الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

جامعة باجي مختار- عنابة

Année : 2016 / 2017

Faculté des Sciences de l'Ingéniorat
Département d'Informatique

THESE

Présentée en vue de l'obtention du diplôme de
DOCTORAT EN SCIENCES

ADAPTABILITE DE LOGICIELS DANS L'EMBARQUE

Filière: Informatique
Option: Informatique Industrielle

Par:
Abdelghani ALIDRA

Devant le Jury:

Président :	Pr BOUDOUR Rachid	Univ. Badji Mokhtar- Annaba
Rapporteur:	Pr KIMOUR M. Tahar	Univ. Badji Mokhtar- Annaba
Examinateurs:	Pr CHAOUI Allaoua	Univ. Constantine II.
	Pr REDJIMI Mohamed	Univ.20 Aout 55- Skikda

Thèse réalisée au sein du: Laboratoire des Systèmes Embarqués (LASE)

Résumé

Les nouvelles problématiques introduites par l'omniprésence du logiciel dans différentes branches de l'industrie afin de faire face à la complexité croissante des systèmes et à l'incertitude de leur environnement, ont poussé les ingénieurs logiciels à se tourner vers l'auto-adaptation. Les systèmes auto-adaptatifs sont capables de faire face à un environnement en constante évolution et à des exigences émergentes qui peuvent être inconnues au moment de la conception. La construction de tels systèmes de façon rentable et de manière prévisible est un défi majeur de l'ingénierie des systèmes logiciels. Dans cette perspective, le paradigme des Lignes de Produits Logiciels Dynamiques (LPLD) qui se base sur la notion de caractéristique a apporté un apport considérable.

Dans cette thèse, nous avons étudié les différentes approches les plus notables pour l'adaptabilité des systèmes embarqués et nous avons mis en relief leurs limites dans le contexte de la prise de décision. Particulièrement, nous avons mis l'accent sur le fait que ces travaux implémentent la prise de décision sur de simples règles Evénement-Condition-Action. Ce type de règles ne permettent pas de prendre en charge des scenarios omis ou non prévus au moment de la conception, ou avec des stratégies orientées objectifs qui reportent la planification de l'adaptation au moment de l'exécution en associant des valeurs de fitness aux configurations cibles et en sélectionnant la meilleure d'entre elles.

Ces approches proposent des outils supports ayant une complexité calculatoire très importante (en termes de consommation de ressources et de temps d'exécution). Les ressources et le temps sont deux facteurs critiques dans le cas de l'adaptabilité à l'exécution. Dans ce sens, les approches existantes ne sont pas appropriées pour les systèmes embarqués modernes.

Cette thèse appréhende le défi de la prise de décision dans les systèmes logiciels embarqués adaptables comme étant un problème de premier plan en ne prenant en considération que le processus de prise de décision qui doit vérifier un certain nombre de contraintes liées aux paramètres du temps et des ressources disponibles.

Dans cet ordre d'idée, nous avons défini la notion de dépendance transitive entre les caractéristiques, sur laquelle une nouvelle approche pour le raisonnement sur les modèles de caractéristiques a été proposée dans cette thèse.

Afin de supporter la prise de décision, nous avons défini une variante améliorée de l'algorithme génétique pour la sélection de caractéristique, capable d'implémenter le problème particulièrement délicat de l'auto-guérison dans les systèmes embarqués temps réel. En outre, et afin d'analyser les modèles de caractéristiques nous avons implémenté un outil support qui présente l'avantage d'être plus efficace et performant tant sur le plan qualité que sur le plan du temps de calcul.

***Mots Clés**: Ligne de produits Logiciels Dynamiques, Adaptabilité du logiciel, Modèles de caractéristiques, Algorithmes génétiques, Systèmes embarqués, MAPE-K, informatique autonome, sensibilité au context, planification de l'adaptation.*

ملخص

التواجد الدائم للأنظمة المعلوماتية في مختلف القطاعات الصناعية أدخلت إشكاليات جديدة متعلقة بالتعقيد المتزايد لهذه الأنظمة و اللبس المرتبط بمحيطها مما أدى بمهندسي البرامج إلى الاستعانة بتقنيات التأقلم الذاتي. الأنظمة المتأقلمة ذاتيا هي الأنظمة القادرة على مواجهة محيط في تغير دائم و متطلبات قد تكون مبهمة في مرحلة إنشاء النظام. إن تطوير هذا النوع من الأنظمة بطريقة فعالة يشكل تحديا مهما لهندسة الأنظمة البرمجية. من هذا المنظار، تعتبر تقنية سلسلة المنتجات البرمجية المتغيرة التي تتمحور حول فكرة الخاصية البرمجية تقنية ذات فائدة معتبرة.

