de reconfiguration dynamique. Une telle approche est basée sur l'utilisation combinée des réseaux de neurones multicouches et des algorithmes génétiques.

Ce mémoire est organisé en six chapitres en plus d'une introduction.

Le chapitre 1décrit un état de l'art sur les systèmes sensibles au contexte.

Le chapitre 2 est un survol des solutions non SPL existantes pour le problème de la sensibilité au contexte.

Le chapitre 3 Un survol des solutions SPL pour la sensibilité au contexte et la mise en évidence de la nécessité d'une solution SPL dynamique.

Le chapitre 4 met en clair les contributions que nous avons pu réaliser dans le cadre de ce travail.

Le chapitre 5 décrit une implémentation d'une étude de cas.

Finalement, nous terminons par une conclusion.

Chapitre I

Contexte et fondement

Sommaire

Introduction :

De nos jours, l'informatique devient de plus en plus omniprésente. À la différence des systèmes d'information classiques, L'explosion de la téléphonie mobile et la présence sur le marché des ordinateurs portables, d'assistants numériques et de toutes sortes de plates-formes clientes portables, constituent des indicateurs forts des utilisations futurs des systèmes informatiques impliquant plus de mobilité et de variété d'usage.

Dans le cadre de ce mémoire, on s'intéresse aux applications sensibles au contexte dans un environnement dynamique et hétérogène. Ce dernier est soumis aux caractéristiques variables des ressources, des dispositifs utilisés et à la mobilité des usagers. Ce qui rend nécessaire l'utilisation des applications sensibles au contexte qui détectent les variations de l'environnement et adaptent leurs comportements en conséquence.

Nous proposons dans ce chapitre une introduction aux systèmes sensibles au contexte, les solutions classiques et SPLs existantes pour le problème de la sensibilité au contexte, ainsi qu'un aperçu des principaux avantages offerts par leurs utilisations et leurs limites, Ainsi que la nécessité des SPL dynamiques.

1. Notion de contexte

On distingue dans les articles de recherche plusieurs définitions du mot contexte suivant les domaines de l'ingénierie. Nous commençons par donner plusieurs définitions utilisées dans le domaine de l'informatique.

La définition littérale du mot contexte considère le contexte comme « l'ensemble des unités d'un niveau d'analyse déterminé (phénomène, unité lexicale, phrase...) constituant l'entourage temporel (parole) ou spatiale (écriture) d'une unité » [C. national]. Suivant la même source, le terme contexte représente aussi « un ensemble de circonstances liées à un événement qui se produit ».

Plusieurs travaux qui définissent le contexte à travers des énumérations des entités spécifiques qui représentent le contexte associé à une application. Schilit et Theimer [Schilit, 1994] qui considère le contexte comme un ensemble d'informations de localisation et l'identité des personnes et des objets à proximité ainsi que les modifications pouvant intervenir sur ces objets, est l'une des premières définitions du contexte. Sensiblement la même définition a été reprise par Brown et al [Brown, 1997], ces derniers considèrent le contexte comme étant un ensemble d'informations de localisation, du profil de personnes auxquels il ajoute des informations temporelles. Ryan et al [Ryan, 1997] ajoutent à cette énumération des informations sur l'environnement telles que la température.

Dans des définitions plus générales du contexte, on trouve les définitions suivantes : Le contexte est la propriété qui minimise la quantité d'informations échangée entre des entités communicantes : humain-humain, humain-machine ou machine-machine [Dey, 1999]. C'est une information implicite de l'entourage du discours pour rendre le dialogue beaucoup plus simple et efficace.

Les différentes notions ont été proposées. Dans [Dey-a, 2001], *Dey* définit le contexte comme "toute information pouvant être utilisée pour caractériser la situation d'une entité (personne, objet physique ou informatique). Plus généralement, tout élément qui peut influencer le comportement d'une application. Dans [Brézillon, 2002] *Brézillon*, définit le contexte par "ce qui n'intervient pas directement dans la résolution du problème, mais qui contraint sa résolution". *Gaetan Rey et Joelle Coutaz* [Rey, 2002] proposent plusieurs axes de définitions du contexte:

- il n'y a pas du contexte sans contexte: le contexte doit être défini en fonction d'une finalité)
- le contexte est un espace d'informations qui sert à l'interprétation : la capture du contexte n'est pas l'objectif, mais les données capturées doivent servir un but.
- le contexte est un espace d'information partagé par plusieurs acteurs : utilisateur et système.
- le contexte est un espace d'information infini et évolutif : il se construit au cours du temps.

Les modèles du contexte sont pluridisciplinaires [Bradley, 2005]. La recherche linguistique est concernée par l'analyse du contexte d'utilisation des signes (mots) dans un langage. [Bunt, 1994] définit cinq types du contexte pour les aspects de communication, qui sont respectivement :

- linguistique : lié au matériel linguistique.
- sémantique : lié à la description du domaine.
- physique : lié à la description de l'environnement dans lequel l'action ou l'interaction se produit.
- sociale : lié à la situation interactive qui se produit entre les acteurs.
- cognitif : lié aux intentions des participants, leur évolutions dépendent de la perception, la production, l'évaluation et l'exécution.

[Coutaz, 2002] propose un modèle cumulatif où le contexte (Ctx) est une agrégation temporaire de situations. Une situation est un descripteur de l'état pour un utilisateur (U) exécutant une tâche (T) à un certain moment (t). Le modèle est décrit par la formule suivante:

$$Ctx(U,T,t) = \bigcup_{n=1}^{m} Situation(U,T,t_n)$$

L'utilité des informations contextuelles dans les systèmes sensibles au contexte sont utilisées principalement pour [Baldauf, 2007, Kirsch, 2005]:

- Adapter le comportement, les interfaces et le contenu d'un système en fonction du contexte d'utilisation, tel qu'afficher un plan du quartier où se trouve l'utilisateur en utilisant ses préférences de visualisation et en accord aux caractéristiques de son dispositif mobile,

- Suggérer le contenu qui soit le plus approprié au contexte d'utilisation. Par exemple, un système du guidage touristique notifie l'utilisateur chaque fois qu'il est proche d'un monument historique dont la description est présente sur le système,
- Annoter des documents à l'aide d'informations contextuelles qui peuvent être exploitées pour la réalisation à posteriori des requêtes ou des mécanismes de partage du contenu,
- Proposer et exécuter des services spécialisés selon le contexte. Par exemple, des services proposés à l'utilisateur selon sa localisation ou son dispositif.
-

1.2. Types de contexte

Selon [Stefan, 2009], il existe trois principaux types de sensibilité au contexte de l'environnement externe:

- *contexte de l'environnement physique* : Il se rapporte à une dimension du monde physique ou des phénomènes tels que l'emplacement, le temps, la température, les précipitations et le niveau de la lumière, etc.
- *contexte humain (le contexte de l'utilisateur ou le contexte de la personne)*: l'interaction est conditionnée par les utilisateurs en termes d'identité, préférences, exigences de la tâche, contexte social et autres activités, expérience de l'utilisateur et la connaissance et types d'utilisateurs.
- *contexte des ICT (Information and Communication Technology) ou le contexte de l'environnement virtuel*: un composant particulier dans un système distribué est conscient des services qui sont disponibles à l'intérieur et à l'extérieur, localement et à distance, dans le système distribué.

2. Notion de sensibilité au contexte

Les systèmes sensibles au contexte utilisent les informations contextuelles de l'environnement dans les systèmes informatiques. L'objectif de ces systèmes n'est pas de soutenir l'ubiquité globale, mais plutôt de soutenir l'ubiquité basée contexte, c'est-à-dire [Stefan, 2009]:

(1) la limitation des ressources nécessaires pour fournir des services omniprésents car offrant des services omniprésents seraient un coût prohibitif,
(2) limitant le choix de l'accès de tous les services possibles aux seuls services utiles,

(3) en évitant de surcharger l'utilisateur avec trop d'informations et la prise de la décision,

(4) soutenir un lieu naturel de l'attention et la prise de la décision calme par les utilisateurs.

Les systèmes sensibles au contexte sont des systèmes qui ont conscience de leurs situations (contexte) de l'environnement physique, virtuel (Information Communication Technology ICT) et de l'utilisateur, et qui peuvent adapter le système aux situations détectées [Stefan, 2009].

La sensibilité au contexte ou 'Context-awareness' est un terme qui est apparu dans les travaux sur l'informatique pervasive, ou l'informatique ubiquitaire. Il a été utilisé pour la première fois en 1994 par Schilit et Theimer [Schilit, 1994] pour se rapporter à un système qui utilise le contexte pour offrir des informations et des services pertinents pour l'utilisateur et l'application. Sabler et al., dans [Salber, 1998], définissent la sensibilité au contexte comme étant la meilleure capacité d'un système à agir en temps réel avec des données provenant du contexte. Il y'a beaucoup d'autres définitions similaires. Par exemple, Dey dans [Dey-b, 2001] stipule qu'un système est sensible au contexte s'il utilise le contexte pour offrir des informations ou des services pertinents pour l'utilisateur, où la pertinence dépend de la tâche de l'utilisateur. Les applications sensibles au contexte sont des applications dont le comportement peut varier en fonction du contexte.

2.1. Architecture abstraite d'un système sensible au contexte

Les points à considérer pour le développement des systèmes sensible au contexte sont liés aux trois phases : capture, raisonnement et action. La première doit prendre en compte les situations à reconnaitre et les informations du contexte qui peuvent être disponibles y compris les capteurs à utiliser. La technique de raisonnement appropriée est alors un choix à faire entre les règles Event-Condition et les techniques d'intelligence artificielle. La plupart des données extraites avec le raisonnement peuvent être enregistrées. Finalement, des effecteurs appropriés, matériels et logiciels sont employés[Loke, 2005].

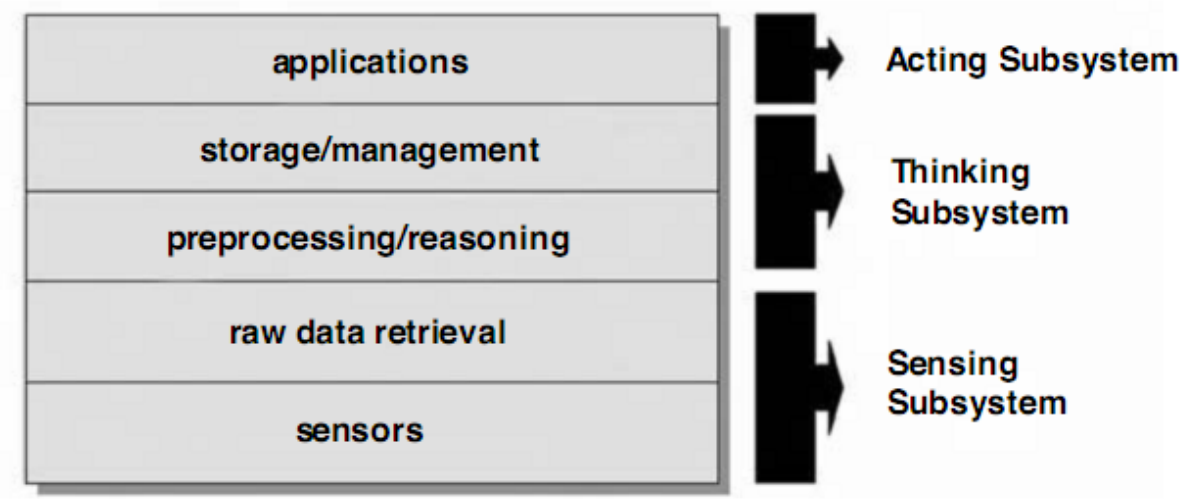

Figure 1.1: Architecture abstrait des systèmes sensibles au contexte [Baldauf, 2004].

Différentes architectures ont été décrite dans la littérature pour les systèmes sensibles au contexte. La figure 1.1 montre une architecture abstraite à plusieurs niveaux, qui est divisée en sous-systèmes. Dans cette architecture:

- Le niveau “sensors”, qui est le niveau le plus bas, définit les capteurs utilisés pour capturer les données utilisées par le niveau « raw data retrieval »,
- Le niveau « preprocessing/reasoning » traite l’information capturée, et les résultats sont les informations du contexte,
- Le niveau Storage / Management dépend des besoins de l’application ou l’utilisateur.

2.2. Sensibilité au contexte Active et Passive

L’objectif de la conception d’un système sensible au contexte est de minimiser le degré du contrôle de l'utilisateur pour adapter le produit aux changements du contexte. On trouve deux types de systèmes conscients du contexte (actif ou passif)[Stefan, 2009].

- ***Système conscient au contexte actif* :**

Le système Ubicom est conscient du contexte de l'environnement pour le compte de l’utilisateur, en ajustant automatiquement le système au contexte sans que l'utilisateur s'en rende compte. Cela peut être utile dans les applications où il y a des contraintes de temps strictes et l'utilisateur ne sera pas en mesure de s'adapter au contexte assez rapidement. Un exemple de ceci est un système d'évitement de collision intégré à un véhicule pour freiner automatiquement lors de la détection d'un obstacle en face d'elle.

- ***Système conscient au contexte passif* :**

Le système Ubicom est conscient du contexte de l'environnement pour le compte de l'utilisateur. Il rend compte de tout le contexte actuel à l'utilisateur, sans aucune adaptation Par exemple, un système de positionnement signale l'emplacement d'un objet en mouvement sur une carte.

3. Lignes de Produits Logiciels

Ces dernières années, les besoins des clients et leurs exigences sont multipliés Cela conduit les entreprises à changer leurs méthodes de production. Dans le domaine d'automobile, *Ford* a été le premier qui a inventé *la ligne de production* qui permet de couvrir un marché vaste avec le moindre coût. L'approche Lignes de produits logiciels (SPL) est une transposition des chaînes de production au monde du logiciel. Son principe est de minimiser le coût de construction de logiciels dans un domaine particulier, en ne développant plus chaque logiciel séparément, mais plutôt en le concevant à partir d'éléments réutilisables, appelés '*assets*' Un *asset* peut ainsi être une exigence, un modèle, un composant, un plan du test ou tout simplement un document.

La Figure 1.2 présente les coûts accumulés nécessaires pour le développement de *n* systèmes différents. La ligne continue représente les coûts de développement des systèmes séparément, pendant que, la ligne discontinue représente les coûts pour l'ingénierie de ligne de produits. Dans le cas de peu de systèmes, SPL est relativement haut, et se baissant pour des quantités larges. Les deux courbes se croisent dans le point 'break-even'. A ce point, le coût est le même pour le développement des systèmes séparément ou par SPL.

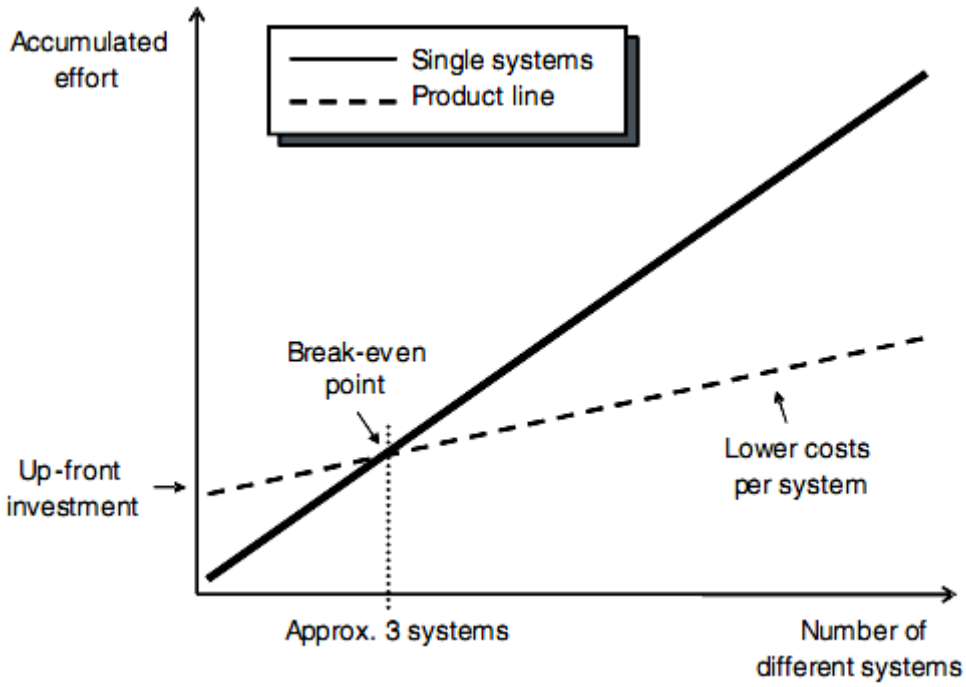

Figure 1.2: L'économie dans l'ingénierie de ligne de produit logiciel.

3.1. Définition

L'ingénierie de *SPL*, ou « Program Family », est un ensemble de programmes dont les propriétés communes sont nombreuses que c'est avantageux d'étudier d'abord ses propriétés communes avant d'analyser les membres individuels de l'ensemble [Parnas, 1976]. SPL est une famille de produits conçue pour prendre l'avantage de leurs aspects communs et de prévoir la variabilité [Weiss, 1999].

L'intérêt de l'approche SPL est de développer des artefacts logiciels génériques qui sont assez flexible pour couvrir en même temps divers besoins du client [Anastassopoulos, 2005]. L'idée de base de cette approche est de séparer les éléments communs d'une famille de produits de ses différences entre les produits. Les éléments communs servent de plateforme pour tous les produits de la même famille.

3.2. Principes de l'approche d'ingénierie de ligne de produits logiciels

Le processus d'Ingénierie des Lignes de Produits est basé sur le développement pour la réutilisation et le développement avec réutilisation, voir Figure 1.3.

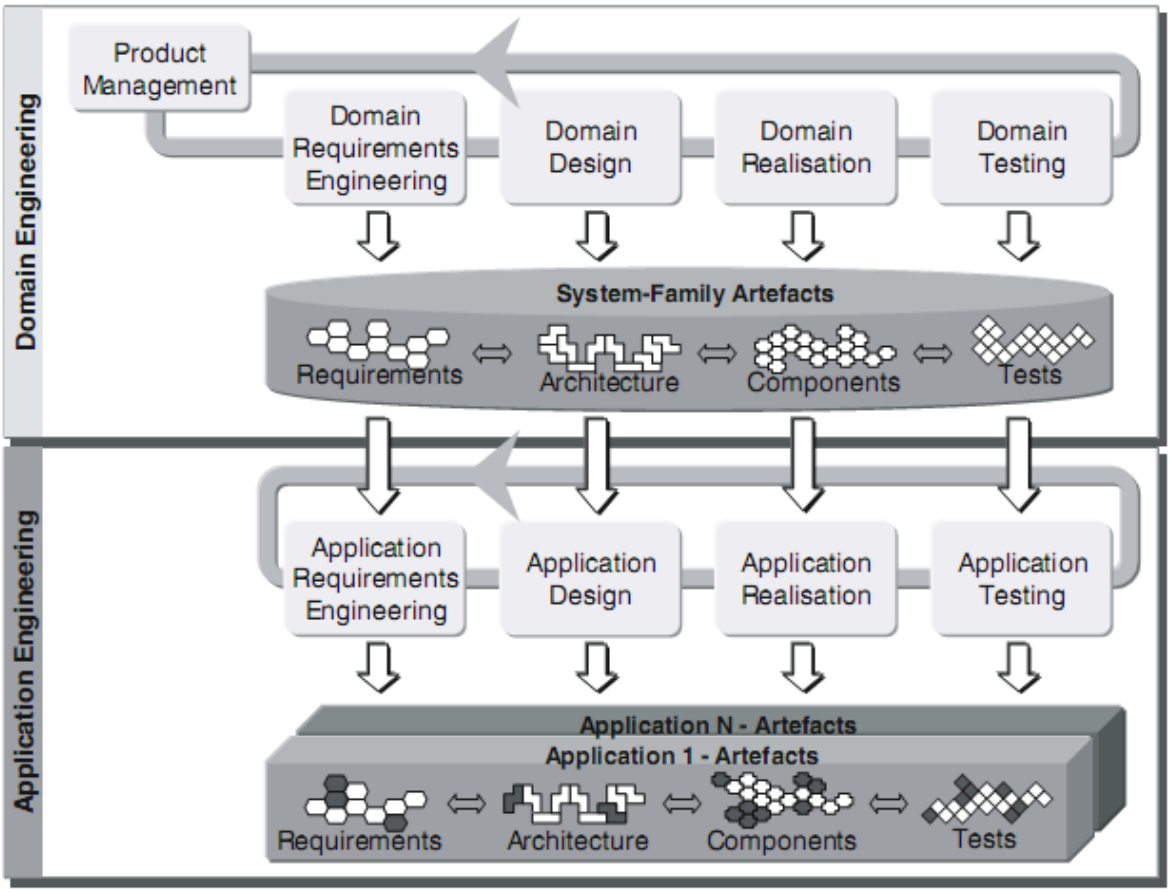

Figure 1.3: les deux cycles de vie de model d'ingénierie de ligne de produit logiciel [Frank, 2007]

Dans la figure 3.1, on trouve :

- Ingénierie du domaine (développement pour réutilisation),
- Ingénierie d'applications (développement avec réutilisation)

La première sert comme une base pour le développement effectif des produits individuels. Au contraire des approches traditionnelles, SPL couvre tous les artefacts logiciel (assets) qui ont des relations dans les étapes du cycle de vie de la phase spécification des besoins jusqu'à l'architecture et l'implémentation au teste. Les différents assets eux aussi peuvent contenir des variabilités explicites. La deuxième (ingénierie de l'application) est dirigée par l'infrastructure de ligne de produits, qui contient la plupart des fonctionnalités nécessaires. La variabilité modélisée explicitement sert comme une base pour dériver un produit individuel. Les besoins sont constitués et catégorisé (communs et variabilité) comme une partie de départ d'une SPL

pour le nouveau produit (*Product Configuration*). Puis les différentes assets sont instanciés d'une façon correcte pour produire une version initiale (*Product Derivation*).

3.3. Variabilité

Rares sont les éléments logiciels directement réutilisables. Avant de pouvoir réutiliser un élément logiciel, il est nécessaire d'identifier sous quelles conditions celui-ci peut être effectivement réutilisé et comment. Plus l'élément pourra être réutilisé dans des contextes différents, moins son utilisation sera soumise à des conditions et plus sa réutilisabilité augmentera. De manière simplifiée, chaque produit appartient à une ligne de produits logiciels détermine un contexte particulier.

Chaque produit SPL se compose d'un ensemble d'éléments logiciels représentés par des « caractéristiques » ou « features ». Une caractéristique regroupe les exigences devant être satisfaites par les produits logiciels finaux intégrant cette feature. L'objectif d'une feature est de délimiter un ensemble d'exigences fortement connexes et directement réutilisables par différents produits finaux. Les features sont successivement décomposées en sous-features en obtenant des features terminales. Idéalement, chaque feature terminale est associée à un élément logiciel réutilisable (composant, service…) implémentant les exigences déterminées par la feature correspondante. En outre, ces features permettent de distinguer les produits les uns des autres.

L'analyse de la variabilité permet d'identifier les features, de spécifier les contraintes les reliant et d'expliciter les alternatives offertes lors de la sélection (réutilisation) des features. Cependant, de nombreuses sources de variation existantes peuvent apparaître durant toutes les phases du développement logiciel [Jean, 2009].

3.3.1. La gestion de la variabilité :

L'objectif SPL (Software Product line engineering) est de supporter une gamme vaste des produits destinés à chaque type de clients ou à la totalité de différentes parties du marché. On parle de *variabilités* qui existent entre les produits. Ces variabilités ont besoin d'être définies représenter, exploité et implémenter, etc. d'une autre façon *la gestion de la variabilité* [Frank, 2007]. La gestion de la variabilité permet de déterminer sous quelles conditions les *assets* peuvent être réutilisées de manière optimale.

3.3.2. Les types de variabilité:

Avant la gestion de la variabilité il faut les regrouper dans trois(03) groupes [Frank, 2007].

1. Le noyau ou les éléments communs (*Commonality*): sont les caractéristiques (*Features*) qui sont communes, partagées ou existées dans tous les produits de SPL.
2. Variabilité (*Variability*): sont les caractéristiques qui désignent un sous-ensemble de produit, leurs sélection créée la déférence entre les produits.
3. Spécifique à un produit (*Product-specific*): une ou plusieurs caractéristiques qui peuvent s'ajouter à un produit au dernier moment (Ingénierie d'applications) pour satisfaire des besoins spécifiques à un client.

Durant le cycle de vie de la ligne de produits des caractéristiques peuvent s'ajouter ou de changer de nature, Figure 1.4.

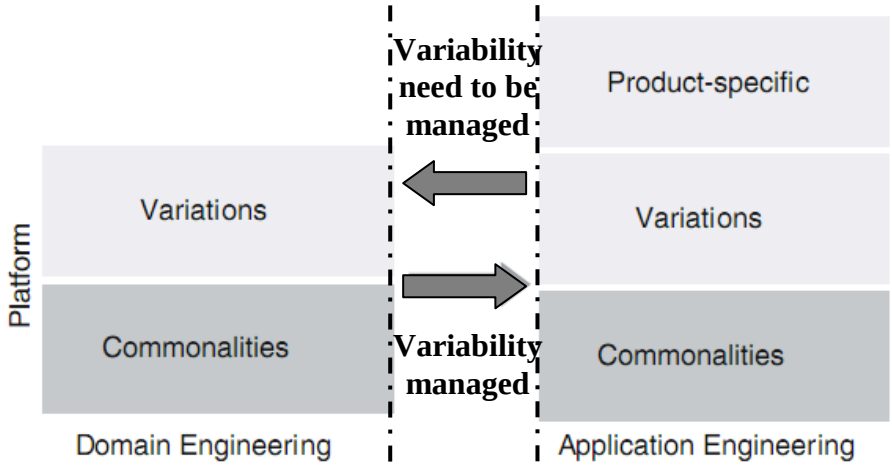

Figure 1.4: la relation entre les différents types de variabilités[Frank, 2007].

3.3.3 Représentation de la variabilité

La représentation de la *variabilité* est nécessaire pour faciliter sa gestion (*Ingénierie du domaine*) et la composition des assets (*Ingénierie d'applications*).

1- ***Point de variation*** (*Variation point*) : elle décrit les différences qui existent entre les produits finales.

2- ***Variant***: les différentes possibilités qui existent pour satisfaire un point de variation.

3- ***Dépendances entre les Variabilités*** (*Variability dependencies*) : utilisées pour contrôler les différents choix (*variants*).

4- ***les Contraints de dépendances*** (Constraint dependencies): elle décrit les dépendances de sélection de certaines *variantes*. Il existe deux formes:

a- ***Requires***: la sélection d'un variant spécifique peut nécessite la sélection d'un ou plusieurs *variante* (peut-être pour différents points de variation).

b- ***Excludes***: La sélection d'un variant spécifique peut interdire la sélection d'autres *variante* (peut-être pour différents points de variation).

4. Ligne de produits logiciels statiques

Une ligne de produits traditionnelle des systèmes sensibles au contexte à comme objectif de dériver un produit conforme aux exigences de l'utilisateur et à l'environnement d'exécution. Ces exigences doivent être déterminées durant le processus de configuration pour pouvoir sélectionner les features conformes aux besoins des clients et le contexte d'exécution (la plateforme d'exécution, l'environnement physique, la langue de l'utilisateur, etc.). Le résultat de cette ligne de produits est un produit conforme à l'exigence définie au moment de conception. L'inconvénient de cette approche et qu'une fois le produit est dérivé il est impossible de modifier son comportement ou sa configuration au moment d'exécution.

Les lignes de produits traditionnels ne prennent pas en compte la modification à l'exécution des applications. Elle consiste à dériver plusieurs versions pour chaque nouveau contexte identifié. Le produit est adapté au moment de dérivation avant de pouvoir le réutiliser dans son nouveau contexte. Cette technique (SPL statique) a été adoptée pour plusieurs développements. Elle présente une solution simple et efficace, car à chaque dérivation on s'occupe à un seul contexte identifié. Cependant, elle présente l'inconvénient de la difficulté pour la prise en charge de nouveaux contextes d'utilisation (environnement dynamique).

5. Ligne de produits logiciels dynamique

Dans les systèmes d'informatiques pervasifs, l'application doit offrir ses services aux utilisateurs dans les différentes conditions afin d'accomplir ses besoins. Cette capacité pour s'évoluer au moment d'exécution (reconfiguration dynamique) représente un défi pour les constructeurs des applications sensibles au contexte. Une ligne de produits logiciels dynamiques DSPL offre aux systèmes (applications) un mécanisme pour reconfigurer eux-mêmes au moment d'exécution [Brice, 2008]. Un DSPL est un SPL traditionnel [Hallsteinsen, 2008] mais qui contient un ensemble de configuration qui dépend à chaque état de situation

pour assurer une adaptation au changement du contexte [Morin, 2008]. Dans DSPL chaque feature sensible au contexte doit être reliée au contexte qui convient [Núñez, 2009].Un DSPL a comme objectif la dérivation des produits qui peuvent s'adapter au moment d'exécution au changement des besoins d'utilisateurs et les conditions de l'environnement [Parra-b, 2011].

Toutes les approches qu'utilisent DSPL, à comme objective la dérivation des produits qui adaptent leurs architectures dynamiquement aux différentes situations du contexte selon des règles et des techniques employées permettant la modification de la configuration des applications durant l'exécution aux différentes situations du contexte.

6. Les systèmes informatiques autonomes

L'autonomie concerne la propriété d'un système qui lui permet de contrôler son action d'une façon indépendante. Un système autonome est en interaction avec d'autres systèmes et environnements afin de contrôler ses actions. Un système autonome est défini comme un système autogouverné, qui est capable de prendre des décisions et exécutées des actions d'une façon indépendante. Il est orienté but et fonctionne essentiellement afin d'appliquer une stratégie ou d'accomplir un objectif. Il ya plusieurs types des systèmes autonomes sur l'internet. Un système autonome est un système qui est dirigé par une politique de routage pour un ou plusieurs réseaux contrôlé par le même administrateur derrière une seule entité administrative. Un système d'agent logiciel est caractérisé comme système autonome [Stefan, 2009].

Les systèmes autonomes peuvent être désignés, c'est-à-dire que leurs buts sont assignés d'une façon automatique (peut-être par l'utilisateur). Ainsi, plutôt que l'utilisateur a besoin d'interagir et de contrôler chaque tâche d'interaction de bas niveau, l'utilisateur a besoin seulement d'interagir pour spécifier les tâches de haut niveau (buts). Alors le système lui-même planifie automatiquement l'ensemble des tâches de bas niveau et les ordonnancées automatiquement pour accomplir le but, pour réduire la complexité au l'utilisateur. En cas d'échec pour atteindre le but le système réplanifie le plan ou l'ordonnancement des tâches. Le problème de planification est résolu avec l'utilisation de l'intelligence artificielle [Stefan, 2009].

6.1. Intelligence

L'intelligence peut permettre aux systèmes d'agir de façon plus proactive et dynamique afin de supporter les différents comportements dans les systèmes ubiquitaires: Agir de façon plus proactive, dynamiquement et humainement à travers: [Stefan, 2009].

- Modélisation de son environnement physique: un système intelligent (IS) peut harmoniser son comportement pour agir plus efficacement en prenant en compte un modèle de changement de son environnement lorsqu'il s'agit de décider comment il doit agir.
- basée-Objectif / planification : Modélisant et limitant son environnement humain: il est utile pour un IS d'avoir un modèle de l'humain afin de mieux supporter iHCI. Un IS pourrait permettre aux humains d'être en mesure de déléguer des objectifs de haut niveau pour le système plutôt que d'interagir avec en précisant les tâches de bas niveau nécessaires pour accomplir l'objectif.
- Manipulation incomplète: les systèmes peuvent aussi être incomplets, car les environnements sont ouverts aux changements parce que les composants du système peuvent échouer. AI peut supporter la re-planification pour présenter des plans de rechange. Une partie du système peut être partiellement observable. La connaissance incomplète de l'environnement d'un système peut-être complété par un raisonnement du type AI sur le modèle de son environnement afin d'en déduire ce qu'il ne peut pas savoir qu'il se produira.
- Gérer l'incertitude : Manipulation du comportement non déterministe: les systèmes Ubiquitaires peuvent fonctionner dans des environnements de services ouverts et dynamiques. Les actions et les objectifs des utilisateurs peuvent ne pas être complètement déterminés. La conception du système peut avoir besoin de supposer que leur environnement est un milieu semi-déterministe (appelé aussi environnement du système volatil) et être conçu pour gérer cela. Les systèmes intelligents utilisent des modèles explicites pour gérer l'incertitude.
- interaction basées sémantiques : les systèmes Ubiquitaires sont également susceptibles de fonctionner dans des environnements ouverts et hétérogènes. Les types de systèmes intelligents définissent des modèles puissants pour supporter l'interopérabilité entre des systèmes hétérogènes et de leurs composants, par exemple, l'interaction basée sémantique.

6.2. Adaptation

Dans les systèmes d'informatique pervasive, l'adaptation est la propriété qui caractérise un système capable de changer son comportement lui-même en fonction des changements produits dans son environnement, afin d'améliorer ses performances ou de continuer son exécution [Chaari. 2007]. L'adaptation est fortement associée à un processus qui modifie le comportement d'une application qui inclut : détecte les événements et les informations qui permettent de savoir s'il ya des changements, planification de l'ensemble de changements et d'accomplir ces changements sur l'application. La référence la plus connue pour ce modèle est qui a été présenté par IBM [IBM, 2006] est MAPE pour les phases qui incluent : Monitor, Analyze, Plan, Execute.

6.2.1 Feedback Control Loops (FCLs)

Les systèmes autonomes sont caractérisés par la capacité d'observer son environnement et de détecter les situations d'adaptation. Ces systèmes autocontrôles (*self-management systems*) peuvent accomplir ses fonctions sans intervention humaine, l'ensemble des propriétés qui caractérise les systèmes autocontrôles sont : [Jeffrey, 2003] [Salim, 2006] [Daniel, 2007].

- Configuration autonome (*Self-configuration*): concerne les systèmes qui sont capables d'adapter leurs comportements au contexte d'exécution sans l'interruption complète des différents services.
- Optimisation autonome (Self-optimisation) : sont les systèmes qui doivent contrôler et observer les ressources consommées. À chaque dégradation d'un service, une reconfiguration est nécessaire par ces systèmes pour maintenir la performance et l'efficacité.
- Entretien autonome (Self-healing) : la détection, l'analyse, la prévention, et la résolution des problèmes sont gérés par le système lui-même. Le système doit surmonter l'échec du matériel et logiciel.
- Protection autonome (Self-protecting) : le but est l'anticipation, identification, détection et la protection contre les menaces d'une façon automatique.

Ces quatre propriétés groupent les préoccupations principales qui sont associées à la maintenance de système.

La Boucle MAPE-K

La référence standard d'IBM initiative informatique autonome, codifie un extérieur approche de FCL qui comporte: moniteur -Analyse-Plan-Exécute le Model (MAPE) [IBM, 2006]

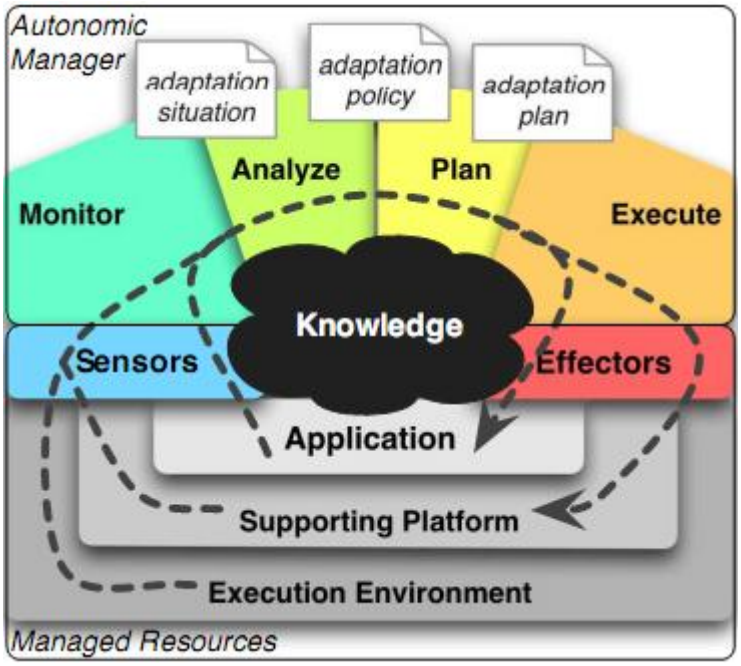

Figure 1.5: présentation générale d'un Model MAPE autonome [IBM, 2006].

- Le moniteur (The Monitor) fourni un mécanisme qui collecte, regroupe, filtre et signale l'information (mesures et topologies) collecté à partir de gestionnaire de ressources.
- L'analyse (The Analyze) contient le mécanisme qui suit et modélise les situations complexes d'adaptation.
- Le Plan (The Plan) c'est une fonction qui regroupe les mécanismes de construction des actions nécessaires pour accomplir le but et l'objectif. La planification utilise les informations de politique d'adaptation pour guider leurs travaux.
- L'exécution (The Execute) c'est une fonction qui groupe les mécanismes pour contrôler l'exécution de Plan d'adaptation avec la considération des mises à jour dynamiques.

Dans le modèle de MAPE-K Loop présenté dans la figure 1.5, les connaissances ou *Knowledge* (symptômes, politiques) sont des données partagées entre les activités moniteur, l'analyse, Le Plan et l'exécution. Les connaissances doivent être complétées au moment d'exécution.

7. La relation entre l'informatique autonome et l'informatique sensible au contexte

L'informatique autonome se focalise sur le développement des applications qui contrôlent leurs actions d'une façon indépendante. Cette propriété (autonomie) réduit les responsabilités de l'administrateur de système, qui interagit seulement pour spécifier les tâches de haut niveau (buts). Ce type de systèmes identifie clairement différentes tâches qui composent le processus d'adaptation aussi, les données requises pour les différentes tâches. Comme n'importe quel processus, la circulation de l'information est essentielle pour aboutir chaque but d'adaptation.

L'informatique sensible au contexte propose le développement d'applications qui ont la capacité de réagir automatiquement au changement de l'environnement. Dans ces systèmes, l'information du contexte est la clé d'adaptation. Les règles peuvent être aussi considérées comme informations du contexte. Dans l'informatique autonome, l'adaptation est déduite selon la situation actuelle. Donc, les deux types de systèmes sont des approches complémentaires [Klein., 2008].

En particulier, les applications sensibles au contexte peuvent être conçues suivant le principe de ***FCLs***. Les données échangées dans les différentes étapes de la boucle correspondent aux informations du contexte collectées à partir de l'environnement. Dans chaque phase de ***FCL***, ces informations sont traitées afin de produire des informations de haut niveau qui seront utilisées dans l'étape suivante. Le résultat de FCL est l'exécution de l'adaptation basée-contexte des applications exécutées dans un environnement avec une intervention humaine minimale [Francisco, 2011].

8. objectif :

Les produits sensibles au contexte dérivés par un SPL, doivent adapter leurs comportement aux différentes situations du contexte d'une manière autonome. C'est-à-dire, elles doivent observer son environnement et de détecter les situations d'adaptation. Dans une autre façon, la variabilité des configurations (comportements, adaptation) dépend de situation de l'environnement (utilisateurs, physique, virtuelle), où chaque situation détermine une configuration ou un sous ensemble de configurations. Déterminé une situation se fait à partir d'un ou plusieurs capteurs (matériel, ou logique) fournis par l'appareil qu'exécute l'application. Une configuration est un sous ensembles de caractéristiques sélectionnées, composées et liées pour produire un comportement spécifique au besoin des utilisateurs et au contexte déterminer.

Notre objectif dans ce mémoire est de proposer un DSPL autonome « ASPL » pour dériver des produits auto-adaptatifs. Afin d'atteindre cet objectif selon le principe de l'informatique autonome, il faut :

- Définition des connaissances : L'utilisation de modèle de caractéristiques FM de SPL pour déterminer les différentes situations du contexte, et la variabilité des configurations liées à chaque situation du contexte.
- Identifier un algorithme d'optimisation afin de gérer un nombre important du plan de reconfiguration lié à chaque situation du contexte.
- L'utilisation des réseaux de neurones pour générer des reconfigurations lors de l'exécution.

Conclusion :

Dans ce chapitre on a introduit brièvement les systèmes sensibles au contexte, le principe de l'ingénierie de ligne de produits logiciels et les systèmes autonomes.

Avant d'entamer la présentation de notre approche pour la reconfiguration des applications au contexte selon le principe de l'ingénierie de ligne de produits logiciels SPL, nous présentons les différents travaux classiques et SPL liés au problème de sensibilité au contexte.

Chapitre II
Approches non-SPLs pour les systèmes sensibles au contexte

Sommaire

Introduction

Dans la littérature, on trouve plusieurs thèmes qui traitent le développement et l'adaptation des systèmes sensibles au contexte. Les différentes approches utilisent le domaine de l'ingénierie logiciel à base de composants (*Component-Based Software Engineering*), pour le développement ainsi que pour l'adaptation [Szyperski, 1998]. Cette dernière est assurée par l'ajout ou la suppression de composants d'une façon statique au moment de la conception ou dynamique (à l'exécution) [Philip,S, 2004]. De plus, il ya d'autres approches qui s'intéresse à la modélisation du contexte avec [Strang-a, 2004] :

(1) avec Modèles de valeurs de clés (*Key value models*),
(2) les systèmes de balisage (*markup schemes*),
(3) Modèles graphique (*graphical models*),
(4) des ontologies,

(5) des modèles orientés objet,

(6) les modèles logiques.

Ces travaux concentrent sur tous les aspects de capture, d'interprétation, de modélisation, de stockage et de dissémination du contexte. Malgré cela, il y a beaucoup moins de travaux qui s'intéressent à l'adaptation et à l'actualisation de l'application selon le contexte. Dans ce chapitre, nous présentons les travaux existants pour la conception de plateformes de déploiement et de développement d'application sensibles au contexte.

1. Modèles de sensibilité au contexte

.Les modèles du contexte proposés définissent un ensemble de six familles, dont les caractéristiques distinguent différents styles d'architectures numériques pour la capture du contexte, des plus simples associations par paires « clés-valeurs » aux plus complexes « modèles ontologiques »[Strang-a, 2004].

1.1 Les modèles à paires clés-valeurs

Cest une structure simple pour la modélisation des informations contextuelles. Ils associent la valeur d'un objet du contexte, comme la localisation, à une variable du système d'information. Cette structure est simple à implémenter et à maintenir, mais elle n'est pas structurellement assez riche pour inférer des relations entre les multiples dimensions du contexte. Les modèles du contexte « Context toolkit » [Dey-a, 2001] et « CMF » [Korpepaa, 2004] sont des exemples issus de cette famille.

Les modèles pairs clés-valeurs sont des solutions simples pour modéliser le contexte. La simplicité de cette méthode a permis de mettre en place des systèmes efficaces de découverte dynamique des services [Samulowitz, 2001].

1.2 Les modèles logiques

Ils appliquent en général les formalismes mathématiques de la logique des propositions et/ou de la logique des prédicats à la description des dimensions du contexte de leurs relations. Ils formalisent la notion du contexte et définissent ses dimensions par des propositions articulées logiquement avec un ensemble de règles. Les propositions contextuelles sont ajoutées ou supprimées du modèle logique par la perception de nouveaux faits issus soit par des objets du contexte mesurés, ou soit par d'autres propositions du modèle. McCarthy propose une première modélisation formelle du contexte [McCarthy, 1993], ceux de Akman et Surav [Akman, 1997] qui réutilisent la théorie des situations, ou ceux plus récents, de Kaenampornpan [Kaenampornpan, 2004] qui propose une théorie des activités.

1.3 Les modèles orientés objet

Portent au niveau de la description du contexte les notions de classe, d'héritage, d'interface, etc. La plupart des approches existantes, considèrent les mesures du contexte comme des objets auxquels le système accède par des interfaces spécifiques. Les détails des méthodes de perception du contexte sont encapsulés dans les objets eux-mêmes et cachés du reste du

système. Les modèles d'objet du contexte s'intègrent facilement dans le développement de systèmes d'information basés sur une conception orientée objet.

Ces approches ont également été proposées à l'aide d'UML [Sheng, 2005], mais également à l'aide du langage ORM (Object Role Modeling) [Henricksen, 2002] qui s'avère très efficace pour générer des modèles relationnels et les bases de données qui leur sont associées.

1.4 Les modèles à langage de description

Ils structurent hiérarchiquement les objets du contexte dans un modèle de données à partir de balises d'attributs et de valeurs. En général, le langage de balisage, est choisi pour exprimer les dimensions du contexte, qui se définit par rapport à un autre langage de description, communément de la famille XML (eXtended Markup Language). Il est ainsi possible de passer d'un modèle du contexte à un autre, en appliquant des règles de transformation de schémas XML. Par exemple, par l'utilisation de feuilles de styles XSL. La description d'un profil d'utilisateur est une utilisation courante des modèles langage de description. Plusieurs niveaux hiérarchiques distinguent, par exemple, ses adresses, ses activités ou encore ses préférences d'utilisation.

1.5 Les modèles graphiques

Ils proposent les différentes sémiologies graphiques pour la représentation des dimensions du contexte, des objets, et leurs relations. En général, ces modèles facilitent la compréhension de la composition et des relations des éléments du contexte par un langage graphique aisément lisible et modifiable par les concepteurs. Suivant le modèle adopté, l'application des règles de transformation automatiques ou semi-automatiques permet de passer de la représentation graphique à un modèle entité relation définissant une structure de données informatique. UML (Unified Modeling Language), est l'un des langages de modélisation graphique les plus connus, est souvent utilisé pour exprimer les relations entre différents objets ou groupes d'objets du contexte.

1.6 Les modèles ontologiques

Ils décrivent le contexte comme un ensemble de concepts liés par des relations. Les ontologies ont montré une capacité réelle pour modéliser des phénomènes de la vie réelle. Elles proposent des outils de formalisation et de raisonnement, qui sont appliqués à la modélisation contextuelle, permettent d'inférer de nouvelles données contextuelles, d'un niveau d'abstraction supérieur, pour réagir à la connaissance d'un contexte dans son ensemble

plutôt que de limiter le comportement adaptatif à des actions de mesure et de correction. Exemple CoBrA [Chen, 2004].

2. Principales plateformes existantes de sensibilité au contexte

Dans [Chaari, 2007], l'auteur présente quatre implémentations principales de plateformes sensibles au contexte :Contexte Toolkit,Contexte Broker Architecture (CoBrA), Context Management Framework (CMF), Service oriented context-awar middleware (SOCAM) plus on représente son architecture SECAS**.**

2.1. Context Toolkit

Contexte Toolkit [Dey-a, 2001], est une architecture qui supporte le développement et le déploiement d'applications sensibles au contexte. Elle offre une boite à outils et des composants logiciels. Le nom de l'outil Context-toollkit vient de l'analogie avec les bibliothèques graphiques de widgets (composants graphiques). Comme pour les « toolkits » graphiques qui offrent un modèle computationnel et des widgets d'utilité publique (menu, formulaire, button), L'objectif de « Context-Toolkit », est d'offrir un modèle d'exécution et des composants de base réutilisables pour les applications sensibles au contexte.

Context-Toolkit comporte cinq catégories de composant (voir figure 2.1), qui implémente des fonctions distinctes. « Context-widgets » acquiert les informations du contexte. « Interpreters » transforme et augmente le niveau d'abstraction de l'information du contexte. « Interpreters » peut combiner plusieurs morceaux du contexte pour produire un niveau plus haut d'information du contexte. « Aggregators » prend l'information du contexte liée à une entité pour faciliter l'accès par l'application. « Services » exécute le comportement sur l'environnement avec l'utilisation du contexte acquis. Finalement, « Discoverers »permet à l'application (et autres composants) de déterminer la capacité de l'environnement, Il maintient une vue globale de 'frame-work', qui concerne la disponibilité des composants (widgets, interpreters, aggregators, et services) utilisés par l'application.

Quand un composant est exécuté, il notifie le « Discover » de sa présence et sa capacité. « Widgets » indique le genre du contexte qui peut fournir. « Interpreters » indique l'interprétation qui peut accomplir. « Aggregators » indique les entités représentées et leurs types du contexte fournis. « Services » indique le service de sensibilité au contexte à fournir et les types d'informations du contexte requis pour son exécution. Quand n'importe quel composant a échoué, c'est la responsabilité de « discover » pour déterminer le mauvais composant.

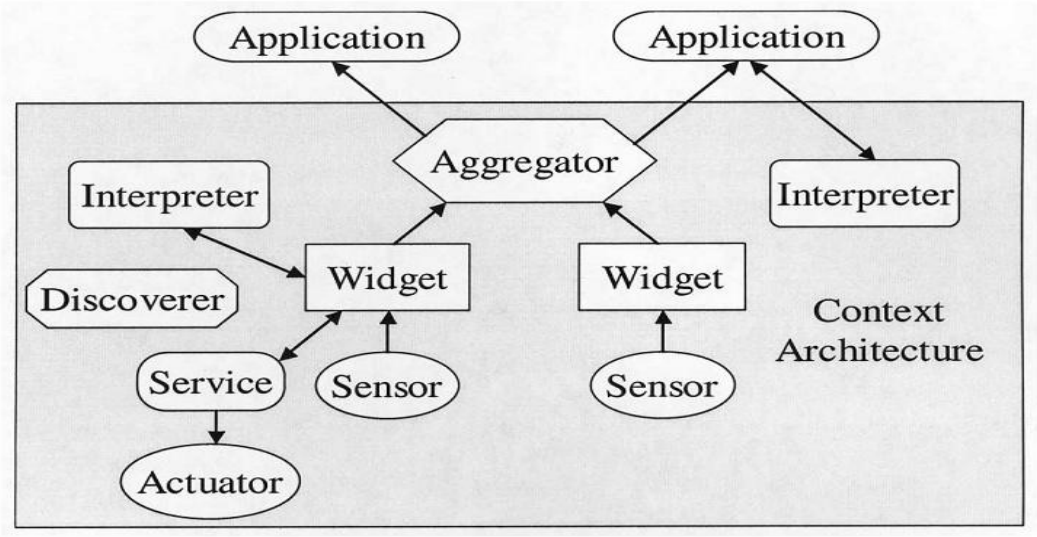

Figure 2.1 : Exemple des composants du contexte Toolkit [Dey-a, 2001].

2.2. Contexte Broker Architecture (CoBrA)

Context broker architecture (CoBrA) [Chen, 2004], est une architecture orientée agents qui vis à aider les appareils, services et agents afin de devenir sensibles au contexte dans un espace intelligent. CoBrA utilise 'Semantic Web languages' tel que 'RDF' et Web Ontology Language 'OWL' pour définir les ontologies du contexte. Un élément nommé '*context broker' (resource-rich agent),* est utilisé pour gérer et maintenir un modèle partagé du contexte. Le *context broker* peut déduire les informations du contexte (exemple, les intentions, le rôle et les tâches de l'utilisateur) qui sont difficile à capturer par les capteurs physiques. aussi il peut détecter et résoudre les informations incohérentes qui se produisent souvent comme des résultats imparfaits des capteurs.

CoBrA offre un langage de contraintes qui permet aux utilisateurs de contrôler ses informations contextuelles. Basé sur les contraintes définies par l'utilisateur, le '*broker'* contrôle dynamiquement la granularité de l'information de l'utilisateur qui va être partagée, et choisit le destinataire approprié pour recevez les notifications de changement du contexte de l'utilisateur.

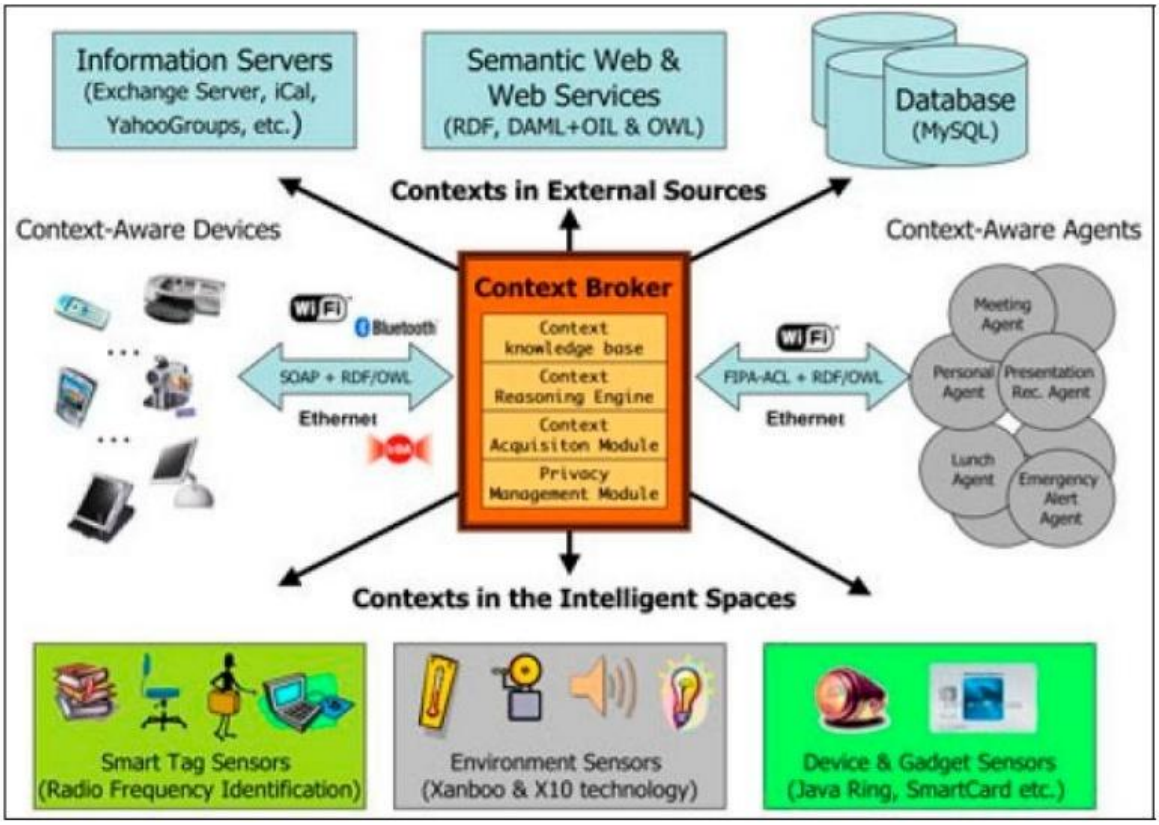

Figure 2.2 : Architecture globale du système CoBrA[Chen, 2004].

La Figure 2.2, présente l'architecture conceptuelle de CoBrA. Le 'Contexte brocker' est un serveur spécialisé de l'entité qui s'exécute sur des ordinateurs immobiles riches en ressources dans l'environnement. Toutes les entités informatiques dans l'espace intelligent sont présumées avoir une information a priori qui concerne la présence de 'Context Brocker', et un agent de haut-niveau présumé pour communiquer avec d'autre 'Brocker'.

2.3. Context Management Framework (CMF)

Le context management framework (CMF) [Korpepaa, 2004], est une plateforme permettant la reconnaissance sémantique du contexte en temps réel, en présence des informations incertaines. Elle est composée de quatre entités principales :

(1) *resources server* (capteur de contexte),
(2) *contexte recognition service* (interprétation du contexte),
(3) *context manager* (dissémination du contexte)
(4) *application*.

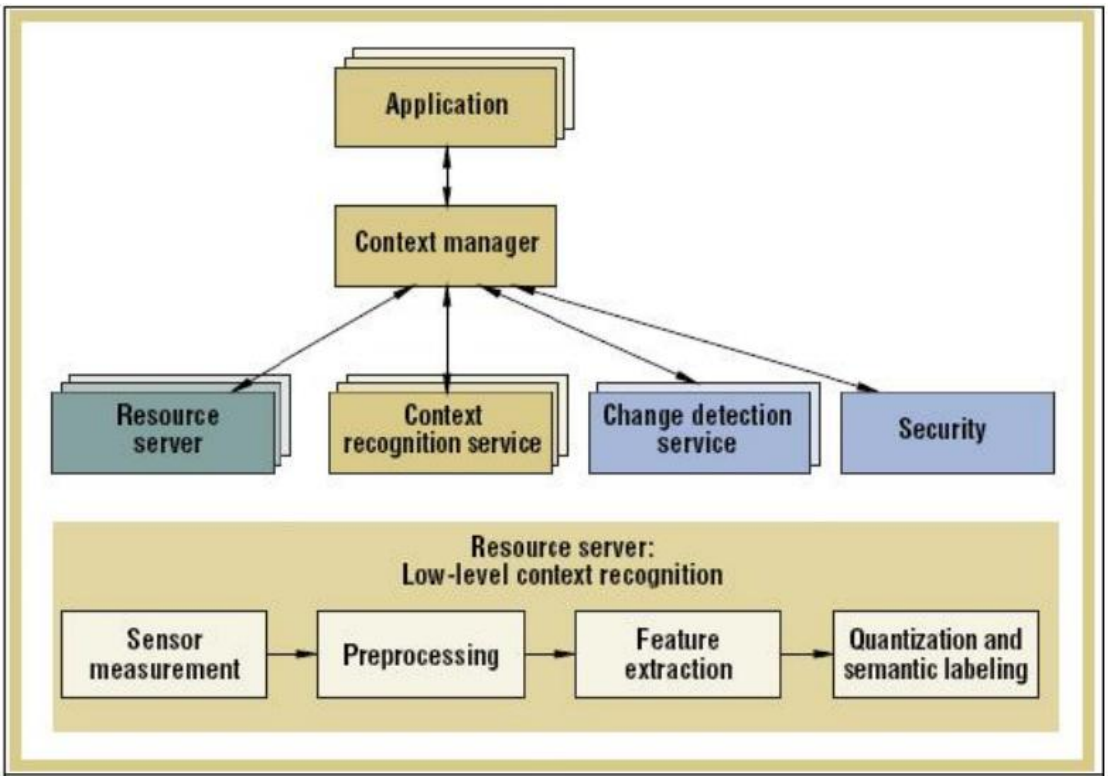

Figure 2.3 : l'architecture générale du Context Management Framework (CMF) [Korpepaa, 2004].

Lorsque des entités communiquent, le gestionnaire du contexte fonctionne comme un serveur central tandis que les autres entités agissent comme des clients et utilisent des services fournis par le serveur. « *context manager* ».

Au cœur d'un terminal mobile, le *gestionnaire du contexte* stocke les informations du contexte à partir de toute source disponible pour le terminal, et il sert à des clients de trois façons:

- Les clients peuvent interroger directement le gestionnaire (comme une base de données du contexte) pour obtenir des données du contexte.
- Les clients peuvent souscrire à divers services de notification de changement du contexte.
- Les clients peuvent utiliser le niveau supérieur (composite) du contexte d'une façon transparente, où le « *context manager* » contacts les services de reconnaissance nécessaires.

2.4. Service oriented context-aware middleware (SOCAM)

SOCAM [Gu, 2004], est une architecture qui vise à fournir une infrastructure qui supporte le développement de services sensibles au contexte dans des environnements intelligents. Cette architecture propose un middleware distribué qui convertit les différents espaces physiques dans lesquels le contexte peut être partagé et les fournit à des services sensibles au contexte. Dans la figure 2.4, SOCAM comprend les composants suivants :

- *Context Providers* : il construit une vue abstraite du contexte à partir de sources hétérogènes externes ou internes, et les transforme vers une représentation OWL pour que le contexte puisse être partagé et réutilisé par d'autres services.
- *Context interpreter* : c'est le service de raisonnement logique, pour traiter l'information du contexte.
- *Context database* : enregistre les ontologies du contexte pour un sous-domaine. Il y a une seule base de données du contexte pour chaque domaine.
- *Context-aware services* : il utilise les différents niveaux du contexte, et adapte la manière de les obtiens selon le contexte courant.
- *Service locating service* : il assure un mécanisme quand le « *context provider* » et le « *context interpreter* » peuvent annoncer leurs présences, et aussi permet aux utilisateurs ou les applications pour découvrir ces services.

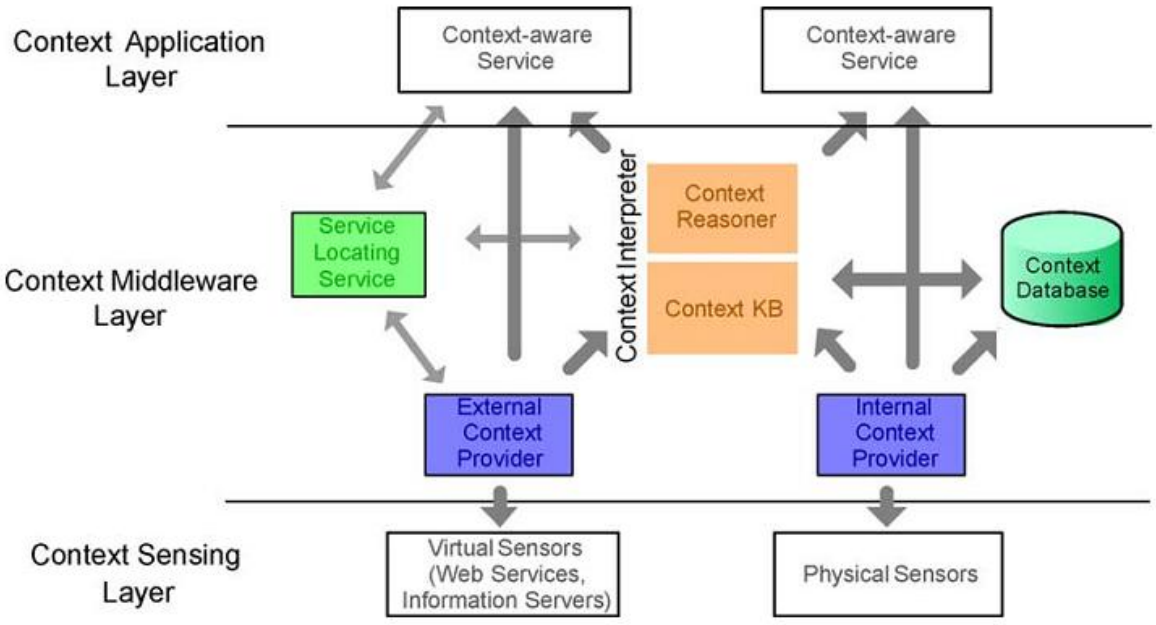

Figure 2.4 : L'architecture globale de la plateforme SOCAM[Gu, 2004].