في هذه الأطروحة، سنتطرق لدراسة أهم تقنيات إنجاز الأنظمة المدمجة المتأقلمة ذاتيا مع التركيز على جانب اتخاذ القرار. تحديدا، وجدنا أن أغلبية الأعمال الموجودة تستعمل قواعد بسيطة على شكل فعل-شرط-ردة فعل، لإتحاد القرارات المتعلقة بالتأقلم الذاتي. هذا النوع من القواعد لا تسمح بالتعامل الصحيح مع السيناريوهات التي لم يتم توقعها في مرحلة إنشاء. عائلة أخرى من هذه التقنيات تستعمل إستراتجيات موجهة الأهداف و التي تأخر إتحاد القرار إلى مرحلة التشغيل و ذلك من خلال ربط قيم تعكس جودة النظام ثم اختيار هيئة النظام التي لديها الجودة القصوى. هذه التقنيات تعاني من استهلاك مفرط لموارد النظام خاصة فيما يخص موارد الطاقة و الوقت. تعتبر هذه الموارد عوامل مهمة في حالة التأقلم الذاتي. من هذا المنظور فإن التقنيات الموجودة غير مناسبة في إطار الأنظمة المدمجة الحديثة.

هذه الأطروحة تتعامل مع التحدي إتحاد القرار في الأنظمة البرمجية المدمجة و المتأقلمة ذاتيا كإشكال من الدرجة الأولى. في هذا الإطار، قمنا بطرح فكرة و استغلال هذه الفكرة لمعالجة نموذج الخصائص في إطار عملية اتخاذ القرار. بصفة خاصة ، طورنا لإنجاز اتخاذ القرار خوارزميا وراثيا محسنا يستعمل التعلق المتعدي بين الخصائص البرمجية لإيجاد هيئة النظام القريبة من القصوى. قمنا أيضا بتعديل هذا الخوارزمي للتعامل مع إشكالية التعافي الذاتي. أخيرا، قمنا بإنجاز برنامج لتجربة مختلف الأفكار المقدمة في هذه الأطروحة و الذي يبين أن مقترحاتنا تمثل تطورا من ناحية الجودة و استهلاك الموارد.

كلمات مفتاح : الأنظمة المتأقلمة ذاتيا، التعافي الذاتي ،الأنظمة البرمجية المدمجة، التعلق المتعدي بين الخصائص البرمجية، إتحاد القرار، سلسلة المنتجات البرمجية المتغيرة.

Abstract

Because modern embedded systems are increasingly complexes while evolving in uncertain environments, system designers more and more adopt self-adaptation. Self-adaptive systems are those systems who can deal with ever-changing environments and emerging requirements that may be unpredicted at design time. Designing such systems in a cost-effective manner is a major challenge. From this perspective, the Dynamic Software Product Lines (DSPLs) paradigm, which is based on the feature concept, has brought a substantial benefit.

This thesis has studied various approaches in the literature to tackle the adaptability problem and has highlighted their drawbacks with regard to the decision-making challenge. It has underlined that the most notable works have implemented decision-making with simple event-condition-action (ECA) rules or with objective-oriented strategies that postpone planning to runtime by associating fitness to target configurations and selecting the best one.

Such approaches still are not handling scenarios that were not predicted at design time, and tools supporting such approaches are very computationally expensive (resources and time consuming). Resources and time are two critical factors in the case for runtime adaptability. In this sense, they do not fit modern systems as the set of possible environmental changes as well as possible actions is hardly predictable. Thus, they are not well suited to modern, complex embedded systems.

To cope with the above-mentioned problems, this thesis addresses the decision-making challenge in adaptable embedded software systems as a first class problem, while taking into account that for this kind of systems, the decision-making process must fulfill a certain number of conditions related to the constraints of resources and time.