2.5. Plateforme d'adaptation d'applications a de nouveaux contextes d'utilisation SECAS

SECAS [Chaari, 2007], est une architecture d'adaptation d'applications à de nouveaux contextes d'utilisation. La plate-forme SECAS s'appuie sur trois couches (voir Figure 2.5) :

(1) *La couche de gestion du contexte :* a comme objectif la capture, l'interprétation du contexte et sa dissémination à la couche d'*adaptation*,

(2) *La couche Adaptation :* adapte le comportement de l'application au contexte capturé. L'adaptation est assurée sur les trois composants des entités logicielles de l'application : le service, l'interface utilisateur et les données échangées avec le service,

(3) La *Couche de déploiement d'application :* permet d'intégrer de nouvelles applications à la plateforme en vue de leur adaptation

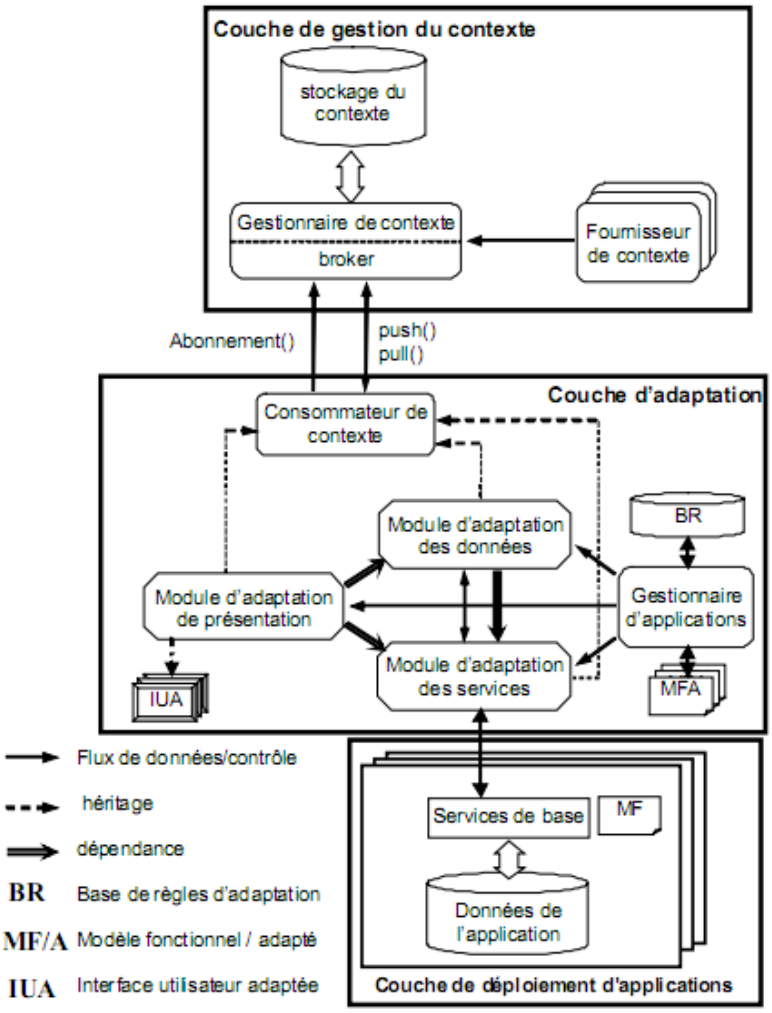

Figure 2.5 : Architecture générale de SECAS[Chaari, 2007]

Conclusion

L'objectif des travaux et les approches présentées dans ce chapitre est le développement des applications sensibles au contexte, sans prendre en compte la réutilisation des logiciels. Leur but principal est de résoudre un problème spécifique pour un domaine spécifique. Les architectures présentées pour assurer la sensibilité au contexte accordent moins d'importance à comment modifier le comportement de l'application pour qu'elle s'adapte au contexte [Chaari, 2007].

Chapitre III
Approches SPLs pour les systèmes sensibles au contexte

Sommaire

Introduction :

Une ligne de produits logiciels (**SPL)**, est un ensemble des produits crées avec l'utilisation des architectures en communs, et des artefacts logiciels réutilisables, ainsi qu'un processus de dérivation des produits [Bosch, 2000]. Elle permet de concevoir et de développer une famille de produits qui prennent en compte des facteurs de variation et permet de minimiser le coût et le temps de réalisation. Les facteurs de variation peuvent être :

(1) *techniques* (utilisation d'une variété de matériels associés aux logiciels),
(2) *commerciaux* (création de plusieurs versions allant d'une version limitée à une version complète),
(3) ou *culturels* (logiciels destinés à plusieurs pays).

Pour représenter la variabilité avec le modèle de caractéristiques (feature Model **FM**) de **SPL**, il est indispensable d'identifier les caractéristiques qui sont en commun pour tous les produits de **SPL** ou spécifiques à la majorité de ses membres [Arboleda, 2009]. Le rôle

principal de SPL, est *la dérivation des produits* qui est la phase finale de construction d'un produit [Deelstra, 2004]. La dérivation du produit peut être appliquée au moment de l'exécution pour rendre un SPL dynamique [Hallsteinsen, 2008].Un *SPL* dynamique (DSPL) permet de produire des systèmes qui peuvent être adaptés au moment de l'exécution aux différents besoins d'utilisateur ou aux changements de situations des ressources disponibles [Parra-a, 2009].

Dans ce chapitre, nous présentons les insuffisances et les limites d'une SPL traditionnelle pour l'implémentation des systèmes sensibles au contexte, et la nécessité des solutions présentées dans le cadre des DSPL pour le développement et le déploiement des systèmes sensibles au contexte.

1. Ligne de Produits Logiciels

Selon Clements et Northrop [Clements, 2002], une ligne de produits logiciels comporte un ensemble de produits logiciels qui partagent des caractéristiques (Features) communes et satisfaisantes des besoins d'un domaine particulier. L'ingénierie de ligne de produits logiciels concerne la production d'une famille de systèmes similaires au lieu de la production d'un système individuel. L'ingénierie de ligne de produits logiciels comporte trois activités principales :

(1) L'ingénierie de domaine,
(2) L'ingénierie d'application,
(3) La gestion de variabilité.

Ces trois activités sont complémentaires. L'ingénierie de domaine se préoccupe de la production des artefacts logiciels (software assets) utilisés par de différents produits de *SPL*. D'une part, l'ingénierie d'application se préoccupe de la production des systèmes spécifiques à des besoins individuels, la gestion de la variabilité concerne la sélection des ressources qui appartiennent au domaine et aux activités de l'application.

En général, une SPL implique une distinction entre deux processus séparés, qui sont le processus d'ingénierie du domaine, et le processus d'ingénierie d'application. L'ingénierie de domaine est définie comme une activité de collection, d'organisation et d'enregistrement de l'expérience de développement des systèmes ou des parties des systèmes pour un domaine particulier dans une forme d'artefacts réutilisables (Exemples de tels artefacts : architecture, modèles, code, etc.). Aussi pour donner un sens adéquat de réutilisation de ces artefacts quand un nouveau système est développé [Czarnecki-b, 2000].

L'utilisation de l'approche de développement pour la réutilisation (ou design-for-reuse, en anglais) ou l'ingénierie du domaine (ou *développement d'artefacts* [Clements, 2002]) est responsable pour déterminer la similitude et la variabilité parmi des membres d'une famille d'un produit. En général, l'ingénierie du domaine est divisée en deux analyses de domaine, la conception du domaine et l'implémentation du domaine.

L'ingénierie d'application est le processus de développement d'un système d'un domaine particulier [Czarnecki-b, 2000]. L'ingénierie d'application (*Développement de produit* [Clements, 2002]) est responsable de dériver un produit concret à partir d'une SPL selon l'approche « conception avec réutilisation ». Pour atteindre cet objectif, elle réutilise les

artefacts réutilisables développés précédemment. Ce processus est divisé aussi en deux analyses de l'application : la conception de l'application et l'implémentation de l'application.

1.1 La gestion de la variabilité des systèmes sensibles au contexte:

Avant de montrer la représentation des variabilités du contexte, nous en introduisons la notion Dey [Dey-a, 2001] qui le définit comme couvrant toute les informations qui peuvent être utilisées pour caractériser la situation d'une entité. Une entité est une personne, un lieu, ou un objet qui peut être pertinent pour l'interaction entre l'utilisateur et l'application. Une autre définition [Chaari, 2007], défini le contexte comme l'ensemble des paramètres externes à l'application pouvant influer sur son comportement en définissant des nouvelles vues sur ses données et ses fonctionnalités. Ces paramètres peuvent être dynamiques et peuvent donc changer durant l'utilisation de l'application. L'ensemble des situations possibles d'une entité observée sont définies en aval de la conception de l'application. Chaari a définit aussi dans [Chaari, 2007] une ***situation contextuelle*** comme étant une instance de paramètres d'un contexte donné qui caractérise une situation contextuelle.

Exemple d'une situation contextuelle : {utilisateur='médecin néphrologue',
terminal='PDA', localisation='Domicile du patient X'}.

Catégorisation de contexte

Catégoriser les informations du contexte, consiste en la facilité de sa manipulation dans les systèmes adaptables. Parmi les travaux qui ont proposés une catégorisation des informations du contexte, on trouve [Kirsch, 2005] qui en a classifié selon un contexte physique de l'utilisateur et organisationnel. Une autre catégorisation est donnée selon un contexte utilisateur (les préférences de l'utilisateur), un contexte physique (les caractéristiques des dispositifs physiques) et un contexte social (concerne les interactions entre les utilisateurs existants et les nouveaux arrivants) [Amara, 2006]. Dans [Chaari, 2007] l'auteur a défini formellement une situation contextuelle comme étant un espace multidimensionnel où chaque dimension représente une facette du contexte. Ensuite, il a défini le contexte d'utilisation comme un vecteur à cinq dimensions : terminal, communication, utilisateur, localisation, environnement.

1. Le **Terminal** est une facette qui décrit toutes les caractéristiques matérielles et logicielles des terminaux utilisés incluant par exemple la description de l'écran (taille, résolution,…).

2. La **Communication** comprend les caractéristiques des réseaux utilisés comme leurs bandes passantes, leurs connectivités et leurs qualités de services.
3. Les **Utilisateurs** décrivent toutes les caractéristiques de l'utilisateur final de l'application incluant ses capacités, ses préférences, son profil et ses droits d'accès.
4. La **Localisation** décrit l'emplacement logique et physique de l'utilisateur. L'emplacement logique peut être une adresse IP alors que l'emplacement physique est généralement l'adresse géographique de l'utilisateur (pièce, adresse postale).
5. L'**Environnement** est complémentaire aux autres facettes du contexte. Il sert à enrichir le modèle du contexte par d'autres paramètres utiles selon le domaine d'application. Par exemple, nous pouvons trouver dans cette facette la température ambiante, l'activité ou l'état actuel de l'utilisateur (conduire une voiture, dormir, occupé…).

1.2 Modélisation de contexte et l'adaptation dynamique avec Le modèle de caractéristiques

L'adaptation autonome et les systèmes dynamiques adaptent leurs comportements selon le contexte d'exécution. Les informations contextuelles produisent des multiples facteurs de variabilité qui provoquent autant que possible des configurations pour une application au moment de l'exécution. Les règles d'adaptations consistent à relier la variabilité dynamique du contexte avec les configurations (*combinaison de caractéristiques*) possibles de système. Le travail d'Acher et Collet [Mathieu, 2009] consiste à utiliser le modèle de caractéristiques FM pour modéliser le contexte et les variantes de système ensemble avec leurs inter relations, comme une façon pour configurer l'adaptation d'un système avec le respect d'un contexte particulier.

1.2.1 Modélisation des caractéristiques du contexte

Les caractéristiques du contexte doivent être bien identifiées, modélisées et gérées pour l'adaptation autonome, afin de bien raisonner sur le changement du contexte et d'adapter en conséquence une application selon un ensemble de stratégies ou un plan de reconfiguration. La diversité et la combinaison des caractéristiques d'un contexte sont nécessaires pour satisfaire les différents scenarios d'adaptation des systèmes sensibles au contexte avec la présentation de la variabilité dans les propriétés du contexte. La notion de variabilité du contexte devient particulièrement importante pour relier les caractéristiques d'un système

directement à l'environnement physique [Hartmann, 2008]. Comme le contexte détermine l'environnement où le système fonctionne, il faut identifier cette variabilité.

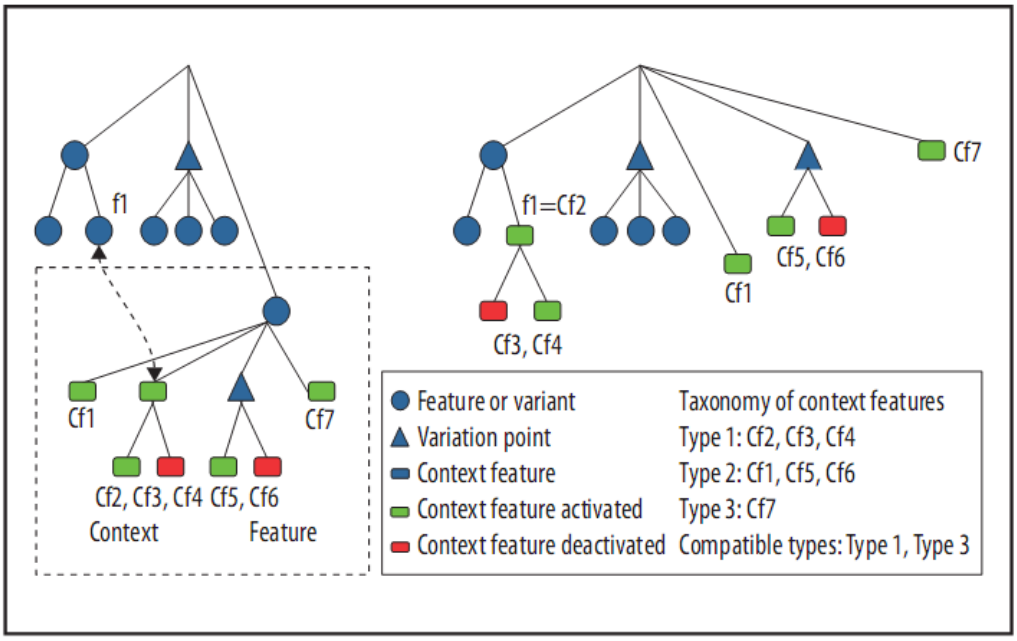

Figure 3.1 : Modèle Caractéristiques du contexte qui présente l'alternative qui utilise la taxonomie des types prédéfinis pour supporter le changement dynamique de variabilité structurel [Capilla, 2014].

Le concepteur d'une famille de systèmes sensibles au contexte doit identifier la variabilité des informations du contexte utilisées pour adapter le comportement d'un système. [Capilla, 2014] proposent deux approches pour modéliser les caractéristiques du contexte : Lier les deux modèles de caractéristiques et un seul modèle de caractéristiques.

A- Relie les deux modèles de caractéristiques

Le modèle à gauche (figure 3.1), représente un modèle de caractéristiques qui inclut deux branches où le concepteur modélise les caractéristiques du contexte séparément des autres non-contextuelles (application). Les dépendances entre celles contextuelles et non-contextuelles surchargent la relation entre elles, une telle dépendance peut être bidirectionnelle. Exemple (figure 3.1, le modèle à gauche), la caractéristique F1 (Fonction) peut avoir différentes valeurs qui dépendent d'un contexte C (la F1 est dépend des valeurs{Cf2,Cf3} du contexte C), ou une caractéristique particulière du contexte C [Capilla, 2014].

B- Un seul Modèle de caractéristiques :

Dans le model à droite (Figure 3.1), les caractéristiques contextuelles et non-contextuelles sont modélisées dans le même Modèle, pour diminuer le nombre de dépendances entre elles, comme la relation entre la caractéristique *f1* et la caractéristique contextuelle *Cf2*. Les caractéristiques contextuelles sont étiquetées dans FM et classifiés selon un ensemble de types prédéfini. Dans la Figure (Figure 3.1), ils présentent en vert et en rouge, respectivement ces caractéristiques qui sont activées et désactivées à un moment donnée.

L'approche A est plus réutilisable où plusieurs contextes sont compliqués, le concepteur modélise les caractéristiques du contexte séparément des autres non-contextuelles (application). [Capilla, 2014].

[Guo, 2011], proposent un FM étendue qui inclut les informations des ressources disponibles (exp : performance, CPU, Mémoire et le coût de développement), pour les caractéristiques dynamiques du système. Ces ressources sont représentées dans FM avec l'utilisation des *caractéristiques attributs* (*feature attributes* [Benavides, 2005]), *feature attributes* est liée aux caractéristiques qu'a un impact sur le produit final.

1.2.2 Analyse du contexte:

Selon [Capilla, 2014], l'analyse du contexte est une opération complémentaire de l'analyse du domaine d'application. Elle est utilisée pour identifier et modéliser les propriétés du système du contexte, et les transitions entre ses situations.

L'analyse du contexte doit suivre les étapes suivantes :

- Identifier les propriétés physiques du système : Le concepteur identifie les propriétés du système pertinentes aux changements de l'environnement ou avec lesquelles le système va interagir.
- Modéliser les caractéristiques du contexte : Ces propriétés physiques qui varient selon le changement dans le contexte sont modélisées dans une branche séparée.
- Identifier les caractéristiques qui modifient la variabilité structurelle dynamiquement : au plus des caractéristiques du contexte qui peuvent activer ou désactiver, le concepteur doit aussi en identifier celles qui ont un impact sur la variabilité structurelle, quand le système à besoin de modifier ses fonctionnalités.
- Définir les règles opérationnelles avec les contraintes (requiers et exclues) utilisée dans les lignes de produits traditionnels, ces opérations de dépendances contrôlent l'activation et la désactivation des caractéristiques du contexte au moment de l'exécution.

- Définir le multiple mode de liaison : le concepteur doit définir le moment de liaison pour chaque caractéristique ou un group de caractéristiques du contexte afin de spécifier les transitions entre les différents moments de liaison et les modes d'opération du système.

1.2.3 Modélisation de l'adaptation

La logique de l'adaptation consiste en la définition du comportement d'un système selon les informations du contexte. Cela implique que L'SPL doit changer sa configuration selon les informations contextuelles disponibles [Mathieu, 2009]. Les règles d'adaptations sont définies avec le langage de contraintes de FM. La syntaxe abstraite des règles est constituée de deux parties: une partie gauche '*Left hand side (LHS)'* et une partie droite '*Right hand side (RHS)*' en plus des types de connexions entre les caractéristiques (*and, or* et *not*). Avec ce mécanisme, une règle *Event-Condition-Action* (ECA) peut être exprimée, LHS représente la partie condition qui est une expression basée sur les informations du contexte, l'action est le changement dans la configuration des variantes (RHS de la règle).

1.3 Modélisation des caractéristiques dynamiques d'une application

Les caractéristiques dynamiques dans les systèmes sensibles au contexte peuvent être activées ou désactivées au moment de l'exécution selon un contexte détecté. [Bencomo, 2008] introduit les caractéristiques dynamiques comme étant des variantes qui présentent des solutions alternatives aux situations détectées au moment de l'exécution, mais il n'a pas différencié entre les caractéristiques dynamiques et statiques dans le MF. [Dinkelaker, 2010,Czarnecki-a, 2007], Identifient les caractéristiques dynamiques comme étant une extension des caractéristiques statiques, avec une notation spéciale pour que la ligne du produit dynamique soit spécifique. Ils introduisent les contraintes dynamiques pour :

i) l'activation d'une caractéristique qui dépend de la sélection d'une autre.
ii) deux caractéristiques qu'il ne faut pas activer simultanément (*alternative*).
iii) les contraintes de précédence qui déclarent qu'un ensemble de caractéristiques peuvent être activées en même temps, mais avec un ordre d'exécution pour accomplir un objectif (précèdes).

La figure 3.2, Montre une représentation visuelle d'une caractéristique dynamique et les différentes contraintes.

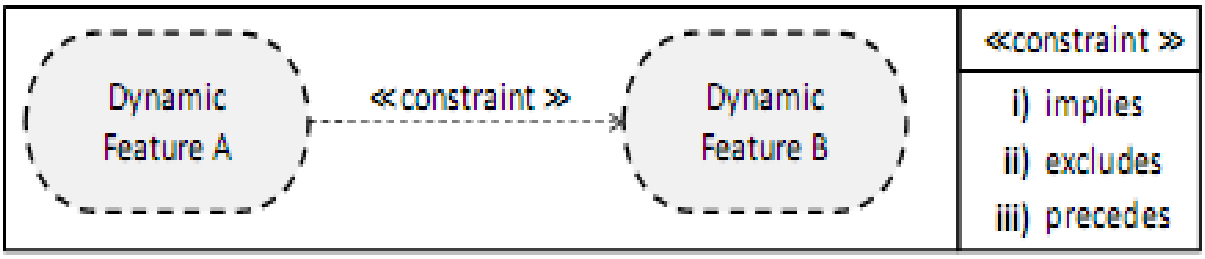

Figure 3.2: Notation de caractéristiques dynamique [Dinkelaker, 2010].

2. SPL Statique pour la dérivation des systèmes sensibles au contexte

Un SPL statique assure l'adaptation des systèmes à un contexte particulier défini au moment de conception, soit au niveau modèle ou code source, dans les deux cas l'adaptation est une partie de processus de dérivation d'un produit à partir d'une configuration déterminée par l'utilisateur.

Une ligne de produit traditionnel pour les systèmes sensible au contexte a comme objectif la dérivation d'un produit conforme aux exigences de l'utilisateur et à la situation de l'environnement. Cependant, ces exigences et le contexte de l'environnement doivent être définis et identifiés au moment de la conception pour pouvoir choisir les caractéristiques conformes à ces exigences (la plateforme d'exécution, l'environnement physique, la langue de l'utilisateur, etc.). Le résultat de cette ligne de produits est un produit conforme aux exigences définies au moment de la conception. L'inconvénient de cette approche et qu'une fois le produit est dérivé il est impossible de modifier son comportement ou sa configuration au moment d'exécution.

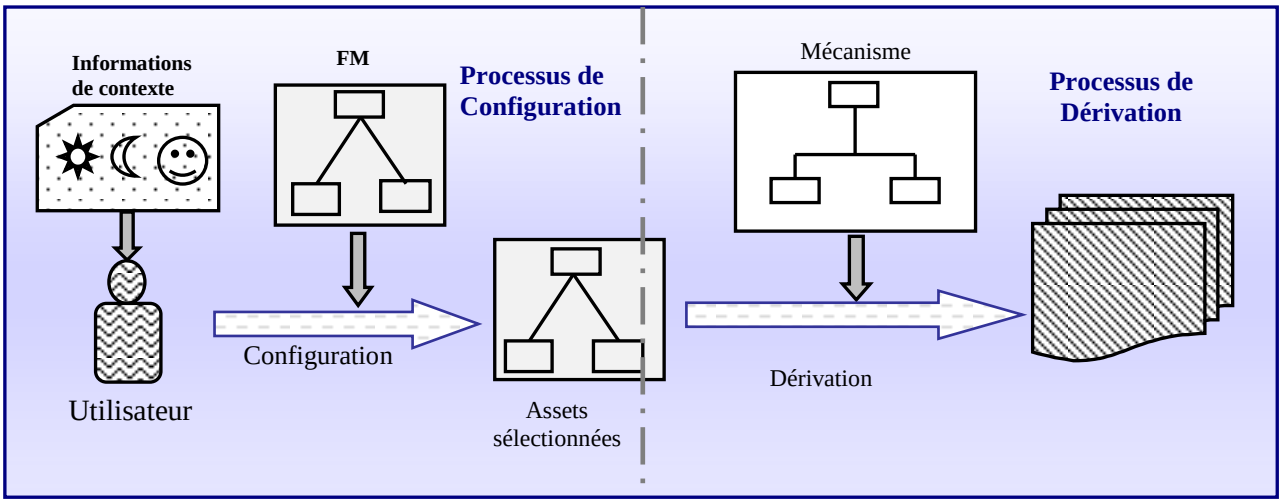

Figure 3.3: Le processus de dérivation d'un produit conforme à l'état du contexte.

Pour dériver un produit, l'utilisateur doit sélectionner une bonne combinaison de caractéristiques à partir de FM qui satisfait ses besoins, ainsi que les états de l'utilisateur et son environnement de travail, doivent être identifiés et représentés au moment de processus de dérivation, afin de bien choisir ses propres caractéristiques conformes à sa situation (voir Figure 3.3, Processus de Configuration). Dans cette étape, les caractéristiques sélectionnées sont conformes au contexte de l'utilisateur.

Après la sélection d'un sous-ensemble de caractéristiques, elle vient l'étape de composition des assets correspondantes à chaque caractéristique sélectionnée afin de dériver un produit conforme à l'exigence de l'utilisateur. Pour bien adapter le code à générer à partir des caractéristiques sélectionnées, des mécanismes et des techniques sont utilisés pour composer et définir les liens (binding) entre les composants présentés dans le FM.

Les différentes approches [Arboleda, 2009, Perrouin, 2008] qui utilisent les SPLs traditionnelles se concentrent sur l'adaptation du code généré à partir des besoins identifiés (Modèle). C'est-à-dire, le SPL est sensible au contexte dans le sens de la dérivation du produit au moment de la conception (Ingénierie d'application). Il est alors conscient au changement dans la phase de configuration du produit (sélection des caractéristiques). [Arboleda, 2009] avait l'objectif de produire un générateur de code conscient au contexte, *il utilise MDE (model driven engineering*) pour définir un modèle de variabilités qui sera transformé en un modèle intermédiaire (Aspect Oriented Modeling **AOM**) puis la génération de code.

Limites des SPLs statiques et le besoin de SPLs dynamiques

Une SPL traditionnelle compose et définit des liens (*bind*) entre la plupart des caractéristiques statiques avant la livraison d'un produit [Hallsteinsen, 2008]. Les Produits composés avec des caractéristiques statiques fonctionnent bien dans des contextes où les changements des besoins (l'environnement du produit et d'autres aspects connexes) sont rares. Cependant, ils souffrent dans des environnements plus dynamiques où de tels changements se produisent fréquemment. Les exemples incluent les environnements d'exécution des systèmes pervasifs et ubiquitaires, les systèmes ultras larges, etc. Ces systèmes nécessitent un degré plus élevé d'adaptabilité leur permettant de s'adapter dynamiquement à l'évolution du contexte et des objectifs.

Une SPL avec des caractéristiques qui s'adaptent dynamiquement à l'évolution du contexte et des objectifs est connue comme ligne de produits logiciels dynamique (DSPLs :dynamiques Software Product Lines [Bencomo, 2008].. Une DSPL prend en charge différents temps de

liaison (*binding*). Quelques points de variation liés à des propriétés statiques de l'environnement sont liés (*binding*) avant l'exécution, tandis que d'autres points de variation, plus dépendants de l'environnement et des objectifs dynamiques, sont liés (*rebound*) au moment de l'exécution.

Une DSPL doit prendre en charge plusieurs mécanismes de variabilité. Ses produits nécessitent un support spécialisé pour la liaison dynamique [Cetina-a, 2008, Dinkelaker, 2010, Gomaa, 2006, Trinidad, 2007] pour les cas où des variations peuvent être liées avec des variantes différentes lors de l'exécution.

3. SPL Dynamique pour la dérivation des systèmes sensibles au contexte

La nécessité d'avoir des applications qui peuvent s'adapter dynamiquement au changement du contexte a conduit les chercheurs d'utiliser des mécanismes d'adaptations par la réutilisation des applications indépendamment des *intergiciels* [Blair, 2004, Garlan, 2002]. L'intérêt des *intergiciels* est de s'occuper de la reconfiguration des applications à l'exécution, pour séparer la spécification des fonctionnalités de l'application de la préoccupation d'adaptation afin de remédier à la complexité.

Les SPL statiques utilisent les variabilités identifiées comme des connaissances pour configurer une application au moment de la conception. Lorsque la situation à laquelle le produit est configuré change au moment d'exécution, on doit reprendre le processus de configuration (nouvelle version pour chaque situation identifiée). Une SPL Dynamique a pour objectif de réexécuter les processus de configuration et de dérivation au moment de l'exécution [Bencomo, 2008].

Les systèmes sensibles au contexte sont fortement liés au changement de l'état de l'environnement. Ces systèmes doivent évoluer ou s'adapter aux différentes informations du contexte détectées. L'évolution d'un système est une modification de son comportement d'une façon continue [Oreizy, 1999, Oreizy-b, 2008]. L'adaptation d'une application a besoin d'un processus qui comprend :

- détecter l'information liée à un changement dans l'environnement,
- générer une reconfiguration,
- appliquer cette configuration sur l'application.

La référence le bien connu pour ce modèle est présenté par IBM [IBM, 2006] connue comme MAPE (voir chapitre I – Section 6.2.2). Une DSPL prend en charge différents temps

de liaison (*binding*). Quelques points de variation liés à des propriétés statiques de l'environnement sont liés avant l'exécution, tandis que d'autres points de variation, plus dépendants de l'environnement et des objectifs dynamiques, sont liés (*rebound*) au moment de l'exécution [Abbas, 2011].

Approches SPLs Dynamiques

Toutes les approches DSPL ont comme objectif la dérivation de produits qui s'adaptent pendant leurs exécutions aux différentes situations du contexte. Elles définissent des règles d'adaptation et des technologies qui permettent la modification de l'architecture des applications durant l'exécution selon les différentes situations du contexte. Dans cette section, nous nous intéressons aux travaux liés à notre contribution.

[Bencomo, 2008] propose une approche DSPL pour les systèmes conscients du contexte. Cette approche supporte la conception et l'opération de l'adaptation dynamique des systèmes. Ils utilisent le concept de modèle de variabilités de lignes de produits logiciels pour définir comment les systèmes s'adaptent au moment de l'exécution aux changements de leurs environnements. Grâce à la machine à états, chaque état représente une situation particulière et qui exécute une configuration adéquate à cette situation, les transitions entre les états définissent le changement dans le contexte.