To this end, we introduce the novel notion of transitive dependency between features. Exploiting this new concept, we propose a new reasoning approach on feature models, allowing us to handle the decision making process. We have defined a variant of genetic algorithm to address the particularly hard problem of self-healing in real-time embedded systems. Furthermore, to perform automatic analysis of the feature model we have developed a tool support that offers the advantage of being more computationally efficient and less time consuming,

---

***Keywords**: Dynamic Software Product Line, Software adaptability, Feature models, genetic algorithms, embedded systems, MAPE-K, autonomic computing, context awareness, adaptation planning.*

«Tout ce que je sais, c'est que je ne sais rien»
Socrate

Dédicaces

À Toute ma famille,
À Tout mes amis,

Abdelghani…

Remerciements

Mes premiers remerciements vont naturellement à professeur Mohammed Tahar Kimour mon directeur de thèse qui, par sa direction, m'a permis de réaliser ce travail et en même temps de découvrir l'antichambre du milieu de la recherche. Je tiens également à le remercier pour tout ce qu'il a bien voulu me procurer comme soutien, aussi bien sur le plan scientifique que sur le plan moral. Qu'il sache que sa gentillesse, son entière disponibilité, ses précieux conseils et encouragements m'ont profondément marqué et m'ont permis de continuer à mener l'ensemble de ce travail à bien, y compris durant les inévitables périodes de doute. Sa grande rigueur scientifique et encadrement de qualité exceptionnelle ainsi que son perpétuel souci quant à mon avenir professionnel ont grandement contribué à l'accomplissement de cette page de ma vie. Pour tout cela, et pour tout le reste, je tiens à le remercier le plus chaleureusement possible.

Je tiens à remercier également et chaleureusement Monsieur Rachid Boudour, Professeur à l'Université Badji Mokhtar d'Annaba qui m'a fait le grand plaisir de présider mon jury de soutenance ainsi que monsieur Chaoui Allaoua, Professeur à l'université de Constantine et monsieur Mohammed Redjimi, Professeur à l'université 20 Aout 55 de Skikda qui m'ont fait le grand bonheur de bien vouloir examiner cette thèse et apporter leurs jugements sur ses contributions. Je veux également les remercier pour leur disponibilité et leur immense gentillesse.

Ma plus grande gratitude va également à Dr Stéphane Ducasse ainsi qu'a tout les membres de l'équipe Rmod de l'institut Inria Lille-Nord Europe pour m'avoir invité dans son équipe et pour son soutien et ses encouragements.

Je tiens aussi à remercier du plus profond de mon cœur et de mon âme tous les membres de ma famille.

N'ayant pu dresser une liste exhaustive de toutes les personnes ayant contribué de près ou de loin à ce travail, Je remercie toutes celles et ceux qui m'ont aidé à le réaliser.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Introduction Générale

La complexité des systèmes embarqués actuels et l'incertitude liée à leur environnement d'exécution ont incité la communauté du génie logiciel à chercher des solutions dans les domaines connexes tel que la robotique et l'intelligence artificielle, afin de trouver de nouvelles manières de concevoir et de gérer ces systèmes (Abdeson et al. 2000) (Diao et al. 2005) (Di Marzo-Sergendo et al. 2005) (Brun et al. 2007). Aussi, la capacité des systèmes embarqués à ajuster leurs comportements aux fluctuations de leurs environnements sous la forme d'un processus d'auto-adaptation est devenue une exigence majeure particulièrement pour les systèmes mobiles et omniprésents. Le préfixe auto (ou «self» en anglais) indique que les systèmes décident de manière autonome (c.-à-d., Sans ou avec très peu d'intervention humaine) comment s'adapter ou s'organiser pour tenir compte des changements dans leurs contextes et dans leurs environnements.

Par ailleurs, le logiciel est devenu la pierre angulaire de nombreux systèmes embarqués complexes (c'est-à-dire, des systèmes composés de parties interconnectées qui dans leurs ensemble présentent une ou plusieurs propriétés qui ne sont pas prévisibles à partir des propriétés des parties prises à part). La complexité de ces systèmes est devenue tel, que l'effort qu'ils requièrent à l'utilisateur humain pour les faire fonctionner et les maintenir opérationnels est intolérable. Ainsi, les caractéristiques de ces systèmes complexes doivent être l'auto-adaptation, l'auto-guérison et l'émergence (Ottino, 2004). Les systèmes embarqués doivent pouvoir s'adapter plus facilement à leur environnement en constante évolution. Ils doivent être flexibles, tolérants aux pannes, robustes, résistants, disponibles, configurables, sécurisés et auto-guérissables. Idéalement, et nécessairement pour les systèmes de grandes tailles, l'adaptation doit se produire de façon autonome. La communauté scientifique qui s'est formée autour des systèmes auto-adaptatifs a déjà produit des résultats encourageants, faisant ainsi des systèmes auto-adaptables un domaine de recherche important, interdisciplinaire et actif.