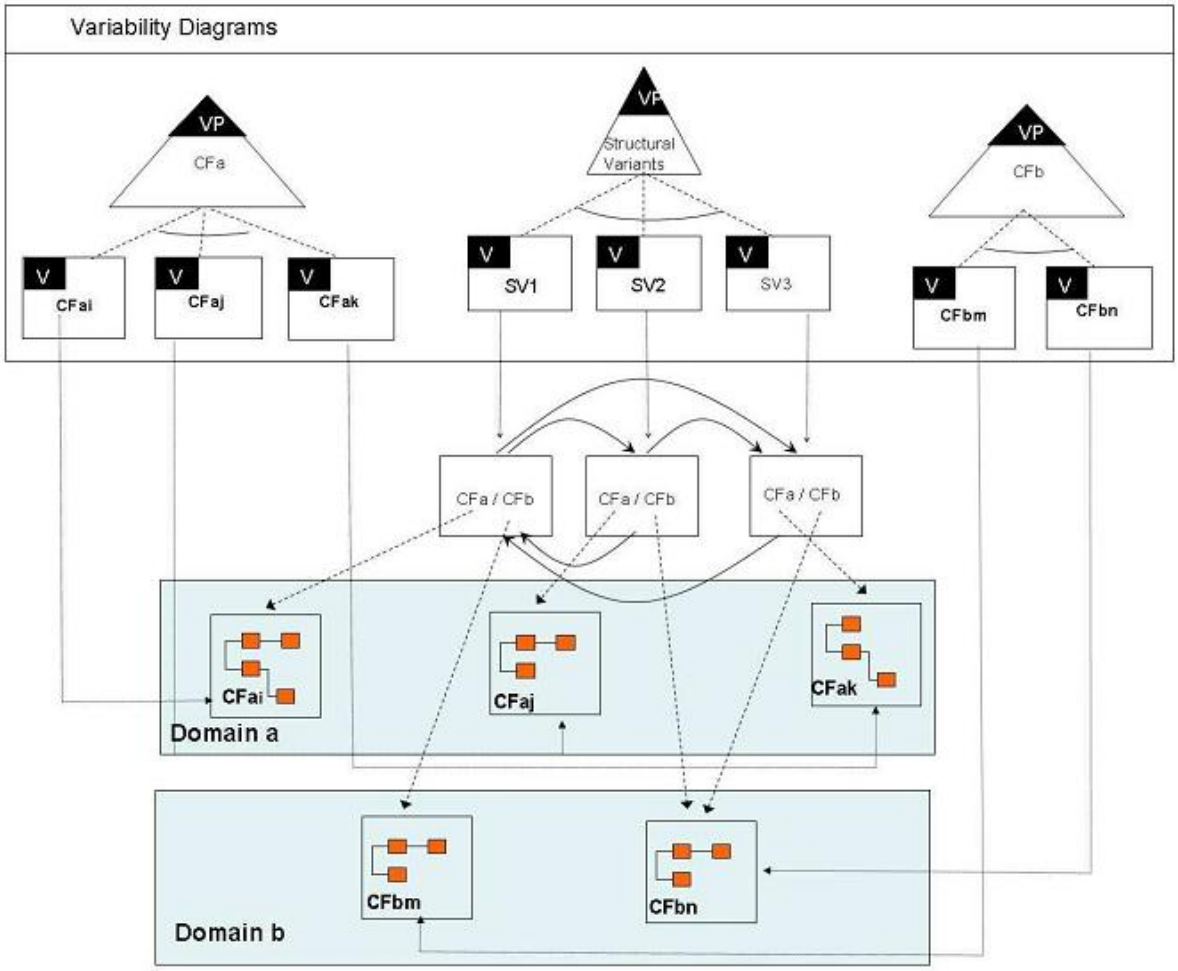

Figure 3.4: Variantes structurels pour les deux domaines de comportement de l'intergiciel. [Bencomo, 2008].

L'approche modélise deux dimensions de variabilités dynamiques : la variabilité de l'environnement qui définit les conditions auxquelles un système doit s'adapter, et la variabilité structurelle qui définit les configurations possibles de l'architecture. Les dimensions des variabilités identifiées sont modélisées par un langage spécifique au domaine ''domain-specific modelling languages (DSMLs)'' lié à un *intergiciel* d'adaptation qui fournit un support pour la variabilité, selon les conditions de reconfiguration, durant l'exécution (Figure : 3.4).

[Trinidad, 2007] propose une transformation de modèle à partir d'une '*Feature modèle*' (voir Figure 3.5) il génère un modèle de composants (voir Figure 3.6). Pour chaque caractéristique identifiée dans FM a un composant qui l'implémente. Un autre nouveau composant est défini, nommé *configurateur*, qui a comme objectif de créer des liens (définissant les interfaces) entre les composants au moment de l'exécution pour représenter

une architecture qui repend la configuration choisie par l'utilisateur, le *configurateur* applique la configuration au moment d'exécution.

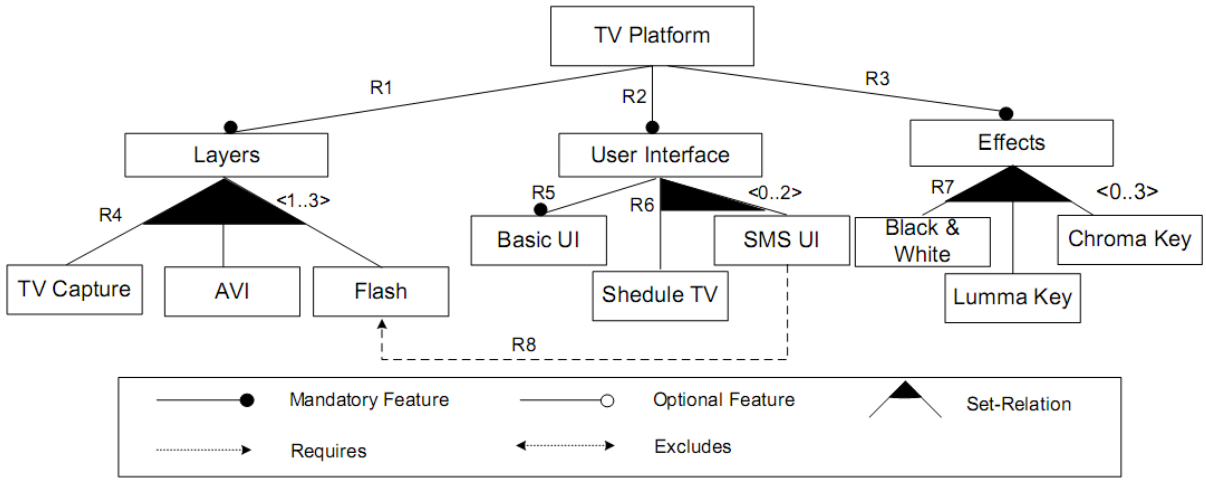

Figure 3.5: Exemple de Features Model. [Trinidad, 2007].

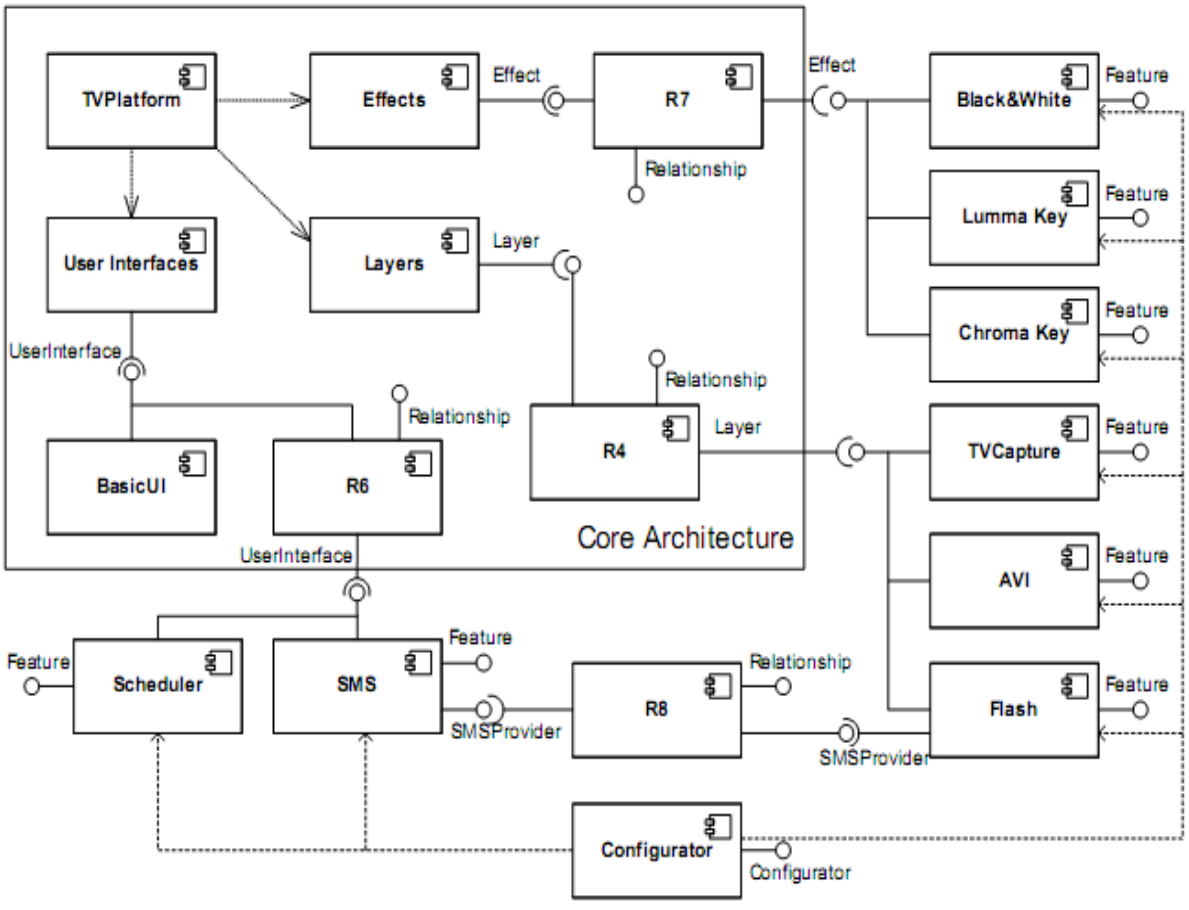

Figure 3.6: Modèle de composants. [Trinidad, 2007].

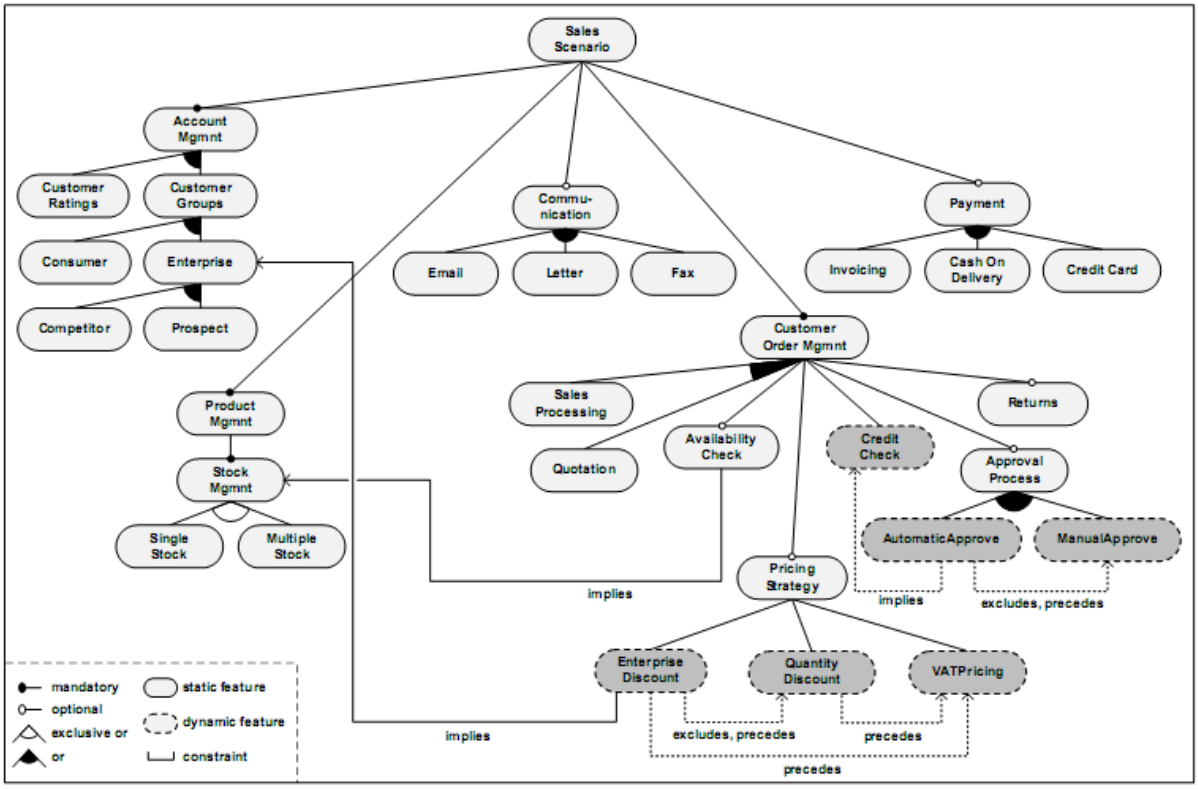

Figure 3.7 : Modèle de caractéristiques dynamique. [Dinkelaker, 2010].

[Dinkelaker, 2010] utilisent un modèle d'aspects au moment de l'exécution pour implémenter une ligne de produits dynamique. Cette approche utilise les concepts SPL (features et contrainte) pour implémenter un modèle d'aspects. Pour dériver un produit, AOP concerne les variabilités dynamiques (dynamic feature Figure 3.7) qui sont les variantes qui modifient le comportement du système.

[Parra-b, 2011] a proposé un DSPL pour produire des logiciels sensibles au contexte, qui doivent surveiller les événements provenant de leur environnement et réagir en conséquence. Il introduit un modèle de variabilité (voir Figure 3.8) et un modèle de composition pour modéliser les produits sous forme de modèles d'aspects (Figure 3.9). Chaque modèle d'aspects a trois parties : l'architecture, les modifications, et le point de coupe. Il utilise les modèles pour représenter les assets de chaque caractéristique. Il a défini la configuration basée contexte au moment de l'exécution pour surveiller le changement de l'environnement pour activer ou désactiver des composants. Il utilise les modèles d'aspects pour représenter les caractéristiques et leurs dépendances avec le contexte.

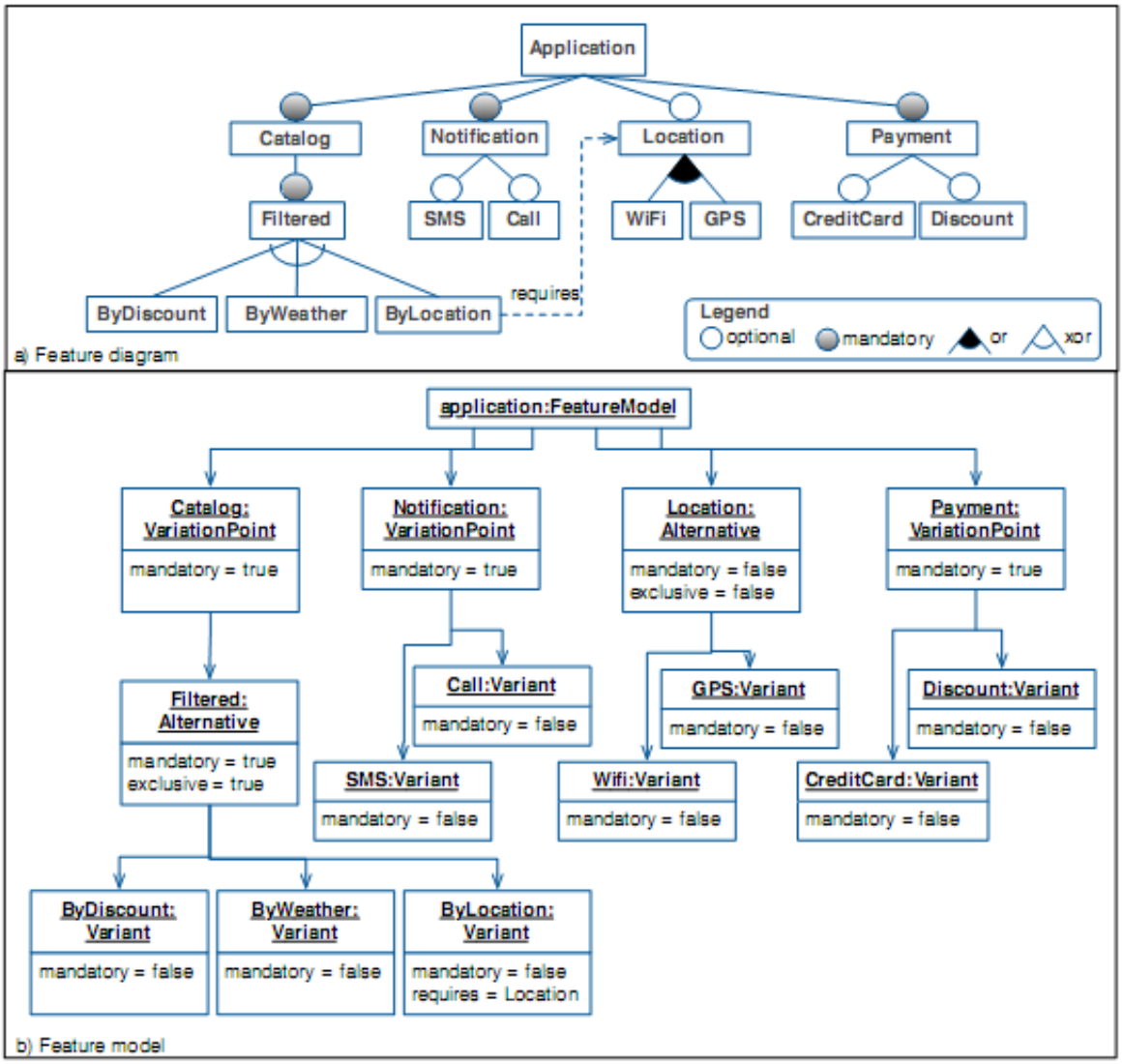

Figure 3.8 : Modèle de caractéristiques CAPucine. [Parra-b, 2011]

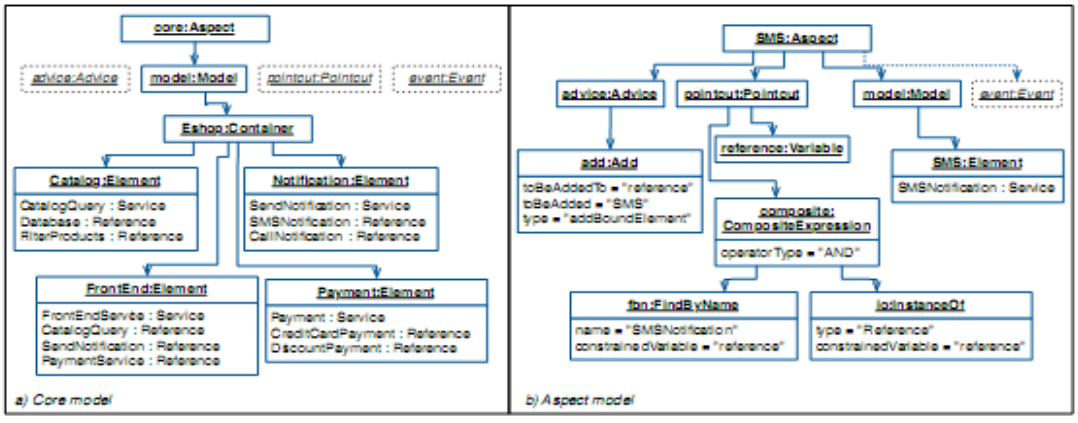

Figure 3.9 : Exemple de modèles d'architecture et d'aspect. [Parra-b, 2011]

[Guo, 2011] se préoccupe du problème de configuration. Les développeurs essayent de dériver une sélection de caractéristiques optimisées, et qui répondent à un ensemble de besoins. Il a identifié, pour chaque FM, un nombre exponentiel de configurations de produits. Il a aussi identifié le problème de sélection qui doit être vérifié, pour respecter les contraintes définies dans le FM. Il a aussi proposé un outil GAFES pour automatiser le processus de configuration. GAFES est basé sur une approche d'optimisation de la sélection des caractéristiques définies avec SPL. GAFES utilise un algorithme génétique pour la sélection des caractéristiques. L'approche est exécutée au moment de la conception pour générer une configuration conforme aux contraintes définies dans FM et les conditions de disponibilité des ressources.

[García, 2013] a utilisé le même outil *GAFES* défini par [Guo, 2011], pour proposer une approche selon le principe de l'informatique autonome, et qui s'exécute au moment d'exécution. Il propose un algorithme de transformation de modèle, qui génère à partir de modèle de l'architecture du système un modèle de variabilité de l'architecture (AFM).

4. Ligne de Produits Logiciels autonome ASPL

Une ligne de produits logiciels autonome (ASPL) [Andersson, 2000] est une DSPL qui prend en charge l'adaptation des produits et l'évolution autonome. Les produits d'une ASPL sont équipés d'une boucle de contrôle explicite qui offre aux caractéristiques un support pour l'auto-adaptation (*self-adaptation*), l'auto-configuration (*self-configuration*) et l'auto-optimisation (*self-optimization*) [Cheng, 2009, Oreizy, 1999, Salehie, 2009]. Les produits d'une ASPL sont munis d'un mécanisme de variation dynamique mis en œuvre à travers trois processus clés qui sont :

(1) surveiller le produit et son contexte,
(2) fournir des compositions conscientes au contexte pour réaliser les objectives d'adaptation,
(3) effectuer l'apprentissage en ligne des décisions sur la composition optimale.

L'objectif principal de la communauté a été orienté vers les aspects techniques de l'ingénierie des lignes de produits logiciels (SPLE) pour réaliser une stratégie de réutilisation (comme l'analyse de la ligne de produits, la modélisation et le développement). Tandis que l'aspect « connaissance » a reçu moins d'attention [Hamza-a, 2010]. Cependant, les lignes de

produits maintiennent les connaissances qui constituent l'élément principal de la conception des décisions, telles que« quand » et « comment » lier (*bind*) les variantes des points de variation [Abbas, 2011].

Les Connaissances dans une gamme de produits proviennent de différentes activités et des sources d'objets. Une source importante est la rétroaction ''*feedback*'' (des demandes de changement, les rapports de défaillance, etc.) et des intervenants concernés par la production d'une gamme de produits. D'autres sources importantes des connaissances sont les activités et les artefacts liés à l'évolution des *assets* d'une ligne produits, telle que la disponibilité des composants mis à jour, des algorithmes, des frameworks / API, des structures de données, etc.

La gestion des connaissances joue un rôle important dans le succès des différentes activités d'ingénierie de ligne de produits. Dans les SPLs traditionnelles, les connaissances changent uniquement lorsque le SPL évolue pendant la maintenance ou dans les cycles d'évolution [Svahnberg, 1999]. Cependant, aujourd'hui l'informatique est plus dynamique avec de fréquents changements d'objectifs, de l'environnement, et de la technologie de mise en œuvre. Par conséquent, ce que nous appelons connaissances, à un moment donné dans le temps, peut rapidement devenir obsolète [Abbas, 2011].

Approches ASPLs

Hallsteinsen et al, dans [Hallsteinsen, 2008], identifient la variabilité dynamique, la (re-) configuration, l'auto-gestion et l'adaptation comme des propriétés requises par DSPLs. En outre, ils spécifient la sensibilité au contexte, autonome ou auto-adaptation, et la décision automatique comme des propriétés optionnelles des DSPLs. Une ASPLs soutient toutes ces propriétés en utilisant les principes de réflexion et de boucles de contrôle autonome.

Plusieurs approches à base de caractéristiques existent pour le développement des produits reconfigurables dynamiquement [Fernandes, 2008, Lee, 2006, Trinidad, 2007]. Lee Kang et al, dans [Lee, 2006], suggèrent qu'une approche orientée caractéristique (une ligne de produits) est analysée en fonction de ses caractéristiques et de leur temps de liaison.

Cetina et al, dans [Cetina-b, 2008], ont discuté et analysé des architectures de DSPL et proposent une DSPL mixte. Une DSPL mixte est une alternative intermédiaire entre DSPL

connecté et déconnecté. Une DSPL connectée est celle qui est en charge de l'adaptation des produits, tandis qu'une DSPL déconnectée est un produit qui contrôle lui-même son adaptation. Les DSPLs mixtes produisent des scénarios-conscients des produits configurables qui dépendent selon le scénario pour passer d'une DSPL Connecté à une DSPL Déconnecté et vice versa.

Hamza et al, dans [Hamza-a, 2010], ont proposé une ligne de produits orientée-connaissances, appelée KOPLE pour *Knowledge-Oriented Product Line Engineering*. Leur approche vise à tirer les bénéfices de la recherche dans le domaine de l'ingénierie des connaissances. Les auteurs expliquent que plusieurs défis et problèmes pratiques dans SPLE peuvent être efficacement résolus s'ils ont été analysés selon le contexte des connaissances.

Lutz et al. [Lutz, 2009] proposent d'utiliser les rapports de défauts pour construire les exigences des connaissances dans les lignes de produits. Les auteurs affirment que l'utilisation des rapports de défauts de membres de lignes de produits précédents peut améliorer la qualité des futurs membres, en réduisant leurs besoins liés aux défauts. Les auteurs utilisent le modèle de caractéristiques étendu pour capturer les exigences des connaissances pertinentes à la ligne de produits.

[Mathieu, 2009] L'approche a comme objectif de spécifier des règles d'adaptation pour relier les variabilités dynamiques du contexte avec les variant possibles du système. Ils ont étudié l'utilisation systématique du modèle de caractéristiques, pour modéliser le contexte et les variant du système en ensemble avec leurs relation comme le montre la figure (figure 3.10), comme une façon pour configurer les systèmes adaptatifs avec la considération d'un contexte particulier.

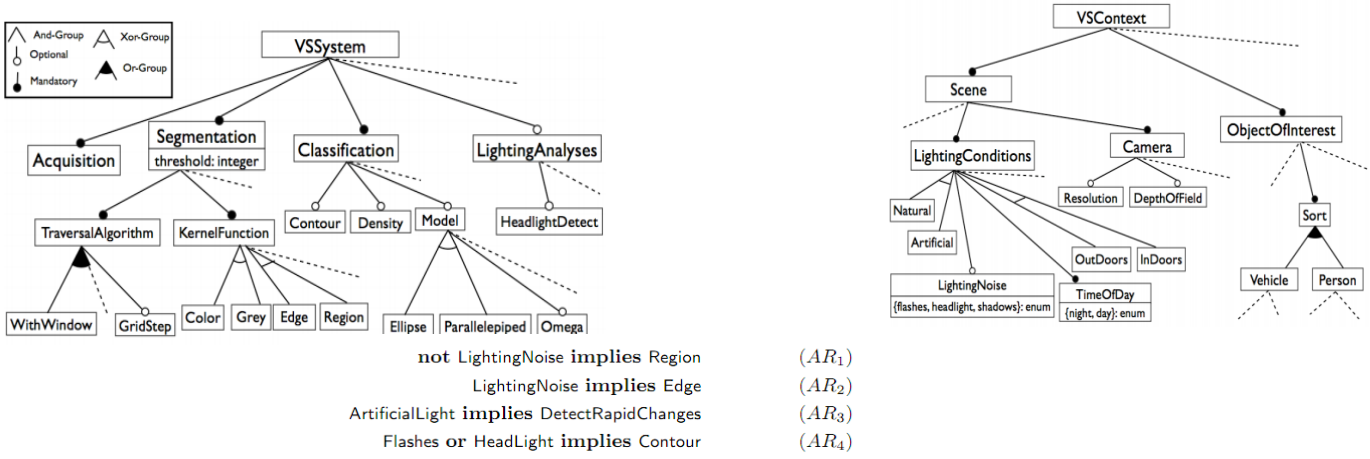

Figure 3.10 : la gestion de variabilité des systèmes sensibles au contexte. [Mathieu, 2009]

[Guo, 2011] L'approche consiste à résolue, le problème de sélection qui doit être vérifié, pour respecter les contraintes définies dans le FM et de gérer un nombre exponentiel de configurations de produits. Il a aussi proposé un outil GAFES pour automatiser le processus de configuration. GAFES est basé sur une approche d'optimisation de la sélection des caractéristiques définies avec SPL. GAFES utilise un algorithme génétique pour la sélection des caractéristiques.

[García, 2013] a utilisé le même outil GAFES défini par [Guo, 2011], pour proposer une approche selon le principe de l'informatique autonome pour la reconfiguration dynamique des applications sensibles au contexte, et qui s'exécute au moment d'exécution (figure, 3.11).

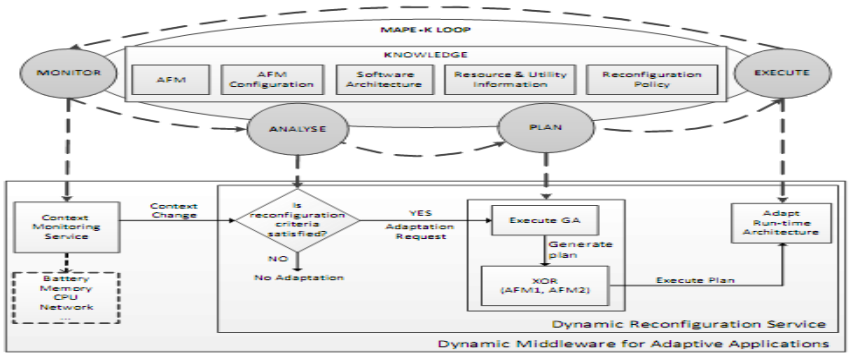

Figure 3.11 : l'adaptation au moment de l'exécution des applications mobiles. [García, 2013]

Conclusion :

Dans ce chapitre, nous avons présenté les différentes approches SPL qui traitent les systèmes sensibles au contexte. Nous avons présenté les travaux SPLs qui utilisent de différentes technologies afin de développer des logiciels sensibles au contexte.

Nous avons aussi présenté les insuffisances et les limites des SPL statiques pour le développement des produits dans des environnements dynamiques, et l'importance de DSPL qui utilisent des mécanismes d'adaptations. Celles-ci définissent les différents temps de liaison (*binding*) afin de permettre à ses caractéristiques de s'adapter dynamiquement à l'évolution du contexte et des objectifs.Nous avons, enfin, présenté les ASPLs qui sont des DSPLs qui prennent en charge l'adaptation des produits et l'évolution autonome, pour développer des produits équipés d'une boucle de contrôle explicite qui offre aux caractéristiques un support d'auto-adaptation (*self-adaptation*).

Chapitre IV
CA-SPL : Ligne de Produits Logiciels Pour Systèmes Sensibles au Contexte

Sommaire

Introduction

Avoir un service de reconfiguration dynamique pour l'adaptation des applications sensibles au contexte durant l'exécution, est assuré par des plateformes. Mais, le principe des approches existantes, consiste à choisir un plan de reconfiguration parmi une liste de configurations disponibles. C'est-à-dire, une personnalisation d'une liste de caractéristiques conforme au contexte de l'environnement de l'exécution. Ce plan de configuration est exécuté au moment de la conception selon le principe de ligne de produit statique.

Un SPLs est modélisé en habitude avec le modèle de caractéristique (MF). Ce dernier fournit un moyen intuitif pour les utilisateurs afin d'exprimer les points de variations, les alternatives et les contraintes entre ces alternatives. Comme un SPL, une caractéristique importante des systèmes sensibles au contexte est la variabilité [Mathieu, 2009]. Les techniques SPL peuvent être appliquées pour modéliser la variabilité définie dans les systèmes sensibles au contexte, mais elles n'offrent aucune solution pour modéliser quand et comment une configuration appropriée doit être choisie selon le contexte de l'environnement.

S'occuper de ce problème au niveau de la programmation prouve qui il est un défi dû au nombre important des contextes et des configurations d'un système qui il faut prendre en considération.

Afin de confronter ce problème, une approche émergée est d'utiliser un SPL dynamique [Pohl, 2005] qui essaie d'accomplir d'une façon autonome la sélection d'une configuration approprié au contexte de l'environnement. L'utilisation et l'exploitation des modèles au moment de l'exécution fournissent une solution pour s'occuper de la complexité et de la nature dynamique des systèmes sensibles au contexte [Morin, 2008].

Dans notre approche, nous présentons un processus qui permet :

1- La génération de façon autonome d'une liste de configurations conformément aux différentes situations du contexte au moment de la conception.
2- La sélection autonome d'une configuration adéquate à un contexte détecté, et l'exécution d'un plan de reconfiguration au moment de l'exécution.

Les configurations générées au moment de la conception sont cohérentes parce qu'elles respectent les contraintes définies dans le modèle de caractéristiques. Pour générer des configurations optimales pour chaque situation du contexte, il faut :

1- Avoir un ensemble de situations possibles (contextes) dont l'application peut se trouver au moment de l'exécution en utilisant le modèle de caractéristiques,
2- Utiliser un algorithme d'optimisation (nous avons choisi d'utiliser un algorithme génétique) pour générer une configuration optimale appropriée à chaque nouveau contexte identifié.
3- L'exécution d'un algorithme d'apprentissage de réseau de neurones afin d'adapter ses poids à un ensemble d'exemples (entrée : contexte, sortie : configuration) pour obtenir la classification d'une configuration selon le contexte.
4- L'exécution du réseau de neurones avec les poids déjà calculés, qui identifiera la configuration adéquate au contexte détecté.

Pour rendre le processus de configuration autonome et d'une qualité améliorée (configuration cohérente et dans temps plus vaste) au moment de l'exécution par rapport à la ligne de produit traditionnel, notre approche consiste alors en deux étapes :

Etape 1 :L'apprentissage du réseau de neurones au moment de la conception, consiste d'en faire un apprenti, le futur système aux différentes situations du contexte et la configuration nécessaire pour chaque situation.

Etape 2 : L'exécution du réseau de neurones avec les poids calculés pour générer une configuration conforme au contexte capturé. Le système bénéfice de l'expérience acquise au moment de la conception pour produire une configuration de qualité dans un temps relativement court.

1. CA-SPL: Une vue générale :

Selon le principe du paradigme de l'informatique autonome (Autonomic Computing AC [IBM, 2006]), la reconfiguration (ou l'adaptation) des applications sensibles au contexte nécessite :

- ✓ La capture des informations du contexte.
- ✓ L'analyse du contexte capturé pour détecter un changement possible.
- ✓ La génération d'un plan de reconfiguration.
- ✓ L'exécution de la reconfiguration.

Dans ce mémoire, nous proposons une DSPL autonome pour produire des systèmes sensibles au contexte qui couvre toutes les étapes de la boucle MAPE (MAPE-K loop) pour produire un système autonome, figure 4.1. L'objectif de notre approche est la même de Pascual et al [García, 2013], qui consiste en l'adaptation de l'architecture de l'application au contexte détecté, selon le principe de l'informatique autonome AC. C'est-à-dire, l'adaptation sans intervention humaine.

Notre approche consiste à utiliser un réseau de neurones au lieu d'un algorithme généthique AG, avec des *poids* adaptés, pour générer une configuration valide adéquate aux entrées (contexte) au moment de l'exécution.

Pour implémenter ce processus, on a suivi la même approche de Pascual et al, Décrite dans [García, 2013]. Cette dernière a comme objectif l'adaptation dynamique avec l'utilisation d'un algorithme génétique selon le principe de l'informatique autonome. Ce travail, consiste en une implémentation des différentes fonctions de la boucle MAPE-K afin de reconfigurer une application mobile au contexte détecté au moment de l'exécution. Ils ont identifiés le *Context Monitoring Service* (CMS), qui a comme rôle d'observer (*Monitoring*) le changement dans l'environnement et fournir ces information au *Dynamique Reconfiguration Service* (DRS). Ce dernier couvre *l'analyse* des informations observées, et la génération et l'exécution d'un *Plan* de reconfiguration. Les deux services sont intégrés dans un *intergiciel* pour le développement des applications adaptatives.

Afin de réutiliser cette approche avec notre objectif, on a modifié le CMS pour la génération d'un vecteur qui contient une situation contextuelle de l'environnement selon un codage binaire. Après avoir analysé le contexte capturé, le DRS décide si une reconfiguration

est nécessaire ou non. Si oui, il génère alors un plan à l'aide d'un réseau de neurones avec l'utilisation des poids adaptés (durant la phase de conception), afin d'adapter l'architecture d'application avec l'utilisation de toutes les étapes de MAPE-K loop [IBM, 2006], comme suit :

Knowledge : les connaissances nécessaires pour l'exécution de la boucle MAPE sont : les poids adaptés durant la conception, l'architecture de l'application, les informations du contexte détectées et leurs configurations adéquates.

Monitor : le système de gestion du contexte (*Context Monitoring Service **CMS***) capture les informations liées aux situations des entités observées et les envois au *service de reconfiguration dynamique*, sous forme d'un vecteur (tableau 4.1) qui contient les états de chaque situation identifiée dans la variabilité du contexte, quand un changement est détecté pour le notifier.

Analyse : après avoir été notifié par un évènement de changement du contexte, le *DRS* vérifie si ce changement est significatif pour lancer le processus d'adaptation, pour cela, il exécute l'opération ***xor (opération de comparaison)*** entre le dernier contexte traité et le nouveau contexte. S'il ya un changement, une adaptation de l'architecture d'application sera alors nécessaire.

Plan : dans le cas où l'analyseur décide qu'une adaptation est nécessaire, il exécute le réseau de neurones(RN) avec les poids adaptés, avec comme données d'entrés le vecteur du contexte pour générer une configuration optimale.

Une opération de comparaison est nécessaire entre les deux configurations, l'actuelle et celle générée pour qu'il se confirme qui il s'agit d'une nouvelle reconfiguration. On applique un opérateur XOR, si un changement a lieu entre les deux configurations, alors le plan d'adaptation de l'architecture de l'application, est exécuté.

Exécution : Finalement le Plan est exécuté afin d'adapter l'application à la nouvelle configuration construite (plan).

Mais avant d'exécuter le RN au moment de l'exécution, il est nécessaire d'adapter ses poids aux différents scénarios contextuels et leurs configurations adéquats au moment de la conception.

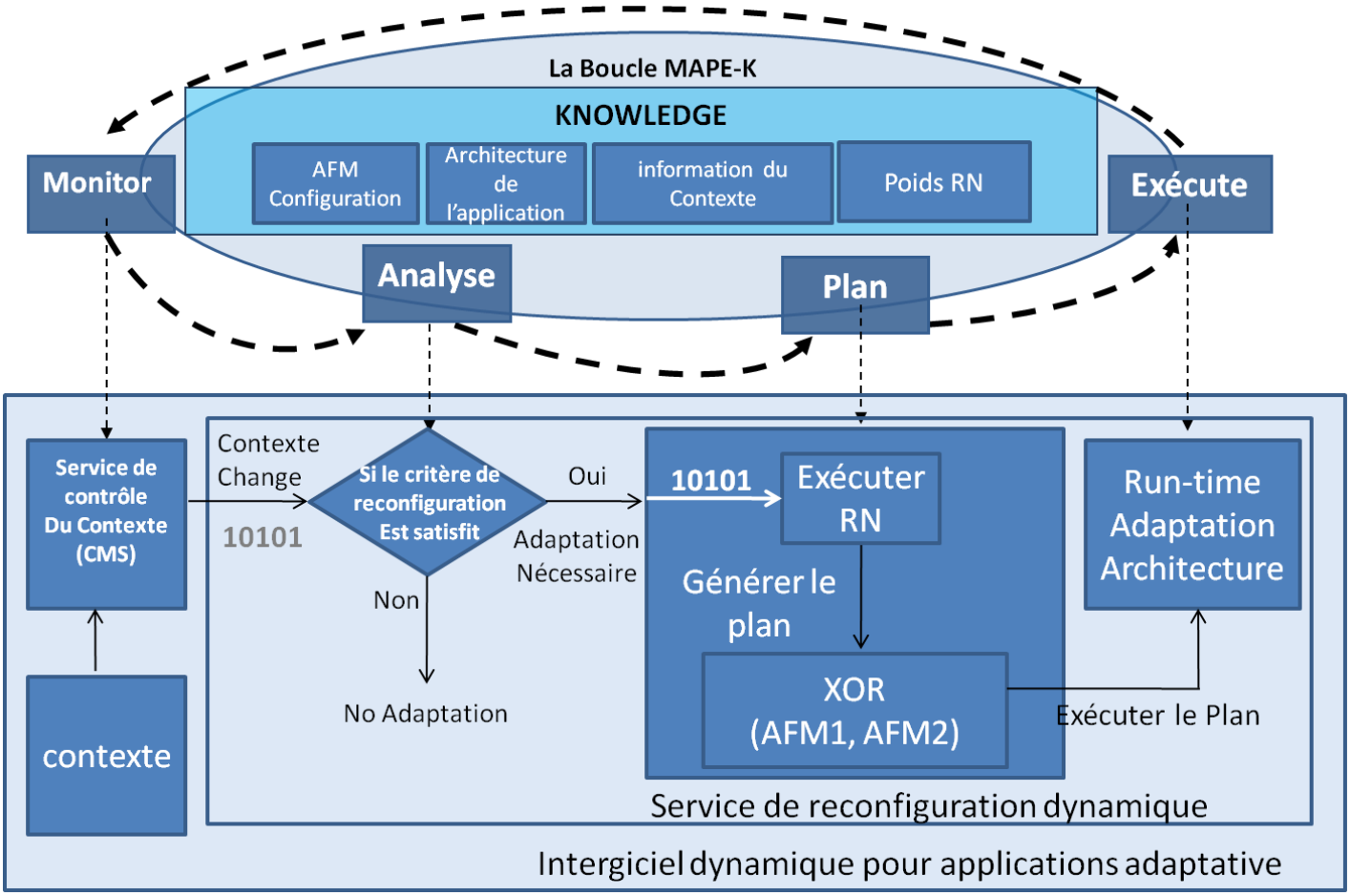

Figure 4.1 : vue globale de l'approche CA-SPL.

2. Adaptation automatique des poids de réseau de neurones

Avant l'utilisation de RN par le service de reconfiguration dynamique (DRS) lors de la phase de l'exécution, il faut tout d'abord adapter ses poids aux différents exemples d'apprentissage (situation, configuration), en exécutant un algorithme d'apprentissage au moment de la conception.

Selon le principe du paradigme de l'informatique autonome (Autonomic Computing AC [IBM, 2006]), les évènements de l'environnement (les situations contextuelles) et les plans de reconfigurations peuvent être généré au moment de la conception, afin de générer des exemples d'apprentissages nécessaire pour l'exécution d'un algorithme d'apprentissage.

La figure (figure 4.2), montre le processus d'adaptation autonome des poids de RN. On a utilisé le principe de l'apprentissage supervisé. L'algorithme d'apprentissage du réseau de neurones a besoin d'exemples (entrée, sortie) où les entrées désignent les informations du contexte qui peuvent produire au moment de l'exécution, et les sorties sont les configurations optimales pour le bon fonctionnement des services sensibles au contexte détecté. Le processus est divisé en quatre (04) sous processus :

1- Le processus de définition des connaissances (figure 4.2, Knowledge): qui consiste de la definition des modèles de variabilités de l'environnement et de l'architecture de l'application.
2- Le processus de génération des évènements contextuels (figure 4.2, Monitor): son objectif est la génération des différentes informations contextuelles valides au moment de la conception, à partir de modèle de variabilité du contexte.
3- Le processus de génération des plans de reconfigurations (figure 4.2, Plan): produire un plan de reconfiguration optimale adéquate au contexte identifié et la composition des exemples d'apprentissages.
4- L'exécution de l'algorithme d'apprentissage (figure 4.2, Execute) : adapte les poids de réseau de neurones aux différentes situations contextuelles. faire apprendre le système aux différentes informations du contexte générées avec leurs configurations optimales (expérience).

L'utilisation du réseau de neurones est la meilleure solution pour faire apprendre le système aux différentes situations du contexte et ses configurations adéquates.

L'apprentissage consiste en l'adaptation des poids du réseau de neurones aux différentes situations contextuelles et leurs configurations adéquates au moment de la conception. Nous proposons un *CMS virtuel* (qui nous génère des informations contextuelles à partir du modèle du contexte) et un DRS pour l'adaptation des poids de RN avec l'utilisation de toutes les étapes de MAPE-K loop.

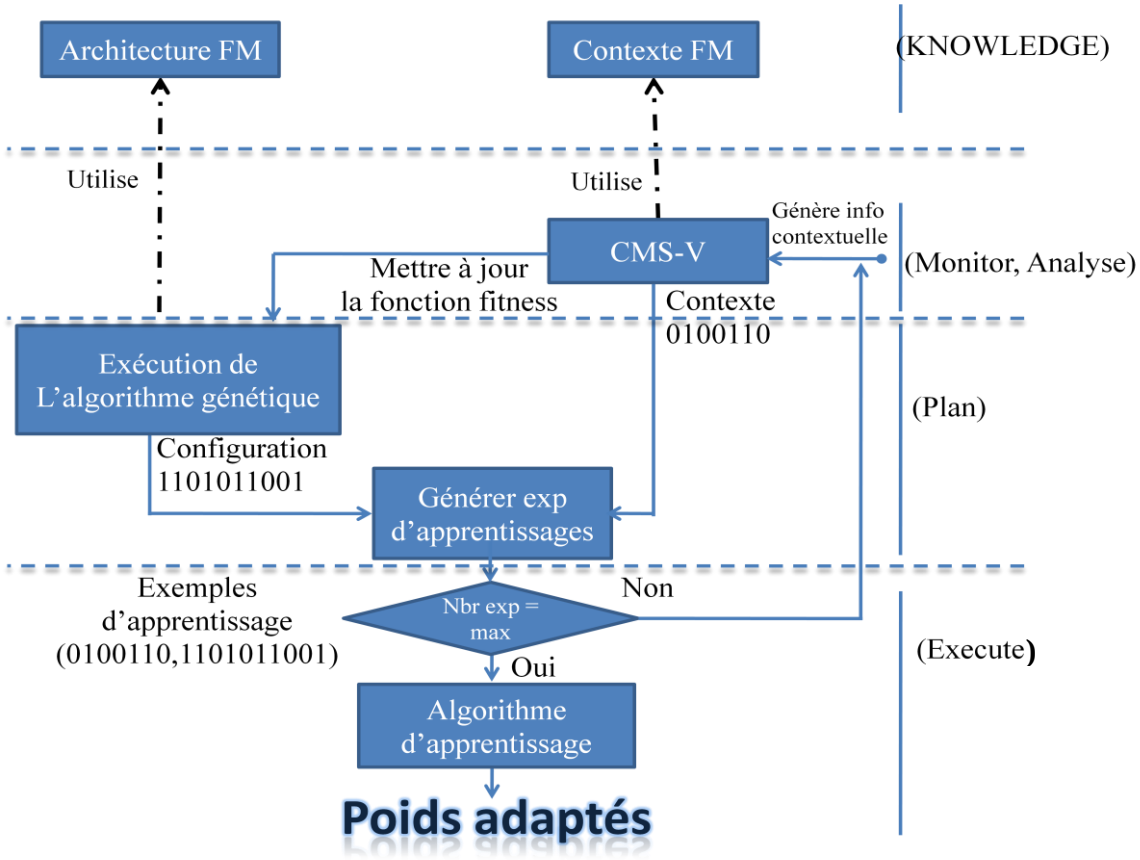

Figure 4.2 : Processus d'adaptation automatique des poids de RN.

2.1 Le processus de définition des connaissances :

La modélisation des systèmes sensibles au contexte a besoin non seulement de la variabilité de systèmes, mais aussi d'un modèle de son environnement et de certaines règles d'adaptation [Mathieu, 2009] qui spécifie qu'elle configuration doit être exécutée dans chaque contexte spécifique. Les variabilités de système sensible au contexte et son environnement sont modélisées en utilisant deux FM indépendants. Puis ils sont connectés par des contraintes de dépendance qui spécifie comment le produit peut s'adapter aux changements de son environnement.

2.1.1 Modélisation des systèmes sensibles au contexte

Avant de construire le FM (Feature Model) d'une famille de systèmes sensibles au contexte, on doit définir les différentes Terminologies, les concepts manipulés, ainsi que le processus de conception [Parra-b, 2011].

a- Abstractions et terminologies

Les concepts sont identifiés, pour être manipulés par le concepteur de l'application sensible au contexte, ils sont utilisés dans le FM pour définir les Caractéristiques (*Features*) utilisées pour la sensibilité au contexte.

1- Capteurs :

C'est la branche qui regroupe la variabilité des capteurs (Logique, physique, virtuelle) utilisés par la famille d'applications pour définir l'état du contexte.

2- Contexte:

C'est la branche qui regroupe la variabilité qui représente ou définit les différents états des entités observées et qui peuvent présenter un sens pour l'application.

3- Dynamique:

C'est la branche qui regroupe la variabilité qui représente les caractéristiques dynamiques. Ces caractéristiques appartiennent au **noyau** fonctionnel (le code métier), mais qui dépendent d'un contexte donné.

4- Noyau:

C'est la branche qui regroupe la variabilité qui représente les besoins identifiés dans la phase de spécification (les besoins fonctionnels des clients).

b- définition du FM des Systèmes sensibles aux contextes

Afin d'enrichir FM (FM comme connaissance [García, 2013]) avec des informations nécessaires pour la configuration autonome, on suit un processus de conception des applications qui comporte des étapes permettant de prendre en compte la sensibilité au contexte et l'adaptation des applications dans un environnement ubiquitaire. Ce processus définit les acteurs, leurs activités, ainsi que les branches du FM produites à chacune des étapes. Les étapes identifiées sont les suivantes :

Etape 1 : Conception des variabilités (branche *Noyau*) : définition des éléments réutilisables communs et variables d'une famille de produits et construit la branche FM du *Noyau*.

Etape 2 : Conception des caractéristiques dynamiques (*branche Dynamique*): Définition à partir des caractéristiques fonctionnelles (Noyau), des caractéristiques dynamiques (*une caractéristique dynamique, est une caractéristique qui dépend du contexte pour sont exécution*) en les déplaçant vers la branche FM Dynamique tout en gardant un lien avec ses *caractéristiques* parents situés dans la branche (Noyau).

Etape 3 : Conception du contexte (*branche Contexte*): définition des différentes situations (*variabilité du contexte*) à surveiller (observable [Taconet, 2011]) présentées dans l'environnement pouvant être observées.

Etape 04 : Conception des capteurs (*branche capteur*): pour chaque situation identifiée, on introduit les capteurs nécessaires à leur interprétation on les associant à la branche FM (Capteurs*).*

Etape 05 : Conception du système sensible au contexte (*FM globale*):

Elle consiste à construire un seul FM qui a comme racine « Systèmes sensible au contexte » et qui contient les différentes branches construites auparavant. Elle définit les différentes contraintes qui existent entre les caractéristiques définies avant. Ces contraintes sont les suivantes :

- **Contrainte (*Noyau – Dynamique*)** : elles définissent un lien entre les caractéristiques de la branche *Dynamique* avec leurs caractéristiques *parents* contenues dans le FM *Noyau.*
- **Contrainte (*Dynamique – Contexte*)** : ce sont les contraintes (les règles d'adaptations) qui relient chaque caractéristique dynamique avec les situations convenables définies dans le FM Contexte.
- **Contrainte (*Contexte – Capteurs*)** : ce sont les contraintes qui définissent pour chaque situation les capteurs nécessaires pour son interprétation.

2.1.2 La gestion de la variabilité des systèmes sensibles au contexte

La première étape dans le processus de conception est la définition des connaissances nécessaires pour la boucle MAPE. Celles-ci incluent :

(1) Le modèle de caractéristiques du système sensible au contexte : Il comporte les variabilités de l'application et celle du contexte, ainsi que les contraintes (les règles d'adaptations) définies entre les caractéristiques dynamiques de l'application et les caractéristiques du contexte.

(2) Les informations contextuelles sont: les différentes situations du contexte de l'environnement définies dans le *FM contexte*. Elles sont représentées selon la formule suivante :

Contexte= Union (états des situations pour chaque entité observée).Un état de situation peut être actif (=1) ou inactif (-1).

Par exemple, le modèle de caractéristiques du contexte (voir Figure 4.3) exprime la variabilité des situations des deux entités à observer : niveau de la batterie et le type de connexion. Ces deux entités appartiennent à la dimension **terminale**. Les situations significatives du niveau de la batterie pour l'application sont : Bas (niveau≤20), Moyen (20<niveau≤60), Haut (niveau >60 ou batterie en charge). Les situations de connexion sont soit par wifi ou soit par Bluetooth. Les situations sont de type alternatif pour chaque entité.

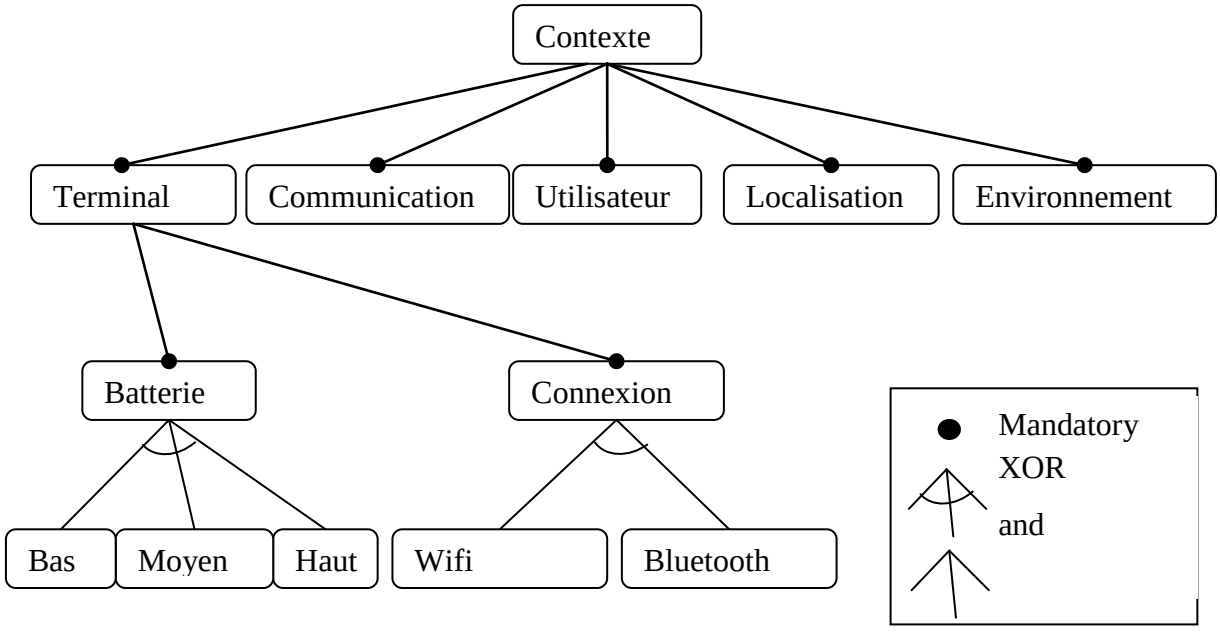

Figure 4.3 : exemple de model de caractéristiques de la branche contexte qui présente la variabilité du contexte pour une application mobile.

A chaque moment (en exécution), une seule situation, parmi les différentes situations, appartenant à une entité est active (=1) le reste, sont inactives (=-1).Par exemple, le niveau de la batterie égal à 40 et le type de la connexion est wifi (tableau 4.1).

Entités	Batterie			connexion	
situation	Bas	Moyen	Haut	Wifi	Bluetooth
Contexte	-1	1	-1	1	-1

Tableau 4.1 : représentation binaire des états des situations du contexte.

C'est alors qu'on peut représenter le contexte par une succession des valeurs binaires qui expriment les informations contextuelles des situations des entités observées. Dans l'exemple de la table 4.1, nous avons un Contexte = {-1,1,-1,1,-1}. Cette représentation nous permettra de l'utiliser dans l'algorithme génétique et aussi dans le réseau de neurones.

2.2 Le processus de génération des évènements contextuels

Dans l'approche de [García, 2013], les auteurs utilisent CMS pour fournir des informations concernant la situation de certaines ressources observées au DRS au moment de l'exécution. Dans notre approche le ***CMS Virtuel*** a comme objectif la génération des informations contextuelles conformes au modèle de variabilité de la branche du *contexte*. Car au moment de la conception, on ne peut pas connaitre les différentes situations du futur contexte. De plus, désigner ou définir les différentes situations possibles du contexte, où une application va se trouver, est aussi une tâche difficile pour couvrir toutes les situations du contexte dans les lignes des produits traditionnelles.

2.2.1 CMS Virtuel

Le CMS virtuel a comme objectif la génération des différentes informations contextuelles possibles au moment de la conception. Les différentes situations des entités (ou ressources) nécessaires pour le fonctionnement optimal des différents services sont déclarés dans le modèle de caractéristiques de la branche « contexte ». L'algorithme de génération d'une information contextuelle (CMS virtuel) se divise en trois (03) étapes.

a. Identifier et regrouper les différentes situations du contexte :

Dans cette étape, CMS Virtuel parcoure la branche contexte du modèle de caractéristiques et identifie les différents entités observées (caractéristique non-terminal C_P) qui seront placées dans la liste LC_P ***(algorithme 1** : la boucle dans la ligne 2).* Pour chaque entité observée C_P , on identifie ses différentes situations (caractéristiques Terminal C_T) et on les places dans une liste LC_T***(algorithme 1** : la boucle de la ligne 6).* Les caractéristiques (terminales), fils d'une caractéristique (non-terminal) parent, sont en relation de type *XOR.* Parce qu'à un moment donné, chaque ressource prend une seule situation.

b. Activation des situations :

Chaque ressource doit prendre une situation qui explique son état à un moment donné (itération), c'est-à-dire pour chaque $C_P \in F_C$ *(caractéristiques du contexte)* on doit en activer une seule C_T, où ($C_T \in C_P$) d'une façon aléatoire, et les mettre dans la liste *Lactf (algorithme 1 :la boucle dans la ligne 11).*

c. Composition du contexte :

Après la construction de l'information du contexte (l'ensemble des situations activées et non activées) le CMS virtuel va concaténer les différentes situations (actives et non-actives) dans une chaine binaire *ctx (algorithme 1 :la boucle dans la ligne 14)* où chaque nombre binaire désigne l'état d'une situation (1= actif, 0 ou -1= non-actif). L'ordre des nombre binaire dans *ctx* est le même ordre des caractéristiques (terminal) du contexte déclaré dans la branche du contexte.

Le nombre des situations du contexte généré par cet algorithme (***Nbr_ctx***) égal au produit de nombres de situations de chaque entité.

2.3 Le processus de génération des plans de reconfigurations

Pascula et al [García, 2013] définissent le DRS comme une approche qui suit les étapes de la ligne de produits logiciels dynamiques (DSPL). Parce que l'intérêt de l'approche DSPL consiste en la production des logiciels qui sont capables de s'adapter selon le changement des besoins de l'utilisateur et de l'état de l'environnement [Hallsteinsen, 2008]. Comme les caractéristiques concernées par ce changement sont modélisées dans la branche dynamique du model de caractéristiques FM, le processus de reconfiguration s'intéresse seulement à ces caractéristiques dynamiques.

Au contraire de [García, 2013], qui exécutent le service de reconfiguration dynamique DRS pour générer une configuration au contexte détecté durant l'exécution. Son exécution au moment de la conception nous sert comme objectif d'adapter les poids du réseau de neurones (apprentissage) et non pas de reconfigurer l'application au moment de la conception ou de l'exécution. Selon [García, 2013] Le principe du DRS se divise en deux étapes, comme illustré dans l'algorithme 2 : analyse et plan.

Algorithme 1 :

CMS_virtuel(F_C)

Input : F_C
Output : Liste ctx //context généré.
Liste LC_P <C_P> ← ∅ ;
Liste LC_T , *Lactf* <C_T> ← ∅ ;
Liste *Lctx* <*ctx*> ← ∅ ;
1 Nbr_ctx ← nombre_des_situation_possibles(Fc) ;
2 **Foreach** *feature* $f \in F_C$ **do**
3 **if** f is not a terminal feature **then**
4 $C_P \leftarrow f$; LC_P.add(C_P);
5 **end**
6 **Foreach** *feature* $f \in F_C$ **do**
7 **if** f is a terminal feature **then**
8 $C_T \leftarrow f$; LC_T.add(C_T);
9 **end**
10 **while** length (Lctx) <= Nbr_ctx **do** Lactf, ctx← ∅ ;
11 **Foreach** $C_P \in LC_P$ **do**
12 Lactf← randomly select a children feature of C_P;
13 **end**
14 **Foreach** $C_T \in LC_t$ **do**
15 **if** $C_T \in$ *Lactf* **then**
16 ctx←ctx+"1";
17 **else** ctx←ctx+"-1";
18 **end**
19 **if** ctx is not ∈ Lctx **then**
20 Lctx.add(ctx);
21 DRS(ctx);
22 **end**;

Algorithme 02 :

DRS (ctx)

Input : FM, ctx;
Output : poids;
1 **If** *Analyse (ctx) = true* **then//** …….. analyse
2 Conf←Plan (*ctx*);// ……. Le plan
3 End
4

a. Analyse

A chaque notification d'un changement du contexte envoyée par *CMS virtuel*, l'analyse se confirme si ce contexte est nouveau ou non *(algorithme 02 : ligne 01)*.S'il s'agit d'un nouveau contexte, il exécute le *plan*, sinon il attend une autre notification. A chaque nouveau contexte reçu *ctx*, l'analyse le compare avec l'historique des contextes reçus et enregistrés

dans la liste du contexte *Lctx*. *Si ctx* est un nouveau contexte, il le déplace dans la liste *Lctx* et exécute le plan avec l'envoi de *ctx* comme paramètre *(algorithme 03)*.