Les systèmes auto-adaptatifs ont été étudiés dans les différents domaines de recherche de l'ingénierie du logiciel, y compris l'ingénierie des exigences (Brown et al. 2006), l'architecture logicielle (Richter et al. 2006) (Garlan et al, 2003), les middlewares (Liu et al, 2006), et le développement basé sur les composants (Peper et al, 2008). Cependant, la plupart

de ces initiatives sont isolées. D'autres communautés scientifiques ont également étudié l'auto-adaptation à partir de leurs propres perspectives: la théorie du contrôle, le contrôle de l'ingénierie, l'intelligence artificielle, les robots mobiles et autonomes, les systèmes multi-agents, les systèmes tolérants aux pannes, l'informatique fiable, les systèmes distribués, les systèmes autonomes, les communications autonomes, les interfaces utilisateurs adaptables, l'intelligence artificielle distribuée, l'apprentissage automatique, les réseaux de capteurs et l'informatique omniprésente. Au cours de la dernière décennie, plusieurs domaines d'application et technologies ont vue le jour. Il est important de souligner que dans toutes ces initiatives, le logiciel a été l'élément commun qui permet l'auto-adaptation. Il est donc impératif de considérer les approches systématiques d'ingénierie logicielle pour développer des systèmes auto-adaptables.

Construire des systèmes embarqués auto-adaptables de manière rentable et prévisible est un défi d'ingénierie majeur. En effet, Les ingénieurs en général et plus particulièrement les ingénieurs logiciels conçoivent les systèmes selon des exigences et des spécifications concrètes et ne sont pas habitués à exprimer les exigences et les propriétés émergentes (Ottino, 2004). Afin que les techniques de l'ingénierie logicielle embarquée évoluent correctement, les ingénieurs logiciels doivent innover dans leurs approches à la construction, l'exécution et la gestion des systèmes logiciels.

L'informatique autonome a émergé comme une approche prometteuse pour le développement des systèmes adaptable (Horn, 2001) L'informatique autonome envisage des systèmes qui s'exécutent dans des environnements en constante évolution sans nécessiter l'intervention d'agents humains. Un système avec des capacités autonome doit être capable d'installer, configurer, ajuster et maintenir ces propres composants en cours d'exécution. Le terme autonome est emprunté à la biologie. Dans le corps humain, le système nerveux autonome est responsable des reflexes inconscients, c.-à-d., les fonctions qui ne requièrent pas notre attention telle que la dilatation de la pupille ou la fonction respiratoire. Sans le système nerveux autonome, nous serions constamment occupés à adapter notre organisme à ses besoins et à son environnement (Cetina, 2009).

Inspirée par la biologie, l'informatique autonome s'intéresse à la création de systèmes et d'applications qui s'auto gèrent afin de décharger leurs utilisateurs (ainsi que, dans une certaine mesure, leurs concepteurs) de la complexité de les maintenir en fonctionnement de manière efficace. A cet effet, le système doit être capable de surveiller son environnement afin de détecter les événements qui nécessitent une phase d'adaptation. En fonction de ces observations, le système doit établir un plan d'adaptation qui sera ensuite exécuté pour le

ramener dans un état de fonctionnement optimal.

De nombreux travaux ont été menés pour étudier les systèmes autonomes notamment les phases de monitoring et de reconfiguration à l'exécution. Toutefois, la prise de décision a souvent été sous-étudiée et n'a fait l'objet d'une étude dédiée qu'a de rares occasions (Khan et al, 2007, Almeida et al, 2014, Pascual et al, 2015a, Pascual et al, 2015b). En réalité, cet aspect du problème de l'adaptabilité a souvent été traité dans le cadre globale de l'adaptabilité et n'a donc pas bénéficier d'une attention particulière. Par conséquence les mécanismes mis en œuvre pour la prise de décision dans les systèmes adaptables sont souvent rudimentaires et insuffisants. Ils constituent souvent le maillon faible des approches existantes (Bencomo et al, 2010, Brataas et al, 2007). En effet, les politiques de prise de décision basées sur les scénarios et les règles d'adaptation explicitent ont montré leurs limites face au défis de complexité des systèmes et des changements fréquent de l'environnement résultant des phénomènes de mobilité et d'omniprésence. Ceci est encore plus vrai dans le cas des systèmes logiciels embarqués à cause des contraintes supplémentaires liées au temps de réponse et à la consommation de ressources.