Algorithme 03 :

analyse (ctx)

```
        Input : ctx;
        Output : confirmation: Boolean;
1       Liste  Lctx  <ctx>  ← ∅ ;
2          If  ctx is not ∈ Lctx  then
3             Lctx.add(ctx);
4       Confirmation=true;
        End
```

b. Plan

Dans le cas d'un nouveau contexte non traité, l'analyseur décide si ce contexte à besoin d'être traité. C'est-à-dire, la génération de sa configuration optimale qui le convienne. La sélection d'une caractéristique dynamique est liée selon l'état de la situation du contexte (*la règle définie pour la sélection d'une caractéristique dynamique qui définie la situation ou un groupe de situations nécessaires pour son activation*). Le DRS doit décider quelle est la configuration qui est adéquate au contexte généré (états des ressources).

Guo et al. Dans [Guo, 2011] formulent ce problème comme étant un problème d'optimisation, avec l'utilisation d'un algorithme génétique AG pour optimiser la sélection d'un sous ensemble de caractéristiques définies dans le FM et qui sont liés aux états de certaines ressources. Ils ont aussi défini une configuration selon un codage binaire utilisé pour représenter une solution potentielle comme chromosome (ex : 1OO1O1O1) où chaque nombre présente la sélection ou la désélection d'une caractéristique.

Dans notre projet, nous utilisons le même algorithme génétique GA afin de trouver une configuration optimale selon le nouveau contexte généré, mais avec une nouvelle représentation des états des ressources, tableau 4.1 représente le contexte comme une succession de situations des ressources, tel que ça été fait dans [Chaari, 2007], une ressource peut prendre plusieurs situations au cours de son exécution, mais à un moment donné, elle prend une seule situation. Par exemple, le niveau de la batterie au moment de l'exécution peut être bas, moyen ou haut, mais à un moment donné, il prend exclusivement une seule situation. Alors le contexte de la batterie sera représenté comme suit :

Contexte $_{\text{Batterie}}$ = {**bas**=actif, **moyen**=inactif, **haut**=inactif}

La représentation binaire de l'information du contexte du même exemple est : $contexte_{batterie}$= {1,-1,-1}, chaque nombre représente l'état d'une situation au moment de l'exécution.

b.1 Matrice de dépendances

La matrice de dépendance DM (Voir tableau 4.2) est une matrice à deux dimensions. Elle traduit les contraintes (les règles d'adaptations) définies entre les caractéristiques dynamiques (branche dynamique du FM) et les caractéristiques du contexte (branche contexte) où les colonnes *j* sont les situations et les lignes *i* sont les caractéristiques (assets). Une cellule DM [*i,j*]représente la sensibilité d'un composant a une situation : si DM [i,j]=1 alors le composants *i* dépend de la situation du contexte *j* .autrement dit, une colonne j représente les composants reliés à une situation, et une ligne i représente les différentes situations nécessaires (contexte)pour l'activation d'un composant donné.

	Contexte				
Features	Sit_1	Sit_2	…	Sit_m-1	Sit_m
Comp_1					
Comp_2					
…					
Comp_n-1					
Comp_n					

Tableau 4.2 matrice de dépendance.

Exemple : une application mobile qu'utilise la connexion avec wifi ou Bluetooth pour transmettre un fichier avec d'autre application mobile, l'utilisation de ces deux types de connexion est liée au niveau de la batterie, si le niveau de la batterie est haut en préfère le Bluetooth, si le niveau est bas ou moyen en préfère le wifi. Le FM de système est le suivant (figure 4.4).

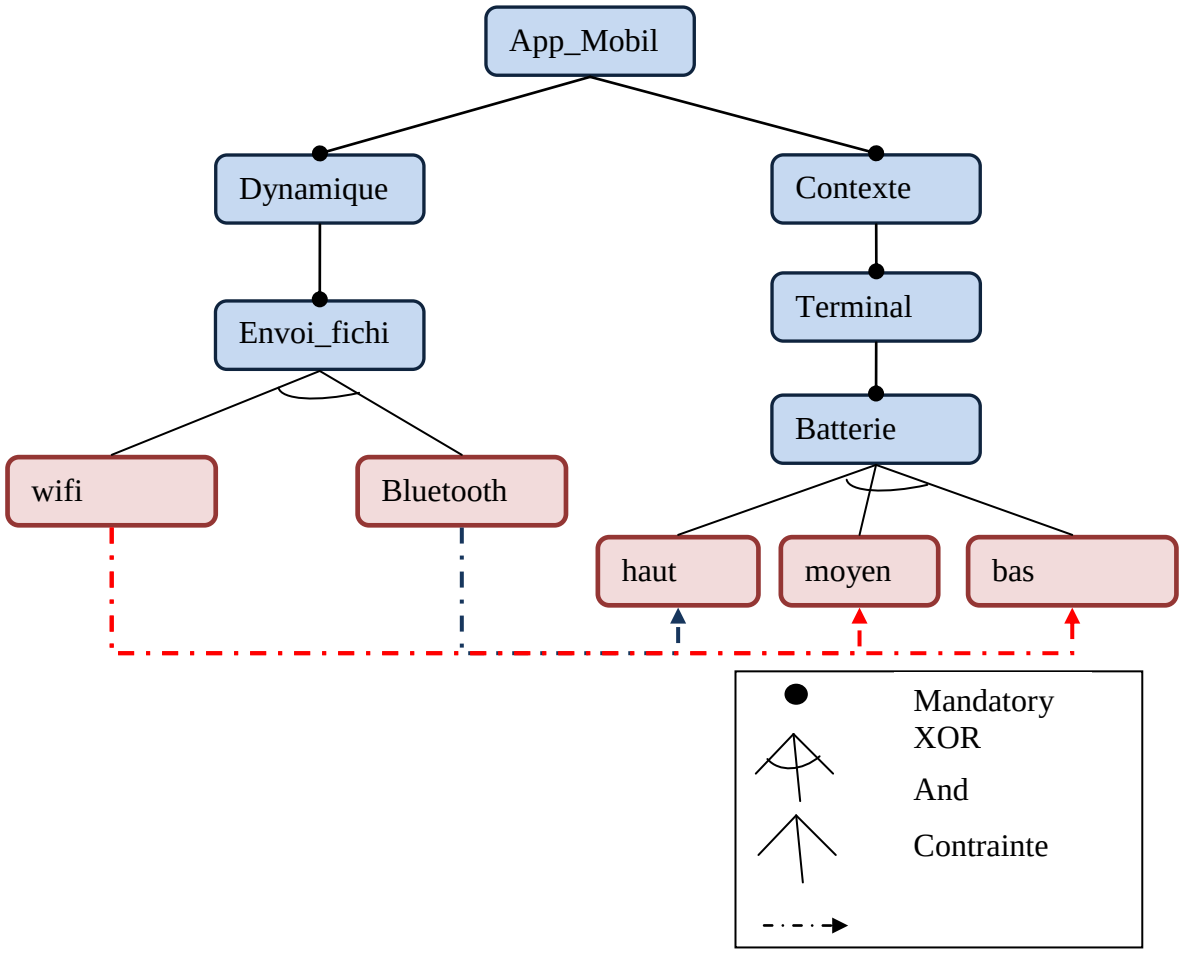

Figure 4.4: FM pour une application mobile contenant deux branches (dynamique et contexte).

Mat_Dep		Niveau de batterie		
		Bas	moyen	Haut
Caractéristiques dynamiques	wifi	1	0	0
	Wifi	0	1	0
	Bluetooth	0	0	1

Tableau 4.3 : Exemple de matrice de dépendance.

A partir du tableau 4.3, le contexte de sélection de la caractéristique wifi est :

contexte = {bas=1 ;moyen=0 ;haut=0}, contexte={bas=0 ;moyen=1 ;haut=0}.

Et pour Bluetooth :

Contexte = {bas=0 ; moyen=0 ; haut=1}.

b.2 Représentation des contraintes dans la matrice de dépendance

La matrice de dépendances définit le contexte d'exécution (les règles d'adaptations) pour chaque caractéristique dynamique (ligne) déclarée dans FM. Une contrainte (*Cross-tree constraints*) est utilisée pour représenter les règles de composition non-hiérarchique dans FM, qui comprend les relations de dépendance mutuel (requires) et l'exclusion mutuel (excludes) [Kang, al]. Elle exprime les contraintes de chaque caractéristique dans une ligne, qui contient les différentes valeurs des situations (colonnes) définies dans une contrainte pour qu'elles soient *vraies*. La représentation des différentes opérations sur les contraintes est comme suit :

Implies : déclare les situations nécessaires représentées par 1.
Excludes : déclare les situations qui ne doivent pas exister dans le contexte, et est représentée par -1.

Une caractéristique qui définit une contrainte qui ne contient que l'opérateur logique *and,* est représentée dans la matrice avec une seule ligne, avec l'activation des situations dans les colonnes qui convient (1 ou -1).Par exemple, soit F la caractéristique dynamique qui est liée aux situations avec cinq (05) situations différentes d'un contexte S. sous la forme suivante :

F **Implies** (S1 and S3 and S5and (not S2)).

Features	S1	S2	S3	S4	S5
F	1	-1	1	0	1

Une caractéristique définissant une contrainte qui contient l'opérateur logique *or* sera représentée par plusieurs lignes, où chaque ligne représente une condition alternative.

F **Implies** (S1 and (S3 or S4)).

Feature	S1	S2	S3	S4	S5
F	1	0	1	0	0
F	1	0	0	1	0
F	1	0	1	1	0

Une caractéristique définissant une contrainte qui contient l'opérateur logique *xor* sera représentée dans la matrice avec plusieurs lignes comme *or*, ou chaque ligne représente une condition alternative avec Excludes des autres conditions.

F **Implies** (S1and (S3 xor S4)).

Feature	S1	S2	S3	S4	S5
F	1	0	1	-1	0
F	1	0	-1	1	0

b.3 Algorithme génétique CA-SPL

Nous utilisons l'algorithme AG *GAFES* (*une approche GA pour optimiser la sélection des caractéristiques dans SPLS*) décrit dans [Guo, 2011], avec notre fonction fitness d'évaluation des caractéristiques sélectionnées (figure 4.5) ayant pour objectif de minimiser l'erreur de la fonction d'optimisation. Elle pénalise les solutions qui ne respectent pas les contraintes des ressources, la meilleure solution est celle qui minimise l'erreur de reconfiguration. La fitness d'un chromosome est définie comme étant la somme des erreurs de sélection des caractéristiques.

1- Codage de chromosome de caractéristiques (Feature chromosome encoding)

[Guo, 2011] Définit un chromosome dans *GAFES*, qui représente la sélection des caractéristiques dans un FM. Pour un FM avec *n* caractéristiques, chaque chromosome est composé de n gènes et chaque gène représente la sélection d'une caractéristique. Il définit une variable de type caractère (*String*) avec *n* chiffres binaires (n binary digits) utilisé pour coder un chromosome. Chaque chiffre binaire représente une caractéristique et une valeur de 1 pour le chiffre qui indique que la caractéristique est sélectionnée (0 est non-sélectionnée).

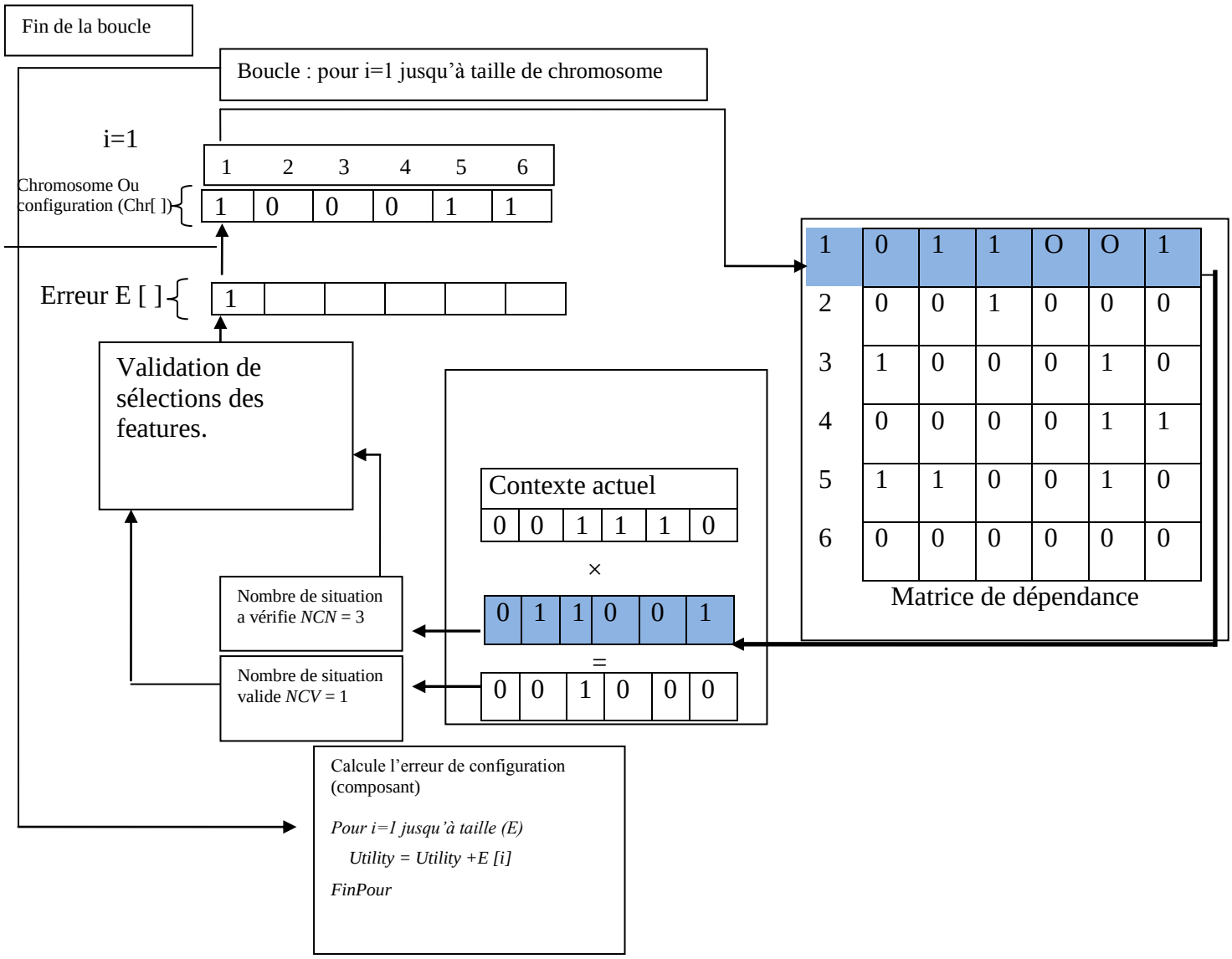

Figure 4.5 : Schéma de calcule de la fonction d'évaluation.

2- L'évaluation d'une configuration

L'évaluation d'une configuration est divisée en deux étapes :

Etape 01 : validation de sélections d'une caractéristique.

Pour chaque gène (caractéristique) du chromosome (configuration), on consulte les informations du contexte nécessaires pour sa sélection qui sont définies dans la matrice de dépendance. Pour chaque vecteur du contexte *V_ctx* (ligne) identifié dans la matrice *Mat_Dep*, on fait la comparaison avec celui généré (*V_ctx_g*). Le résultat de la comparaison est enregistré dans le vecteur *V_Comp (*les éléments du vecteur sont des nombres binaires). Nous avons alors :

$V\_Comp_i = V\_ctx_i \times V\_ctx\_g_i$.

Etape 02 : évaluation d'une configuration.

Cette étape consiste à calculer l'erreur d'une configuration comme suit :

- *NCN* le nombre de situations à vérifier de caractéristique F*i* , *NCN* = nombre des situations autres que 0 (situations nécessaires) défini dans le vecteur *V_ctx*.
- *NCV* est la somme de situations vérifiées. *NCV*= somme des situations définies dans le vecteur *V_Comp*.
- *E est* le vecteur *d'*erreur de sélection ou non des Caractéristiques E*i,*

$\boldsymbol{E_i}$ représente l'erreur de sélection de $\boldsymbol{F_i}$, *est égale à 0 si :*

- le contexte requis pour F_i est défini dans le contexte capturé .
- F_i est sélectionnée.
- le parent de F_i diffère de *xor.*

$\boldsymbol{E_i}$ représente l'erreur de sélection de $\boldsymbol{F_i}$, *est égale à 1 si :*

- le contexte requis pour F_i est défini dans le contexte capturé.
- F_i n'est pas sélectionnée.
- le parent de F_i diffère de *xor*.

Si au contraire, le contexte requis pour F_i est non-défini dans le contexte capturé, et F_i est sélectionnée alors $E_i=1$.

L'erreur d'une configuration est égale à la somme de ses éléments : Erreur $=\sum E_i$

2.4 L'exécution de l'algorithme d'apprentissage

L'exécution consiste à faire un apprentissage du système aux différentes situations du contexte et leurs configurations adéquates pour lui donner la capacité de proposer des configurations exactes aux situations détectées dans un temps raisonnable. La meilleure solution de configuration de l'application est obtenue à l'aide de réseaux de neurones. Parce que les exemples utilisés pour l'apprentissage du réseau de neurones sont exactes et le temps de calcul d'une configuration est minimisé au moment de l'exécution.

2.4.1 Modélisation du réseau de neurones pour l'application sensible au contexte.

Le fonctionnement général d'un neurone artificiel est modélisé selon le schéma de la Figure 4.6, Notre réseau de neurones doit pouvoir calculer les valeurs de sorties (Configuration) en fonction des variables d'entrées (situations). Le réseau de neurones

contient plusieurs couches (série) inter connecté. Chaque série a sa propre fonction d'activation appliquée à ses entrées. Le résultat de chaque série sera une entrée pour la seconde série, sauf la dernière qui représente le résultat final exact (configuration).

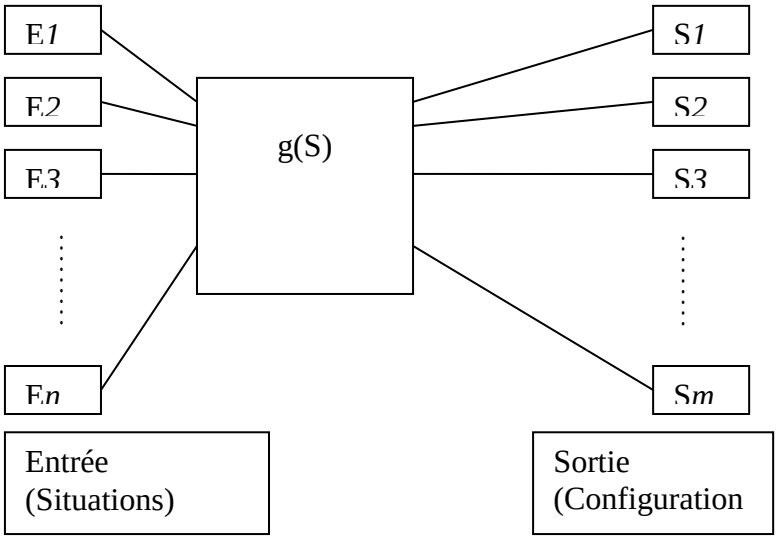

Figure 4.6 : Schéma général d'un réseau de neurones.

2.4.2 L'apprentissage du réseau de neurones

Une caractéristique des réseaux de neurones est la capacité à apprendre (reconnaître une configuration exacte à la situation détectée). Mais cette connaissance n'est pas acquise dés le départ, ces réseaux apprennent à travers des exemples, en suivant un algorithme d'apprentissage. On choisi l'apprentissage supervisé, c'est-à-dire que les valeurs que l'on désire que le réseau obtienne en sortie sont corrects. Après l'apprentissage (ajuster ses poids de connexions), le réseau est testé en lui donnant seulement les valeurs de situations en entrée, tout en observant si le résultat obtenu (configuration) est proche du résultat désiré.

Conclusion :

L'approche que nous avons proposée, permet de développer un *intergiciel* pour l'adaptation des applications au changement du contexte, selon l'approche développée dans [García, 2013], pour donner aux systèmes mobiles la capacité d'adaptation autonome au moment de l'exécution. Une telle approche consiste en :

a- Capture des informations de l'état du contexte.
b- Analyse des informations capturées,
c- Génération d'un plan de reconfiguration,
d- exécution du plan de reconfiguration.

[García, 2013] propose un algorithme génétique AG au moment de l'exécution (étape c- ci-dessus).Pour les appareils mobiles, l'utilisation d'un AG au moment de l'exécution prend beaucoup de temps de calcule et occupe un espace mémoire important. A la place d'une telle solution, nous avons recours à un réseau de neurones artificiel avec des poids adaptés au moment de la conception, afin de générer une configuration optimale selon le contexte détecté. Le choix d'une telle solution, permet par rapport à un AG, de réduire considérablement le temps d'exécution, l'espace mémoire utilisé et la consommation de l'énergie (appareil mobil). Notre approche utilise aussi un AG. Cependant, celui-ci a été utilisé au moment de la conception pour optimiser des configurations adéquates aux différentes informations du contexte possibles. L'ensemble de ces informations générées et leurs configurations adéquates, servent comme des exemples d'apprentissages au réseau de neurones pour adapter les poids aux différentes situations du contexte.

Avec notre approche CA-SPL, une configuration optimale est rapidement générée au moment de l'exécution à l'aide d'un réseau de neurones. Les configurations produites par le RN sont générées au moment de la conception avec l'utilisation d'un GA.

Chapitre V

Cas d'étude

Sommaire

Introduction :

L'adaptation dans CA-SPL, consiste à remplacer la configuration courante d'une application par une nouvelle configuration cible. Le choix d'une configuration est obtenu en utilisant les informations du contexte capturées durant l'exécution. Afin de réduire le temps de la décision (la sélection d'une reconfiguration) au moment de l'exécution, nous avons utilisé un RN pour générer une reconfiguration (sortie RN) selon les informations du contexte capturées (entrées RN). La qualité des résultats produits par le RN dépend des exemples (contexte, configuration) fournis à l'algorithme d'apprentissage afin de générer des poids adaptés au moment de la conception.

Les exemples sont un couple (contexte, configuration). Les informations du futur contexte sont générées à partir des variabilités définies dans le FM de la branche contexte (voir chapitre 4 – section 3).Tandis que la configuration est générée en utilisant un algorithme d'optimisation AG, où la variabilité d'une configuration, dépend des caractéristiques identifiées, des dépendances et des contraintes définies dans la branche dynamique FM.

1. Cas d'étude : un système de notification

Nous allons créer un scénario de ***notification*** des ***tâches*** ou des ***appels*** de la journée d'un utilisateur enregistré dans un agenda. Ce dernier est installé dans un *appareil* mobile. Le système de *notification* (*tâches* et *appels*) dépend des changements du contexte de l'utilisateur :

1- l'endroit (*point de variation PV)* ***endroit*** où il se trouve : dans la ***voiture***, à la ***maison***, au ***travail*** ou en ***réunion***. Ces éléments constituent des Variantes,
2- L'état de la *batterie* (PV) : *faible*, *moyen* et *haut*),
3- ***notification*** des ***appels*** : ***activée*** ou ***désactivée*** (ceci dépend de *l'endroit* de l'utilisateur et *l'importance* de l'appel).

Les appels sont transférés vers le système audio du véhicule *S_Audio_Vehicule* si l'utilisateur conduit sa voiture, sinon il utilise la sortie ***standard*** (appareil mobile). La notification (appels ou tâches) se fait avec une sonnerie ou vibreur, tant que l'utilisateur est en *réunion*, la signalisation des appels se fait via le *vibreur*. Si l'utilisateur est dans sa voiture, à la maison, ou au travail, la notification se fait avec une sonnerie. Dans ce dernier cas, le volume est :

- ***haut :*** si le lieu est la ***maison*** ou si la personne est ***endormie,*** avec un niveau de la batterie (haut ou moyen),
- ***moyen :*** si le lieu est la voiture et le niveau de la batterie (haut ou moyen),
- ***faible :*** si le lieu est le travail ou le niveau de la batterie (faible).

Les tâches sont notifiées par un :

- ***message*** à afficher si l'endroit n'est pas dans la voiture,
- ***orale*** sinon.

L'importance du scenario choisi est de proposer les différentes configurations (caractéristiques dynamiques) et les variations du contexte ainsi que les contraintes qui compliquent pour définir l'adaptation.

2. Défie

L'exemple présenté, utilise des informations d'adaptation qui ne sont disponibles qu'au moment de l'exécution. Notre objectif au moment de la conception, est de générer les informations du futur contexte. La variabilité de l'information du contexte peut être,

modélisée par FM de SPL, pour décider quelle est la reconfiguration à choisir, pour chaque situation particulière du contexte.

CA-SPL supporte la génération des reconfigurations adéquates pour chaque situation du contexte identifiée, d'une manière autonome à l'aide des connaissances acquises, à partir de FM, au moment de l'exécution.

CA-SPL supporte la dérivation dynamique des produits au moment de l'exécution, qui est capable de remplacer la configuration courante par une nouvelle. Pour accomplir cette autonomie et l'importance des connaissances générées au moment de la conception, nous avons identifié plusieurs défis :

a- **Représentation des connaissances à l'aide de FM :**
   Chaque produit sensible au contexte dans la SPL est représenté par deux branches (application et contexte).

b- **Génération des informations du futur contexte au moment de la conception :**
   La variabilité du contexte définie dans FM nous permet de parcourir toutes les situations possibles du futur contexte,

c- **La gestion d'un nombre important de caractéristiques définis dans FM :**
   Afin de générer d'une façon autonome une configuration optimisée pour chaque situation du contexte identifiée.

d- **Transférer l'expérience acquise :**
   L'ensemble des informations du contexte générées et leurs configurations, sont transférées vers des données exploitables par un RN qui sera utilisé au moment de l'exécution (poids adapter).

3. Cycle de vie d'apprentissage du réseau neuronal

Pour bien comprendre le processus d'optimisation d'une reconfiguration pour chaque situation du contexte identifié, et l'apprentissage du réseau de neurones artificiels(RN), nous commençons par la description de la relation qui existe entre le processus d'optimisation de configuration pour chaqu'une des situations du contexte identifiées et le processus d'apprentissage, ainsi que les étapes de fonctionnement dans chaque système (voir Figure 5.1).

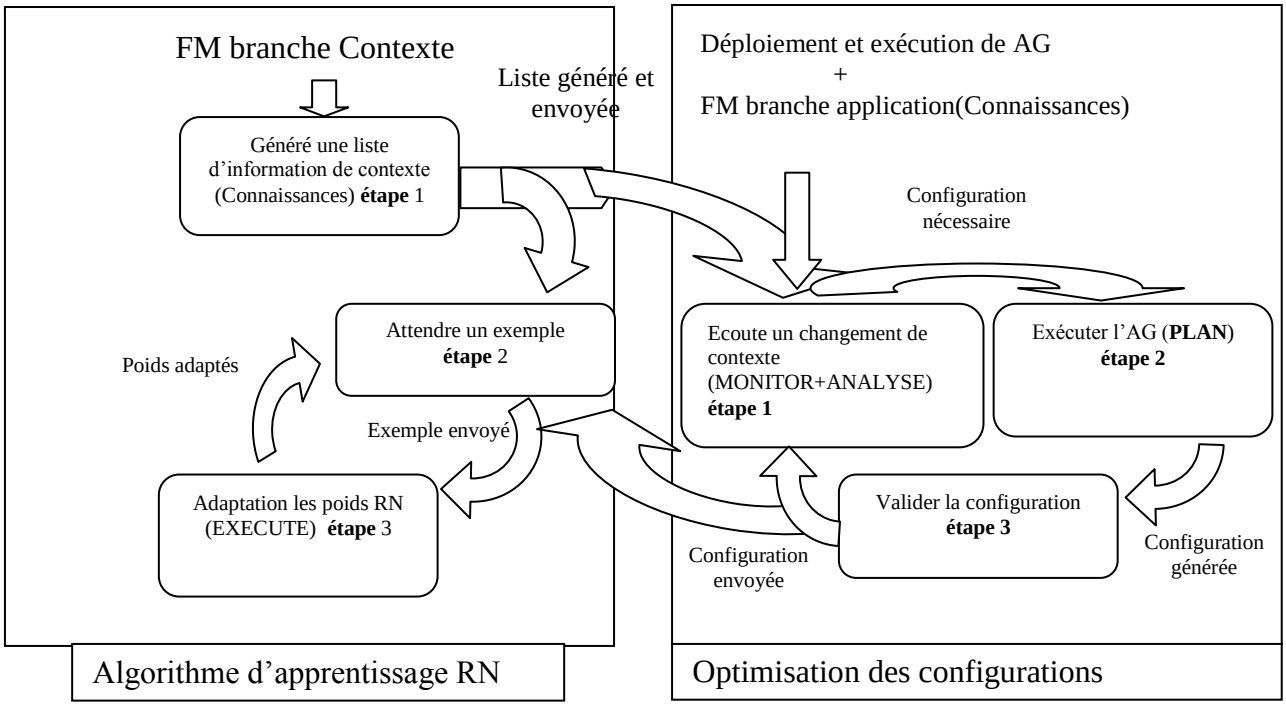

Figure 5.1: Cycle de vie d'adaptation des poids de RN.

a- Cycle de vie de l'algorithme d'apprentissage

Avant d'utiliser le RN, il faut adapter ses poids. Ce processus consiste à adapter les poids du RN à l'ensemble des exemples d'apprentissages produits.

Etape 1 : La génération d'une liste d'informations du futur contexte à partir de la branche contexte du FM.

Etape 2 : Composition des exemples :

Les situations du contexte générées et leurs configurations optimales adéquates, sont composées afin de générer des exemples d'apprentissages.

Etape 3 : Adaptation des poids :

Le RN exécute un algorithme d'apprentissage pour l'ensemble des exemples générés à l'*étape* 2.

b- Cycle de vie de l'optimisation des configurations

Ce processus consiste à générer une sélection de caractéristiques optimisées (configuration) pour chaque situation du contexte identifié, afin de composer des exemples d'apprentissages.

Etape 1 : écouter le changement du contexte :

Pour chaque situation générée, une adaptation est nécessaire. On passe alors à l'étape 2 pour exécute l'algorithme génétique.

Etape 2 : exécuter l'algorithme génétique :

Dans cette étape l'algorithme génétique est exécuté pour générer une configuration optimale appropriée au contexte détecté. A la fin de cette exécution, on passe à l'étape 3.

Etape 3 : Génération d'un exemple d'apprentissage :

Après avoir produit une configuration optimale pour chaque situation, un exemple est construit pour être utilisé par l'algorithme d'apprentissage.

4. Cycle de vie de l'adaptation d'une application

Pour bien comprendre le processus d'adaptation dynamique de l'application au contexte détecté au moment de l'exécution, nous commençons par la description du processus d'adaptation autonome des applications dérivées par ASPL et les étapes de fonctionnement de chaque système comme illustré dans la figure 5.2.

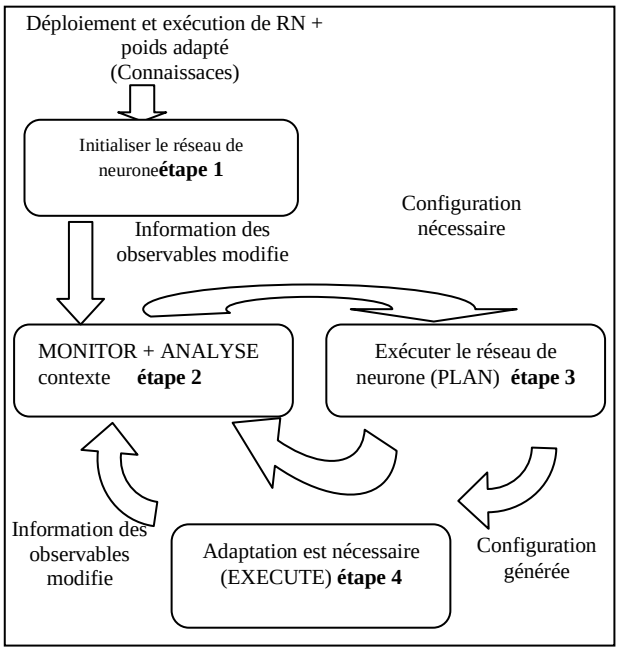

Figure 5.2: Cycle de vie d'une adaptation

Etape 1 : après l'apprentissage du réseau de neurones avec les différents exemples (contextes, configurations, les poids seront transférés vers RN exécuté par *intergiciel* de gestion du contexte.

Etape 2 : Après le changement des informations observables, l'intergiciel décide si une reconfiguration est nécessaire ou non.

Etape 3 : Si la reconfiguration est nécessaire, exécution du plan de reconfiguration. Pour cela, on utilise le RN avec comme entrée les informations du contexte et comme sortie

une reconfiguration. Cette dernière est comparée avec la configuration actuelle afin de décider si une adaptation est nécessaire ou non.

Etape 4 : Si une adaptation est nécessaire: l'architecture de l'application est adaptée selon la configuration générée à l'étape 3, en appliquant un mécanisme de reconfiguration dynamique (reconfigurateur). Puis il revient à l'étape 2.

5. Phase de conception : l'apprentissage du réseau de neurones

Cette phase consiste à adapter les poids du RN aux différents exemples d'apprentissages (les situations du futur contexte et leurs configurations adéquates).

5.1 Modélisation de variabilité avec le modèle de caractéristiques

FeatureIDE est un outil pour FOSD (feature-oriented software development). C'est un plug-in Eclipse pour le développement des logiciels. Il prend en charge toutes les phases de FOSD pour le développement des produits SPL: analyse du domaine, mise en œuvre du domaine, analyse des besoins, et dérivation des produits. Différentes techniques de mise en œuvre de SPL sont intégrées, comme la programmation orientée fonction(FOP), la programmation orientée aspect (AOP)...Etc.[Thuma,2014].

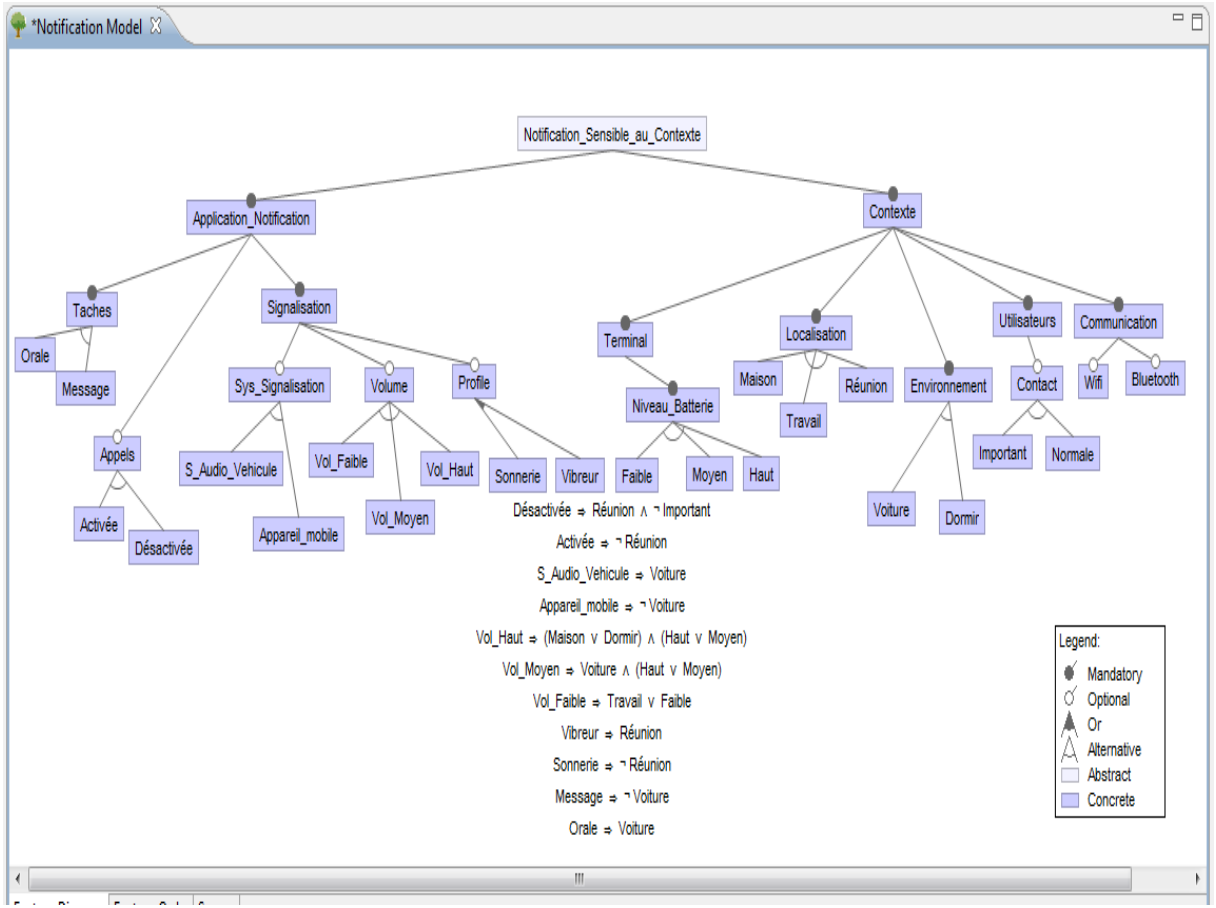

Figure 5.3 : Features Modèle d'une famille des systèmes de notification sensibles au contexte des applications mobiles.

Le modèle de features (voir Figure 5.3) a comme racine ***Notification_sensible_au_contexte.*** C'est un Point de variation *PV*. Il contient deux branches (PV Application_Notification et PV Contexte) avec un ' *And* ' parce que ces deux branches sont nécessaires pour un système sensible au contexte. La branche *contexte* contient les cinq dimensions où chaque dimension PV représente une facette du contexte (**Terminal**, **Communication, Utilisateurs**, **Localisation**, **Environnement**), exemple : le PV terminal s'intéresse au PV Niveau_Batterie de type '*Ou'* qui comme situation les vriants (Faible, Moyen ou Haut).

La branche PV *Application_Notification* de type 'And' parce que elle gère les PVs *appels*, *Tâches* et *Signalisation.* PV *Tâches* a deux variantes de type '*Xor'* (*message écrit* et *message orale)* . c'est-à-dire, les tâches sont affichées sous forme d'un message ou orale. PV *Appels* contient deux variantes de type '*Xor'*. L'un des deux variantes (*Activée* ou *Désactivée*) va être sélectionnés. Le PV *Sys_Signalisation* a deux vriantes de type 'Xor' (*S_Audio_Véhicule et Appareil_Mobile*). Le PV Volume a trois Varaintes (*Vol_Faible*, *Vol_Moyen*et*Vol_Haut*). Le *Profile* contient deux variantes (*Sonnerie*et*Vibreur).*

5.2 Représentation de l'information contextuelle

Selon la branche Contexte (voir figure 5.3), les caractéristiques terminales, représentent une succession de situations, chaque sous ensemble des situations, représente les états d'une entité observée (ou ressources). Le tableau 5.1 montre la représentation d'un contexte selon un vecteur de valeurs binaires. Chaque ligne du tableau, montre une information du contexte. L'ensemble des lignes représente toutes les situations possibles du contexte. Une situation est représentée par '1', à sont état actif et '-1' à sont état passif. Le nombre des situations possibles du contexte est égal au produit du nombre des situations de chaque entité observé.

Exemple : Pour notre contexte formé comme suit :

- *Niveau_Batteire* : nombre d'alternatives =3,
- *Localisation* : nombre d'alternatives =3,
- *Environnement* : nombre d'alternatives =2,
- *Contact* : nombre d'alternatives =2,
- *Communication* : nombre d'alternatives =2.

Nous aurons3 X 3 X 2 X 2 X 2 = 72 configurations du contexte possibles, illustrées dans la table 5.1.

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Contexte	Faible	Moyen	Haut	Maison	Travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
1	1	-1	-1	1	-1	-1	1	-1	1	-1	1	-1
2	1	-1	-1	-1	1	-1	1	-1	1	-1	1	-1
3	1	-1	-1	-1	-1	1	1	-1	1	-1	1	-1
-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-
72	-1	-1	1	-1	-1	1	-1	1	-1	1	-1	1

Tableau 5.1 : représentation des différentes informations du contexte. -

5.3 Représentation des contraintes sous forme d'un tableau.

Le FM de la Figure 5.3, contient onze (11) contraintes (les règles d'adaptations). Chaque contrainte doit être vérifiée pour sélectionner une caractéristique dynamique. Nous allons présenter toutes ces contraintes, sous forme d'un tableau de dépendances (voir tableau 5.13).

Exemple de contraintes :

La caractéristique 'Activée' nécessite (*implies*) pour toutes les situations, autres que 'Réunion' pour sa sélection.

1) Feature (Activée) $\overset{implies}{\leftarrow}$ (¬ Réunion).

Activée = true (ou 1) si Réunion n'est pas activée (=-1)

Elle est représentée avec une seule ligne dont toutes les colonnes (situations) non besoin =0, sauf la colonne (Réunion) =-1.

0 représente les colonnes (situations) qui n'ont pas aucun impact sur la sélection d'une caractéristique.

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Activée	0	0	0	0	0	-1	0	0	0	0	0	0

Tableau 5.2 : Matrice de dépendances de contrainte Activée.

2)- Feature (Vol_Haut) ← (Maison ***or*** Dormir) ***and*** (Haut ***or*** Moyen).

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Vol_Haut	0	0	1	1	0	0	0	0	0	0	0	0
Vol_Haut	0	0	1	0	0	0	0	1	0	0	0	0
Vol_Haut	0	1	0	1	0	0	0	0	0	0	0	0
Vol_Haut	0	1	0	0	0	0	0	1	0	0	0	0

Tableau 5.3 : Matrice de dépendances de contrainte Vol_Haut.

3)- Feature (Vol_Moyen) ← (Voiture) ***and*** (haut ***or*** Moyen).

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Vol_Moyen	0	0	1	0	0	0	1	0	0	0	0	0
Vol_Moyen	0	1	0	0	0	0	1	0	0	0	0	0

Tableau 5.4 : Matrice de dépendances de contrainte Vol_Moyen.

4)- Feature (Désactivée)← (Réunion) ***and*** (¬Important).

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Désactivée	0	0	0	0	0	1	0	0	-1	0	0	0

Tableau 5.5 : Matrice de dépendances de contrainte Désactivée.

5)- Feature (S_Audio_Vehicule)← (Voiture).

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
S_Audio_Vehicule	0	0	0	0	0	0	1	0	0	0	0	0

Tableau 5.6 : Matrice de dépendances de contrainte S_Audio_Vehicule

6)- Feature (Appareil_mobile)← (¬Voiture).

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Appareil_mobile	0	0	0	0	0	0	-1	0	0	0	0	0

Tableau 5.7 : Matrice de dépendances de contrainte Appareil_mobile.

7)- Feature (Vol_Faible)← (Travail) or (Faible)

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Vol_Faible	0	0	0	0	1	0	0	0	0	0	0	0
Vol_Faible	1	0	0	0	0	0	0	0	0	0	0	0

Tableau 5.8 : Matrice de dépendances de contrainte Vol_Faible.

8)- Feature (Vibreur)← (Réunion)

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Vibreur	0	0	0	0	0	1	0	0	0	0	0	0

Tableau 5.9 : Matrice de dépendances de contrainte Vibreur.

9)- Feature (Sonnerie)← (¬Réunion)

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Sonnerie	0	0	0	0	0	-1	0	0	0	0	0	0

Tableau 5.10 : Matrice de dépendances de contrainte Sonnerie.

10)- Feature (Message)← (¬Voiture)

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Message	0	0	0	0	0	0	-1	0	0	0	0	0

Tableau 5.11 : Matrice de dépendances de contrainte Message.

11)- Feature (Orale)← (Voiture)

	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Orale	0	0	0	0	0	0	1	0	0	0	0	0

Tableau 5.12 : Matrice de dépendances de contrainte Orale.

	Contexte											
	Niveau_Batterie			localisation			Environnement		Contact		communication	
Features	Faible	Moyen	Haut	Maison	travail	Réunion	Voiture	Dormir	Important	Normale	Wifi	BT
Activée	0	0	0	0	0	-1	0	0	0	0	0	0
Vol_Haut	0	0	1	1	0	0	0	0	0	0	0	0
Vol_Haut	0	0	1	0	0	0	0	1	0	0	0	0
Vol_Haut	0	1	0	1	0	0	0	0	0	0	0	0
Vol_Haut	0	1	0	0	0	0	0	1	0	0	0	0
Vol_Moyen	0	0	1	0	0	0	1	0	0	0	0	0
Vol_Moyen	0	1	0	0	0	0	1	0	0	0	0	0
Désactivée	0	0	0	0	0	1	0	0	-1	0	0	0
S_Audio_Vehicule	0	0	0	0	0	0	1	0	0	0	0	0
Appareil_mobile	0	0	0	0	0	0	-1	0	0	0	0	0
Vol_Faible	0	0	0	0	1	0	0	0	0	0	0	0
Vol_Faible	1	0	0	0	0	0	0	0	0	0	0	0
Vibreur	0	0	0	0	0	1	0	0	0	0	0	0
Sonnerie	0	0	0	0	0	-1	0	0	0	0	0	0
Message	0	0	0	0	0	0	-1	0	0	0	0	0
Orale	0	0	0	0	0	0	1	0	0	0	0	0

Tableau 5.13 : Matrice de dépendances de toutes les contraintes définis dans FM.

5.4 Exécution de l'algorithme génétique

Le résultat de l'exécution de l'algorithme génétique pour toutes les situations possibles du contexte, varie selon le nombre de génération et le nombre de population. Le tableau 5.14 montre l'optimalité (le pourcentage des configurations optimales), des configurations générées selon le nombre de génération, le nombre de population initiale et le temps d'exécution.

Configuration optimale	génération	population	Temps en ms	
			minimum	Maximum
40 %	10	20	>1	14
77 %	100	20	< 1	77
91 %	200	20	< 1	133
99 %	300	20	< 1	182
99-100 %	400	20	< 1	275
100 %	500	20	<1	280
98-100 %	500	10	<1	191

Tableau 5.14 : Résultat de l'exécution d'un GA selon plusieurs critères.

Le GA et le RN sont implémentés en Java sous Eclipse3.5.2 et Java Virtual Machine (JVM) 1.6. SE : Widows 7, CPU : I3 2.5 GHz.

A partir du *tableau* 5.14, nous remarquons que le taux de réussite de la sélection des caractéristiques valides, dépend du nombre de générations et du nombre d'individus de la population initiale. Dans notre cas, pour garantir le taux le plus élevé, nous avons choisi 500 comme nombre de générations et 20 comme nombre de solutions initiales.

Dans le *tableau* 5.14 nous avons spécifié un intervalle du temps consommé (colonne 4) pour générer une configuration. La variation du temps de génération d'une configuration est liée à la génération de la population initiale.

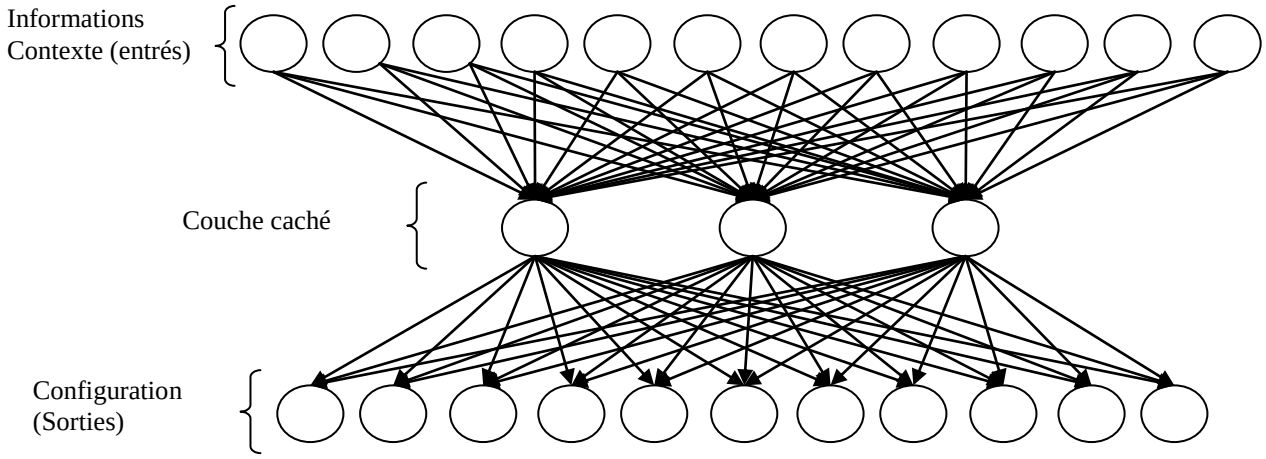

Figure 5.4 : Schéma du réseau de neurone CA-SPL.

5.5 Apprentissage du réseau de neurones artificiels

Pour exécuter l'algorithme d'apprentissage pour notre exemple (*notification*), nous avons besoin de définir le nombre de neurones pour chaque couche (entrée, sortie et caché). La première couche concerne les informations du contexte. Le nombre de neurones dépend alors des situations identifiées pour représenter les informations de l'état du contexte (12 situations). La dernière couche concerne les informations de sélection des features, le nombre de neurones dépend des features dynamiques identifiées pour représenter une reconfiguration (11 features dynamiques). Nous avons défini une seule couche cachée qui est représentée par trois(03) neurones. La figure 5.4 montre le schéma du réseau de neurones.

Le réseau de neurones doit pouvoir calculer les valeurs de sorties (Configuration) en fonction des valeurs d'entrées (contexte). Il contient plusieurs couches de neurones (série) interconnectées. Le résultat de chaque série sera une entrée pour la seconde, sauf la dernière qui représente le résultat final (configuration) qui constitue la sortie du réseau.

L'apprentissage neuronal fait appel à des exemples de comportement. Les variables modifiées pendant l'apprentissage, sont les poids des connexions. L'apprentissage supervisé, consiste à adapter les poids du réseau dans l'optique d'accorder la réponse du réseau aux exemples. Il est souvent impossible de décider, à priori, des valeurs des poids des connexions d'un réseau pour une application donnée.

Dans notre cas d'étude, on a identifié 72 situations différentes du contexte. Ce qui implique 72 exemples d'information du contexte. Chaque situation a sa propre configuration optimale, générée par le GA. Alors, chaque exemple est un couple d'entrée/sortie (exemple *i* = {contexte *Xi*, Configuration *Yi*}).

6. Phase d'exécution : L'utilisation du Réseau de Neurones

Cette phase permet de conserver au réseau un comportement adapté malgré les fluctuations dans les données d'entrées, après l'exécution ou l'utilisation du RN, pour la classification des différentes situations du contexte et leurs configurations. Les résultats obtenus pour toutes les situations du contexte, sont idéales aux exemples fournit. Le temps consommé pour générer une reconfiguration de chaque situation, dépend du nombre de variabilités du contexte et de l'application.

Evaluation :

Dans le Tableau 5.15, nous donnons les résultats d'exécution de CA-SPL (réseau de neurones) ainsi que GAFES : GA Approach to optimized FEature selection in SPLs [Guo, 2011])

Alors que, nous avons programmé l'approche CA-SPL et relevé le temps de l'exécution sur un appareil mobile, A partir du Tableau 5.15, nous remarquons que CA-SPL consomme un temps d'exécution minime pour les différents nombre de caractéristiques définies dans FM.

Nombre de caractéristiques	GAFES ms	CA-SPL
10	107	2
50	412	39
100	1025	98
200	3435	367

Tableau 5.15 : Temps de calcul pour une expérimentation d'un petit modèle de caractéristiques.

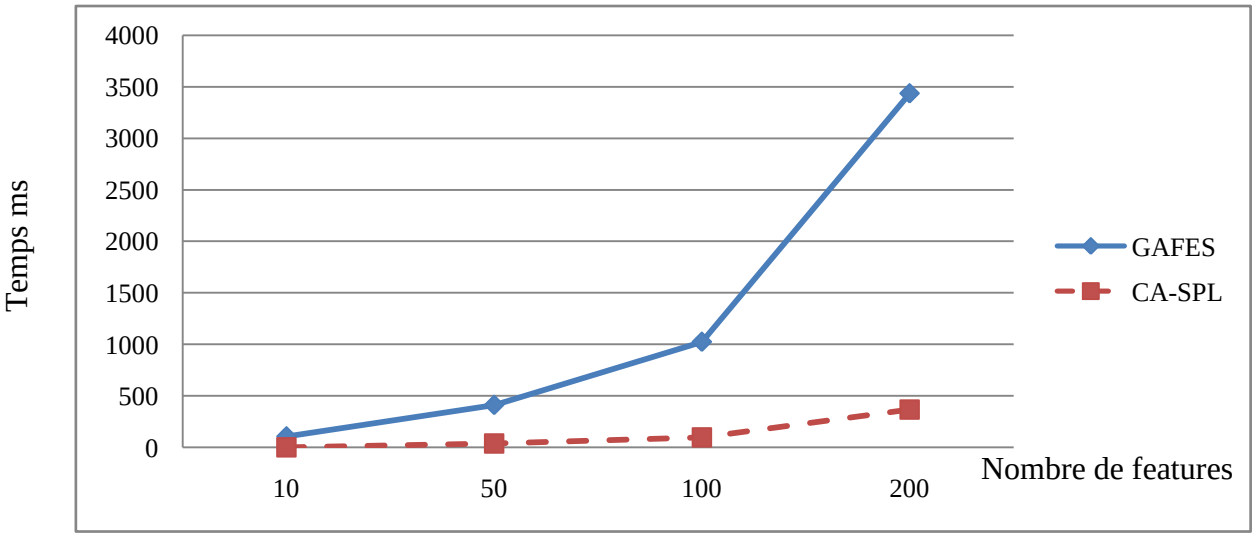

Figure5.5 :Temps d'exécution en millisecondes pour les différentes tailles du modèle de caractéristiques avec GAFES(AG) et CA-SPL(RN).

Conclusion

L'utilisation de l'algorithme génétique au moment de l'exécution afin de générer une configuration optimale, consomme beaucoup de temps de calcule, particulièrement sur les appareils mobiles. Dans notre approche, nous avons exécuté le GA sur un micro-ordinateur avec un nombre de génération et de population plus important, afin de générer des reconfigurations optimales. Cette expérience (exemples du Contexte, configuration) est affectée à un RN qui s'exécute sur des appareils mobiles pour générer des configurations optimales aux situations détectées.

L'adaptation dans CA-SPL, consiste à remplacer la configuration actuelle d'une application par une nouvelle choisie. Le choix d'une configuration est obtenu en utilisant les informations du contexte capturées durant l'exécution. Afin de réduire le temps de décision (la sélection d'une configuration) au moment de l'exécution, nous avons utilisé un RN pour générer une configuration (sortie RN) selon les informations du contexte capturées (entrés RN). La qualité des résultats produits par le RN, dépend des exemples (contexte, configuration), fournis à l'algorithme d'apprentissage, qui est exécuté au moment de la conception pour mettre à jour les poids du RN, qui seront réutilisés par le même RN au moment de l'exécution.

Le nombre des exemples d'apprentissage est lié au nombre des situations du contexte identifiées (phase de conception) et qui ont un impact sur l'application (reconfiguration dynamique). Afin de permettre la couverture de toutes les situations possibles du contexte, et générer une reconfiguration optimale pour chaque situation, au moment de la conception, nous avons proposé une approche qui comporte deux processus qui s'exécutent au moment de la conception :

1- Un processus de génération des informations contextuelles est utilisé pour couvrir toutes les situations du contexte.
2- Un processus de génération de configurations optimales selon le principe de l'informatique autonome (Autonomic Computing [IBM, 2006]) qui comporte quatre étapes : Monitor, Analyse, Plan et Execute.

Monitor : les informations du contexte sont générées par un processus de génération des informations contextuelles (processus 1).

Analyse : L'analyse du contexte n'a pas d'importance parce que les informations du contexte générées ne sont pas redondantes.

Plan : La génération d'un plan de reconfiguration d'une façon automatique dans CA-SPL se fait au moment de la conception. Nous utilisons le même GA que celui de *GAFES* [Guo, 2011] en modifiant la Fonction d'évaluation. La fonction d'évaluation analyse une reconfiguration avec les contraintes de sélection des caractéristiques définies dans FM du SPL (*connaissance*), et calcule l'erreur de sélection ou la désélection des caractéristiques dans une reconfiguration. Au contraire de [García, 2013], la variabilité du contexte est aussi représentée dans FM avec une branche séparée nommée *contexte*. Dans notre approche, on a utilisé les contraintes de sélection *Cross-tree constraints* (SPL) pour définir les règles d'adaptations (stratégie A dans [Capilla, 2014]), au contraire de [Guo, 2011] qui a utilisé des caractéristiques attributs (*feature-attributes*) afin de représenter les ressources nécessaires pour chaque caractéristique (stratégie B [Capilla, 2014]).

Execute : L'adaptation des poids de RN, consiste à exécuter l'algorithme d'apprentissage d'RN, avec des exemples générés (information contextuelle comme entrée de RN, configuration générée comme sortie de RN).

Dans la phase d'exécution, on a utilisé le même DRS [García, 2013] mais qui utilise le RN avec les poids adapté (au moment de la conception) pour la génération d'un plan de reconfiguration.

Conclusion Générale

Sommaire

Conclusion :

La reconfiguration des logiciels au moment de l'exécution présente un nouveau défi pour adapter les produits selon les exigences des utilisateurs et les conditions dynamiques de l'environnement dont lesquelles une application peut être utilisée. Comme conséquence, un processus de développement automatisé est nécessaire pour le développement des systèmes sensibles au contexte. Les stratégies proposées par la communauté SPL sont ambitieuses pour la création des applications configurables et qui peuvent être adaptées à la variété de l'ensemble des besoins. SPL est construit autour d'un ensemble de composants logiciels avec les points de variabilité qui permettent la personnalisation. Cependant, les implémentations de telles stratégies sont encore rares et ne sont pas étendues convenablement aux systèmes logiciels qui ont besoin d'être adaptés continuellement après leurs implémentations. Pour générer une configuration optimisée et qui satisfait un ensemble arbitraire de besoins, les développeurs se trouvent face à un nombre de défis. Même avec un petit modèle de caractéristiques, une combinaison de caractéristiques peut produire un nombre exponentiel de configurations. Différentes recherches dans la communiée SPL ont appliqué et présenté différentes techniques pour résoudre automatiquement le problème de sélection des caractéristiques durant la conception ou au moment de l'exécution. Quoique les techniques d'optimisation présentées fournissent des configurations optimales, le temps consommé demeure encore non raisonnable pour les applications mobiles.

L'objectif de ce travail était d'étudier les solutions existantes et leurs insuffisances pour le problème de la sensibilité au contexte. De plus, il s'agit de mettre en évidence la nécessité des solutions DSPL pour la reconfiguration automatique afin de couvrir dynamiquement de nouveaux besoins et de réagir aux changements des conditions de l'environnement.

Dans ce mémoire, nous avons présenté une nouvelle approche qui fournit un support pour la reconfiguration automatique et dynamique des applications mobiles en réponse aux changements des situations du contexte. La génération d'une reconfiguration valide d'un système est gouvernée par les contraintes définies dans un modèle de caractéristiques. Celles-ci incluent les caractéristiques du contexte et celles des systèmes. Elles sont séparées dans le modèle en deux branches. De cette façon, nous pouvons profiter des outils et des algorithmes qui sont disponibles pour FM afin d'analyser une configuration. Concrètement, l'utilisation d'un algorithme génétique combiné avec FM afin de permettre de générer une configuration optimisée au moment de l'exécution, était déjà proposée. Cependant, l'utilisation d'un AG au moment de l'exécution diminue les performances de l'appareil mobile (le temps consommé, et l'espace mémoire occupé).

Le travail qui m'a été confié consiste à proposer une approche SPL pour la reconfiguration automatique et dynamique des applications mobiles qui permettra d'améliorer la performance (temps d'exécution et l'espace mémoire occupé) du service de reconfiguration. L'approche utilisée est basée sur l'utilisation combinée des réseaux de neurones multicouches et les algorithmes génétiques pour générer un plan de reconfiguration.

1 Contributions

1.1 Représentation binaire de contexte :

Avec les mécanismes SPL de gestion de la variabilité [Hartmann, 2008] utilisés pour modéliser les situations du contexte qui peuvent être activées ou désactivées durant l'exécution. Nous avons représenté une information contextuelle comme une succession des valeurs binaires (voir section 4.4.1.1) dont chaque nombre binaire représente l'État d'une situation (activée='1', non activée='-1') identifiée dans le FM du contexte au moment de l'exécution.

1.2 Représentation Matriciel des contraintes de FM

Chaque contrainte (gestion de la variabilité et de dépendance) liée à une caractéristique dynamique est représentée dans une matrice avec une ou plusieurs lignes dont chaque ligne présente les valeurs (colonnes) nécessaires pour sa vérification (voir section 4.4.1.2.b.).

1.3 Génération des exemples d'apprentissage

Le processus d'ingénierie d'application exécuté au moment de la conception (chapitre 4, section 3.1), consiste à générer des exemples d'apprentissages pour le réseau de neurones. Un exemple est un couple (contexte, reconfiguration). Le processus de génération des informations contextuelles est implémenté pour couvrir toutes les situations du contexte définies dans FM. La configuration adéquate est générée par un AG d'optimisation de la sélection des caractéristiques définies par [Guo, 2011].

L'utilisation des valeurs binaires pour représenter les exemples d'apprentissages (contexte, configuration) minimise le temps d'exécution de l'algorithme d'apprentissage et aussi son utilisation pour la classification (au moment de l'exécution), avec la réduction du nombre de couches cachées nécessaires pour la bonne classification, et aussi le nombre de neurones dans chaque couche caché. Dans notre cas, avec comme entré de RN l'information de situation contexte qui est représenté avec 12 valeurs binaires et la configuration (sortie de RN) 11 valeurs binaires on a identifié une seule couche cachée avec 3 neurones.

2 Les leçons tirées

Dans ce mémoire, nous avons présenté une nouvelle approche qui fournit un support pour la reconfiguration dynamique des applications mobiles. Afin d'atteindre cet objectif, nous avons modélisé, en plus de la variabilité de l'architecture des applications mobiles, la variabilité du contexte avec FM de SPL (stratégie A de [Capilla, 2014]). Nous avons utilisé le FM comme connaissances [Guo, 2011, García, 2013] parce qu'il constitue l'élément principal pour la conception de la décision. Par exemple, quand et comment lier (bind) les différentes variantes [Abbas, 2011]. Nous avons utilisé un algorithme génétique pour la génération des configurations au moment de la conception pour permettre de :

(1) couvrir toutes les situations possibles du futur contexte,

(2) générer une configuration optimale (correcte) adéquate pour chaque situation du contexte déterminée, avec un nombre important de générations et de population appliquée par l'algorithme génétique afin de parcourir un segment large de l'espace de solution,

(3) générer une configuration optimale avec un AG sur des appareils mobiles, consomme un temps important de calcul et exige un espace mémoire qui provoque le disfonctionnement des autres services exécutés sur le même appareil.

L'intérêt de l'utilisation d'un réseau de neurones au moment de l'exécution réside dans la minimisation du temps d'exécution pour générer des configurations. Les résultats de l'exécution du RN sont corrects parce que les poids sont adaptés en utilisant un algorithme d'apprentissage qui couvre toutes les situations possibles du contexte et sa configuration adéquate générée par GA afin d'adapter le mieux possible les poids du RN.

Le temps de calcul consommé par l'exécution du RN pour générer une reconfiguration appropriée à un contexte dépend aussi du nombre de variabilités définies dans l'architecture de l'application et des variabilités du contexte (nombre de neurones définis dans chaque couche).

L'AG utilisé évalue une configuration selon les contraintes définies dans le FM (gestion des variabilités et des contraintes de dépendances).La matrice de dépendance utilisée pour représenter les contraintes est générée manuellement. On peut utiliser une autre méthode pour évaluer la sélection d'une caractéristique dynamique, est d'exécuter une contrainte avec les variables du contexte à condition que la contrainte définisse les dépendances entre les variabilités de l'application et du contexte qui sont impliqué dans sa sélection.

Les informations provenant des capteurs (matériels ou logiques) sont nombreuses. Un gestionnaire du contexte traduit ces informations en informations utiles ayant un sens significatif pour l'application afin de réduire l'espace des situations à analyser.

On peut conclure aussi que les approches SPLs traditionnelles (statiques) dérivent des produits qui s'exécutent dans un environnement statique. Les approches DSPLs dérivent des produits dont l'environnement est dynamique et assurent la reconfiguration de l'architecture

de l'application d'une façon dynamique (durant l'exécution). Les approches ASPLs sont similaires aux DSPLs avec une gestion d'un nombre important de variabilités structurelles selon le principe de l'informatique autonome [IBM, 2006].

3 Perspectives du travail

Dans notre approche, nous n'avons pas dérivé les contraintes directement à partir du FM. Les contraintes sont ajoutées manuellement dans la fonction fitness. L'algorithme génétique utilise les contraintes du contexte pour générer une sélection de caractéristiques sans gérer la variabilité définie dans FM.

Afin d'appliquer le plan de reconfiguration au moment de l'exécution. On utilise un composant nommé ***Configurateur*** [Trinidad, 2007]. Le ***Configurateur***, est un composant qui a comme objectif de créer des liens (définissant les interfaces) entre les composants au moment de l'exécution pour modifier l'architecture de l'application qui repend au plan de reconfiguration.

Pour la génération des informations contextuelles conformes au modèle de l'environnement. On peut utiliser « The fix operator (Gustavo. Pascual, 2015)» pour la génération et la validation des scenarios contextuels.

Dans notre approche, nous n'avons pas dérivé les contraintes directement à partir du FM. Les contraintes sont ajoutées manuellement dans la fonction fitness. L'algorithme génétique utilise les contraintes du contexte pour générer une sélection de caractéristiques sans gérer la variabilité définie dans FM.

Afin d'appliquer le plan de reconfiguration au moment de l'exécution. On utilise un composant nommé ***Configurateur*** [Trinidad, 2007]. Le ***Configurateur***, est un composant qui a comme objectif de créer des liens (définissant les interfaces) entre les composants au moment de l'exécution pour modifier l'architecture de l'application qui repend au plan de reconfiguration.

Pour la génération des informations contextuelles conformes au modèle de l'environnement. On peut utiliser « The fix operator (Gustavo. Pascual, 2015)» pour la génération et la validation des scenarios contextuels.

Bibliographie

Bibliographie:

[Abbas, 2011] Abbas, N., Andersson, J., & Weyns, D. (2011, August). Knowledge evolution in autonomic software product lines. In *Proceedings of the 15th International Software Product Line Conference, Volume 2* (p. 36). ACM.

[Abowd, 1997] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyber Guide : A mobile context-aware tour guide. Wireless Networks, 3(5) :421–433, October 1997.

[Akbar, 2001] Akbar, M.M., Manning, E.G., Shoja, G.C., Khan, S., 2001. Heuristic solutions for the multiple-choice multi-dimension knapsack problem. In: Proceedings of ICCS'01, San Francisco, CA, USA, pp. 659–668.

[Akman, 1997] V. Akman andM. Surav. The Use of Situation Theory in ContextModeling. Computational Intelligence, 13(3) :427–438, 1997.

[Alsuwaiyel, 1999] Alsuwaiyel, M.H., 1999. Algorithms: Design Techniques and Analysis. World Scientific.

[Anastassopoulos, 2005] Anastasopoulos, M. (2005). Software Product Lines for Pervasive Computing.*IESE-Report No. 044.04/E version, 1*.

[Andersson, 2000] Andersson, J. (2000). A deployment system for pervasive computing. In *Software Maintenance, 2000. Proceedings. International Conference on* (pp. 262-270). IEEE.

[Arboleda, 2009] Arboleda, H., Romero, A., Casallas, R., & Royer, J. C. (2009, April). Product Derivation in a Model-Driven Software Product Line using Decision Models. In *CIbSE* (pp. 59-72).

[Arboleda, 2009] Arboleda, H., Romero, A., Casallas, R., & Royer, J. C. (2009, April). Product Derivation in a Model-Driven Software Product Line using Decision Models. In *CIbSE* (pp. 59-72).

[Arboleda, 2009] Hugo Arboleda, Andrés Romero, Rubby Casallas, and Jean-Claude Royer. Product derivation in a model-driven software product line using decision models. In Antonio Brogi, João Araújo, and Raquel Anaya, editors, CIbSE, pages 59–72, 2009.

[Asthana, 1994] A. Asthana, M. Cravatts, and P. Krzyzanowski. An indoor wireless system for personalized shopping assistance. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 69–74, Santa Cruz, California, December 1994.

[Atkinson, 2000] Atkinson, C., Bayer, J., Muthig, D.: Component-Based Product Line Development: The KobrA Approach. In: Proc. of 1st SPLC, Norwell, MA, USA (2000) 289-309.

[Baldauf, 2007] Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, *2*(4), 263-277.

[Benavides, 2005] Benavides, D., Martin-Arroyo, P.T., Cortes, A.R., 2005. Automated reasoning on feature models. In: Proceedings of CAiSE'05, Porto, Portugal, pp. 491–503.

[Benavides, 2008] Benavides, D., Ruiz-Cortes, A., Batory, D., Heymans, P., 2008. First international workshop on analysis of software product lines (ASPL'08). In: Proceedings of SPLC'08,Limerick, Ireland, p. 385.

[Bencomo, 2008] Bencomo, N., Sawyer, P., Blair, G. S., & Grace, P. (2008, September). Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *SPLC (2)* (pp. 23-32).

[Bencomo, 2008] Bencomo, N., Sawyer, P., Blair, G. S., & Grace, P. (2008, September). Dynamically Adaptive

Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *SPLC (2)* (pp. 23-32).

[Blair, 2004] Blair, G. S., Coulson, G., & Grace, P. (2004, October). Research directions in reflective middleware: the Lancaster experience. In *Proceedings of the 3rd workshop on Adaptive and reflective middleware* (pp. 262-267). ACM.

[Bolchini, 2007] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A data-oriented survey of context models. SIGMOD Rec., 36(4) :19–26, 2007.

[Bosch, 2000] Bosch, J. (2000). *Design and use of software architectures: adopting and evolving a product-line approach*. Pearson Education.

[Bradley, 2005] Bradley, N. A., & Dunlop, M. D. (2005). Toward a multidisciplinary model of context to support context-aware computing. *Human-Computer Interaction*, *20*(4), 403-446.

[Brézillon, 2002] Brézillon, P. (2002). Hors du contexte, point de salut. *Séminaire" Objets Communicants*.

[Brice, 2008] Morin, B., Barais, O., & Jézéquel, J. M. (2008). K@ rt: An aspect-oriented and model-oriented framework for dynamic software product lines. In *Proceedings of the 3rd International Workshop on Models@ Runtime, at MoDELS'08*.

[Brown, 1997] Brown, P. J., Bovey, J. D., & Chen, X. (1997). Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*, *4*(5), 58-64.

[Bunt, 1994] Bunt, H. (1994). Context and dialogue control. *Think Quarterly*, *3*(1), 19-31.

[C. national] C. national de la recherche scientifique. *Dictionnaire de la langue franÃ§aise du XIXeme et XXeme siècle*. Centre national de la recherche scientifique.

[C.Taconet, 2011] Chantal Taconet (2011). Intergiciels pour la sensibilit´ e au contexte en environnement ubiquitaire. Soutenance le jeudi 10 février 2011, Télécom SudParis, Laboratoire Samovar, UMR 5157

[Capilla, 2014] Capilla, R., Ortiz, O., & Hinchey, M. (2014). Context Variability for Context-Aware Systems. *Computer*, *47*(2), 85-87.

[Cetina-a, 2008] Cetina, C., Pelechano, V., Trinidad, P., & Cortés, A. R. (2008, September). An Architectural Discussion on DSPL. In *SPLC (2)* (pp. 59-68).

[Cetina-b, 2008] Cetina, C., Fons, J., & Pelechano, V. (2008, September). Applying software product lines to build autonomic pervasive systems. In *Software Product Line Conference, 2008. SPLC'08. 12th International* (pp. 117-126). IEEE.

[Chaari, 2007] Chaari, T. (2007). *Adaptation d'applications pervasives dans des environnements multi-contextes* (Doctoral dissertation, institut national des sciences appliquées de Lyon).

[Chen, 2004] Chen, H An intelligent Broker Architecture for Pervasive Context-Aware Systeme. PhD thesis. Baltimore County: department of CSEE, Univeristy of Maryland, 2004.

[Cheng, 2009] Cheng, B. H., De Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., ... & Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems* (pp. 1-26). Springer Berlin Heidelberg.

[Clements, 2002] Clements, P., & Northrop, L. (2002). *Software product lines: practices and patterns* (Vol. 59). Reading: Addison-Wesley.

[Coutaz, 2002] Coutaz J., Rey G., "Recovering foundations for a theory of contextors", 4th ICCADUI, 2002, Valenciennes, France.

[Cuervo, 2010] Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010, June). MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (pp. 49-62). ACM.

[Czarnecki-a, 2007] Czarnecki, K., & Wasowski, A. (2007, Sepctember). Feature diagrams and logics: There and back again. In *Software Product Line Conference, 2007. SPLC 2007. 11th International* (pp. 23-34). IEEE.

[Czarnecki-b, 2000] Czarnecki, K., & Eisenecker, U. W. (2000). Generative programming.

[Czarnecki-c, 2005] Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: GPCE. LNCS 3676 (2005) 422.437.

[Daniel, 2007] Daniel A. Menascé and Jeffrey O. Kephart. Guest editors' introduction: Autonomic computing. IEEE Internet Computing, 11(1):18–21, 2007. 8, 50

[Deelstra, 2004] Deelstra, S., Sinnema, M., & Bosch, J. (2004). Experiences in software product families: Problems and issues during product derivation. In *Software Product Lines* (pp. 165-182). Springer Berlin Heidelberg.

[Deelstra, 2005] Deelstra, S., Sinnema, M., & Bosch, J. (2005). Product derivation in software product families: a case study. *Journal of Systems and Software*, *74*(2), 173-194.

[Dey, 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (1999, January). Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing* (pp. 304-307). Springer Berlin Heidelberg.

[Dey-a, 2001] [Dey, A..] Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, *5*(1), 4-7.

[Dey-b, 2001] A Dey, A. K., Abowd, G. D., & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, *16*(2), 97-166.

[Dey-c, 2000] DEY, A. K. Providing Architectural Support for Building Context-Aware Applications . PhD thesis, Georgia Institute of Technology, 2000.

[Dinkelaker, 2010] [Dinkelaker et al] Dinkelaker, T., Mitschke, R., Fetzer, K., & Mezini, M. (2010, March). A dynamic software product line approach using aspect models at runtime. In *Fifth Domain-Specific Aspect Languages Workshop* (Vol. 39, p. 40).

[Englada, 2011] Cetina Englada, C. (2011). Applying Software Product Lines to Build Autonomic Pervasive Systems.

[Fernandes, 2008] Fernandes, P., Werner, C., & Murta, L. G. P. (2008, July). Feature Modeling for Context-Aware Software Product Lines. In *SEKE* (pp. 758-763).

[Francisco, 2011] Daniel Francisco, Context as a Resource: A Service-Oriented Approach for Context-Awareness, 2011.

[Frank, 2007] van der Linden, F. J., Schmid, K., & Rommes, E. (2007). *Software product lines in action*. Springer-Verlag Berlin Heidelberg.

[Gamez, 2011] Gamez, N., Fuentes, L., & Aragüez, M. A. (2011). Autonomic computing driven by feature models and architecture in famiware. In *Software Architecture* (pp. 164-179). Springer Berlin Heidelberg.

[García, 2013] García Pascual, G., Fuentes Fernández, L., & Pinto, M. (2013). Run-time Support to Manage

Architectural Variability Speci ed with CVL.

[Garlan, 2002] Garlan, D., Siewiorek, D. P., Smailagic, A., & Steenkiste, P. (2002). Project aura: Toward distraction-free pervasive computing. *Pervasive Computing, IEEE*, *1*(2), 22-31.

[Garlan, 2004] Garlan, D., Cheng, S. W., Huang, A. C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, *37*(10), 46-54.

[Gomaa, 2006] Gomaa, H. (2006). Designing software product lines with uml 2.0: From use cases to pattern-based software architectures. In *Reuse of Off-the-Shelf Components* (pp. 440-440). Springer Berlin Heidelberg.

[Gu, 2004] T. Gu, X.H. Wang, H.K. Pung, and D.Q. Zhang. An Ontology-based Context Model in Intelligent Environments. In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, volume 2004, 2004.

[Gu, 2004] Gu T., Pung H. K., and Zhang D.Q. A middleware for building context-aware mobile services. In Proceeding of IEEE vehicular Technology Conference (VTC), 2004, Milan, Italy.

[Guo, 2011] Guo, J., White, J., Wang, G., Li, J., & Wang, Y. (2011). A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, *84*(12), 2208-2221.

[Guo, 2011] Guo, J., White, J., Wang, G., Li, J., & Wang, Y. (2011). A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, *84*(12), 2208-2221.

[Hallsteinsen, 2008] Hallsteinsen, S., Hinchey, M., Park, S., & Schmid, K. (2008). Dynamic software product lines. *Computer*, *41*(4), 93-95.

[Hamza-a, 2010] [H. Hamza] Hamza, H. S., Martinez, J., & Mugartza, J. L. (2010, October). KOPLE: knowledge-oriented product line engineering. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (pp. 275-276). ACM.

[Hamza-b, 2010] [H. S. Hamza] Hamza, H. S., & Aly, G. M. (2010, October). Using product line architectures to leverage systematic reuse of business knowledge: an industrial experience. In *Proceedings of the 2010 Workshop on Knowledge-Oriented Product Line Engineering* (p. 5). ACM.

[Hartmann, 2008] Hartmann, H., & Trew, T. (2008, September). Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Software Product Line Conference, 2008. SPLC'08. 12th International* (pp. 12-21). IEEE.

[Henricksen, 2002] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In Pervasive Computing : First International Conference, Pervasive 2002, Zurich, Switzerland, August 26-28, 2002 : Proceedings. Springer, 2002.

[IBM, 2006] BM. An Architectural Blueprint for Autonomic Computing.White paper, June 2006.

[Janik, 2010] Janik, A., & Zielinski, K. (2010). AAOP-based dynamically reconfigurable monitoring system. *Information and Software Technology*, *52*(4), 380-396.

[Jean, 2009] Jean-Christophe. Les lignes de produits logiciels Réutilisation et variabilité.TECHN.35. 2009.

[Jeffrey, 2003] Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41-50.

[Kaenampornpan, 2004] M. Kaenampornpan and E. O'Neill. Modelling Context : An Activity Theory Approach. In

Ambient Intelligence : Second European Symposium, EUSAI 2004, Eindhoven, The Netherlands, November 8-11, 2004 : Proceedings. Springer, 2004.

[Kang, 1990] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-oriented domain analysis (FODA) feasibility study, Technical Report CMU/SEI-90-TR-021. Software Engineering Institute, CMU.

[Kirsch, 2005] Kirsch-Pinheiro, M., Villanova-Oliver, M., Gensel, J., & Martin, H. (2005, May). Une formalisation du contexte dans les environnements coopératifs nomades. In *Proceedings of the 2nd French-speaking conference on Mobility and ubiquity computing* (pp. 1-8). ACM.

[Klein., 2008] Cornel Klein, Reiner N. Schmid, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. A survey of context adaptation in autonomic computing. In ICAS, pages 106–111, 2008.

[Korpepaa, 2004] JANI, M., KELA, Juha, MALM, Esko-Juhani, *et al.* Managing context information in mobile devices. *IEEE pervasive computing*, 2003, vol. 2, no 3, p. 42-51.

[Lee, 2006] Lee, J., & Muthig, D. (2006). Feature-oriented variability management in product line engineering. *Communications of the ACM*, *49*(12), 55-59.

[LINDEN, 2005] VAN DER LINDEN, F., & Pohl, K. (2005). Software Product Line Engineering: Foundations, Principles, and Techniques.

[Loke, 2005] Loke, S. (2005). *Context-aware Pervasive Systems: Architectures for a New Breed of Applications*. Auerbach Publications.

[Lutz, 2009] Lutz, R., & Rouquette, N. (2009, September). Using defect reports to build requirements knowledge in product lines. In *Managing Requirements Knowledge (MARK), 2009 Second International Workshop on* (pp. 12-21). IEEE.

[Mathieu, 2009] Mathieu Acher, Philippe Collet1. Modeling Context and Dynamic Adaptations with Feature Models. 2009

[McCarthy, 1993] J. McCarthy. Notes on formalizing context. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, volume 1, pages 555–560, 1993.

[Mitchell, 1996] Mitchell, M., 1996. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA.

[Morin, 2008] Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J. M., Solberg, A., Dehlen, V., & Blair, G. (2008). An aspect-oriented and model-driven approach for managing dynamic variability. In *Model driven engineering languages and systems* (pp. 782-796). Springer Berlin Heidelberg.

[Morin, 2008] Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J., Solberg, A., Dehlen, V., Blair, G.: An Aspect-Oriented and Model-Driven approach for managing dynamic variability. In: Model Driven Engineering Languages and Systems conference. (2008)

[Núñez, 2009] Núñez, A., Noyé, J., & Gasiūnas, V. (2009, July). Declarative definition of contexts with polymorphic events. In *International Workshop on Context-Oriented Programming* (p. 2). ACM.

[Oreizy, 1999] Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., ... & Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, *14*(3), 54-62.

[Oreizy-a, 1999] Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., ... & Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, *14*(3), 54-62.

[Oreizy-b, 2008] Oreizy, P., Medvidovic, N., & Taylor, R. N. (2008, May). Runtime software adaptation: framework, approaches, and styles. In *Companion of the 30th international conference on Software engineering* (pp. 899-910). ACM.

[Parnas, 1976] Parnas : on the design and development of program families, IEEE transaction on soft Engineering, SE-2(1) :1-9, march 1976.

[Parra, 2008] Carlos Parra, Laurence Duchien. Model-Driven Adaptation of Ubiquitous Applications. Proceedings of the First International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2008)

[Parra-a, 2009] Parra, C., Blanc, X., & Duchien, L. (2009, August). Context awareness for dynamic service-oriented product lines. In *Proceedings of the 13th International Software Product Line Conference* (pp. 131-140). Carnegie Mellon University.

[Parra-b, 2011] Parra, C. (2011). *Towards dynamic software product lines: Unifying design and runtime adaptations* (Doctoral dissertation, Université des Sciences et Technologie de Lille-Lille I).

[Paspallis, 2009] [N. Paspallis,] Paspallis, N. (2009). Middleware-based development of context-aware applications with reusable components. *University of Cyprus*.

[Perrouin, 2008] Perrouin, G., Klein, J., Guelfi, N., & Jézéquel, J. M. (2008, September). Reconciling automation and flexibility in product derivation. In *Software Product Line Conference, 2008. SPLC'08. 12th International* (pp. 339-348). IEEE.

[Philip, 2004] Philip, K., Eric, P., & Betty, H. C. (2004). Composing adaptive software.

[Philip,S, 2004] Philip, S. M. Sadjadi, E. P. Kasten, B. H. C. Cheng. A Taxonomy of Compositional Adaptation. Technical report MSU-CSE-04-17, Michigan State University, Department of Computer Science and Engineering, East Lansing, Michigan, 2004.

[Pohl, 2005] Pohl, K., Bockle, G., van der Linden, F., 2005. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer-Verlag, Berlin,

[Preuveneers, 2004] D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In Ambient Intelligence : Second European Symposium, EUSAI 2004, Eindhoven, The Netherlands, November 8-11, 2004 : Proceedings. Springer, 2004.

[Rey, 2002] Rey, G., & Coutaz, J. (2002, November). Le contexteur: une abstraction logicielle pour la réalisation de systèmes interactifs sensibles au contexte. In *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)* (pp. 105-112). ACM.

[Rouvoy, 2009] Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., ... & Scholz, U. (2009). Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software engineering for self-adaptive systems* (pp. 164-182). Springer Berlin Heidelberg.

[Ryan, 1997] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced Reality Fieldwork : the Context-aware Archaeological Assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[Salber, 1998] Salber, D., Dey, A. K., & Abowd, G. D. (1998). Ubiquitous computing: Defining an hci research agenda for an emerging interaction paradigm.

[Salehie, 2009] M. Salehie and L. Tahvildari. Salehie, M., & Tahvildari, L. (2009). Self-adaptive software:

Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *4*(2), 14.

[Salim, 2006] Hariri, S., Khargharia, B., Chen, H., Yang, J., Zhang, Y., Parashar, M., & Liu, H. (2006). The autonomic computing paradigm. *Cluster Computing*, *9*(1), 5-17.

[Samulowitz, 2001] M. Samulowitz, F. Michahelles, and C. Linnhoff-Popien. CAPEUS : An Architecture for Context-Aware Selection and Execution of Services. In Proceedings of the IFIP TC6/WG6. 1 Third International Working Conference on New Developments in Distributed Applications and Interoperable Systems, pages 23–40. Kluwer, BV Deventer, The Netherlands, The Netherlands, 2001.

[Schilit, 1994] Schilit, B., Theimer, M. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5). 1994. pp 22-32.

[Sheng, 2005] Q.Z. Sheng and B. Benatallah. ContextUML : A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In Proceedings of the International Conference on Mobile Business (ICMB'05), pages 206–212, 2005.

[Stefan, 2009] S. (2009). *Front Matter* (pp. i-xxvii). John Wiley & Sons, Ltd.

[Strang-a, 2004] T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. In Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp, pages 34–41, 2004.

[Strang-b, 2003] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL : A Context Ontology Language to Enable Contextual Interoperability. In Distributed Applications and Interoperable Systems : 4th Ifip Wg6. 1 International Conference, Dais 2003, Paris, France, November 17-21, 2003, Proceedings. Springer, 2003.

[Svahnberg, 1999] Svahnberg, M., & Bosch, J. (1999). Evolution in software product lines. *Software Maintenance*, *11*(6), 391-422.

[Szyperski, 1998] C. Szyperski. Component Software: Beyond Object-Oriented Programming. Addison-Wesley Professional, December 1998.

[Taconet, 2011] Chantal Taconet. Intergiciels pour la sensibilit´ e au contexte en environnement ubiquitaire. Méemoire d'Habilitation à Diriger des Recherches en Informatique. février 2011. Université d' ´ Evry-Val-d'Essonne. Sudparis.

[Thuma, 2014] THÜM, Thomas, KÄSTNER, Christian, BENDUHN, Fabian, et al. FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming, 2014, vol. 79, p. 70-85.

[Trinidad, 2007] Trinidad, P., Cortés, A. R., Peña, J., & Benavides, D. (2007, September). Mapping Feature Models onto Component Models to Build Dynamic Software Product Lines. In *SPLC (2)* (pp. 51-56).

[Voelter, 2007] Voelter, M., Groher, I.: Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In: Proc. of the 11th SPLC. (2007) 233-242.

[W.Viana, 2010] Viana de Carvalho, W. (2010). *Mobilité et sensibilité au contexte pour la gestion de documents multimédias personnels: CoMMediA* (Doctoral dissertation, Université de Grenoble).

[Weiser, 1990] Weiser, M. (1991). The computer for the 21st century. Scientific American, 265(3), 66–75.

[Weiss, 1999] Weiss, David M., Lai, Chi Tau Robert: Software Product-Line Engineering: A Family-Based Software Development Process, Addision- Wesley, 1999.

[White, 2009] White, J., Dougherty, B., Schmidt, D.C., 2009. Selecting highly optimal architectural feature sets with filtered cartesian flattening. Journal of Systems and Software82 (8), 1268–1284.

[Ziadi, 2003] Tewfik ZIADI, Loïc HELOUET, Jean-Marc JEZEQUEL. Modélisation de Lignes de Produits en UML. LMO 2003 – Langages et Modèles à Objets

[Ziadi, 2005] Ziadi, T., & Jézéquel, J. M. (2005). Manipulation de lignes de produits logiciels: Une approche dirigée par les modèles. Sébastien Gérard, Jean-Marie Favre, Pierre-Alain Muller, et Xavier Blanc, éditeurs, 1ere journées sur l'Ingénierie Dirigée par les Modèles-IDM, 5, 26.

Résumé

Les applications mobiles s'exécutent dans des environnements où le contexte change d'une façon continue. Donc, il est nécessaire de fournir un support d'auto-adaptation dynamique, afin de permettre aux systèmes d'adapter leurs comportements selon le contexte de l'exécution. Ce support est actuellement accompli par des plateformes d'intergiciels, qui offrent un service de reconfiguration dynamique sensible au contexte. Cependant, le défaut principal d'approches existantes soit qu'elles ne sont pas convenables pour les appareils mobiles ou elles utilisent un algorithme génétique (AG) au moment de l'exécution pour générer un plan de reconfiguration. L'exécution d'un AG sur un appareil mobile provoque le disfonctionnement de certains services avec des configurations produites presque justes (86-90%). Dans ce mémoire, on présente une approche combinée des réseaux de neurones (RN) multicouches et des algorithmes génétiques, afin de permettre la génération automatique d'un plan de reconfigurations au moment de l'exécution, sans diminuer les performances de l'appareil. L'utilisation de RN lors de l'exécution afin de produire un plan de reconfiguration est achevé par : (1) avoir les informations sur la variabilité du contexte et d'applications, (2) l'utilisation d'AG pour générer des configurations optimales pour chaque situation de contexte, dont le but de produire des exemples d'apprentissages, (3) l'exécution de l'algorithme d'apprentissages de RN au moment de la conception afin d'adapter ses poids à l'ensemble d'exemples générés, et finalement (4) le RN est utilisé au moment d'exécution dans le but de générer un plan de reconfigurations dans un temps optimal. Nous avons spécifié un cas d'étude et l'évaluation de notre approche, les résultats montrent que l'exécution de notre approche est efficace.

Mots clés : Reconfiguration Dynamique, Contexte, Lignes de produits logiciels, Algorithme Génétique, Réseau de neurones, informatique autonome.

ملخص

التطبيقات المحمولة تنفذ في بيئات متنوعة أين السياق يتغير بصفة متواصلة، ما يلزم توفير ركيزة للتأقلم الآلي و ديناميكي للبرامج. هذه الركائز هي حاليا موجودة باستعمال قاعدة ما بين البرامج التي توفر خدمة إعادة البرمجة الديناميكية الحساسة للسياق. لكن العيب في هذه القواعد هو استعمال تقنيات غير مناسبة للاجهزة المحمولة أو استعمال الخورزميات الجينية (AG) في وقت التنفيذ لإنتاج تشكيلة محسنة لبرنامج. استعمال الخورزميات الجينية على الأجهزة المحمولة تأثر بصفة سلبية على اشتغال التطبيقات الأخرى و أيضا التشكيلات المنتجة هي بالكاد جيدة. في هذه المذكرة نقدم مقاربة لإنتاج تشكيلات لبرامج بصفة آلية و ديناميكية بدون تخفيض قدرات الأجهزة و التطبيقات الأخرى، هذه المقاربة تقوم باستعمال تركيبة من الشبكة العصبية الاصطناعية (RN) المتعددة الطبقات و الخورزميات الجينية. استعمال الشبكة العصبية الاصطناعية إبان التنفيذ لإنتاج تشكيلات ملائمة لمجموعة الحالات الخاصة بالمحيط تجسد باستعمال (1) توفير المعلومات الخاصة بتنوع السياق و البرنامج المعرفة بنموذج الخصائص، (2) استعمال الخورزمية الجينية لاستخراج التشكيلات المحسنة لكل حالة السياق لإنتاج أمثلة التمرن، (3) تنفيذ خورزمية التمرن الخاصة بالشبكة العصبية الاصطناعية في آونة الهندسة لأقلمة الأوزان لمجموعة الأمثلة المستخرجة، و أخيرا (4) استعمال الشبكة العصبية الاصطناعية مع الأوزان المتأقلمة لإنتاج تشكيلات لكل سياق مكتشف. لقد قمنا بتخصيص دراسة حالة وتقييم مقاربتنا، النتائج تظهر ان تنفيذها على أجهزة محمولة أنها جيدة.

كلمات مفتاحية: Reconfiguration Dynamique, Contexte, Lignes de produits logiciels, Algorithme Génétique, Réseau de neurones, informatique autonome.