Les défis de la prise de décision dans les systèmes embarqués adaptables

L'analyse des approches existantes pour l'adaptabilité dans les systèmes informatisés en général et les systèmes embarqués en particulier nous a permit de diagnostiquer des insuffisances au niveau des mécanismes de prise de décision. Afin de clarifier l'exposé et de proposer une solution efficace, nous avons identifié 6 points qui représentent des mesures de qualité pour les solutions existantes et future au problème abordé par notre travail

1. **Prise de décision pertinente** : La première qualité d'un processus de prise de décision est la pertinence, c'est-à-dire que le plan d'adaptation doit amener le système en cours d'exécution dans un état qui répond au mieux aux attentes de l'utilisateur. Idéalement, cette amélioration peut être quantifiée grâce par exemple à une ou plusieurs fonctions d'utilité.
2. **Différer la prise de décision** : l'établissement du plan d'adaptation doit être différé à l'étape d'exécution et non pas fixé à la conception. Par exemple, Bencomo et al ont montré qu'un plan d'adaptation sera d'autant plus efficace que son établissement sera différé dans le temps (Bencomo et al, 2010). En effet, le plan sera plus pertinent, plus flexible et plus robuste face aux scenarios imprévisibles.

3. **Vérifier la validité du plan d'adaptation** : Le plan d'adaptation ne doit pas amener le système dans un état instable ou inconsistant. L'état cible doit vérifier ces conditions de stabilité et de consistance avant même que la reconfiguration du système ne soit opérée.
4. **Facilité de maintenance, de réutilisation et d'évolution de la stratégie d'adaptation** : La logique d'adaptation est une partie critique qui détermine dans une large mesure la correction et/ou les performances du système. Par conséquence, cette logique doit être assez claire et modulaire pour permettre l'opération de maintenance. Souvent, comme lorsqu'une entreprise développe des produits assez proches, le code de cette logique doit pouvoir être réutilisé avec un minimum d'effort. Enfin, l'évolution de la logique suite à l'évolution du produit lui-même, voir suite à l'évolution du système en cour d'exécution doit être également considérée.
5. **Impact mineur sur la consommation des ressources et le temps d'exécution** : En effet, deux limitations majeures relatives aux applications logicielles destinées aux systèmes embarqués sont les temps de réponse aux changements de l'environnent et une faible consommation des ressources du système (notamment la puissance du calcul et la consommation d'énergie)
6. **Temps de réponse garanti pour l'auto-guérison**: L'auto-guérison désigne la capacité d'un système informatisé à recouvrir (ou au moins à mitiger) la perte ou le dysfonctionnement de l'un de ces composants afin de maintenir un service acceptable (Garlan et al. 2002, Jiang et al. 2007). cette caractéristique est très importante dans plusieurs systèmes embarqués qui ont souvent un caractère critique. En cas de panne, le système d'adaptation doit être en mesure non seulement de trouver une configuration cible adéquate, à même de maintenir une bonne qualité de service, mais également d'établir très rapidement un plan d'adaptation et de prendre en compte dans l'établissement de ce plan, le temps nécessaire à son exécution.

En conclusion, bien que les systèmes auto-adaptables soient devenus largement populaires et constituent une réponse prometteuse à la crise de complexité des systèmes logiciels modernes, les techniques pour leur mise en œuvre restent insuffisantes notamment du point de vue de la prise de décision et plus particulièrement dans le cas des systèmes embarqués. Nous soutenons que la recherche relative aux points susmentionnés peut contribuer à pousser aussi bien les chercheurs que les ingénieurs de ces systèmes vers une

conception efficace et adaptée de la prise de décision dans les logiciels embarqués adaptables.

Les contributions

Nous proposons dans le présent travail de traiter le problème de la prise de décision comme une question de premier plan à cause de son impact majeur sur le processus d'adaptation en ligne. Plus particulièrement, cette thèse vise à proposer une approche à la prise de décision qui réponde aux défis identifiés plus haut. Ainsi, notre contribution peut se résumer dans les points suivants:

- Notre première contribution consiste en la définition du nouveau concept de variabilité liée. En découlent les relations de dépendances transitives entre caractéristiques que nous exploitons pour implémenter efficacement l'ensemble des algorithmes de raisonnement sur la logique d'adaptation avec un impact mineur sur la charge computationnelle du système, la consommation d'énergie et le temps de réponse (préoccupation cinq). Plus particulièrement, les relations de dépendances transitives sont exploitées par un algorithme génétique pour réaliser la prise de décision (préoccupation quatre). Nous montrons à travers une étude comparative l'efficacité de l'approche proposée par rapport à une approche majeure de la littérature.
- Comme seconde contribution, nous définissons le problème de la prise de décision comme un problème d'optimisation opérationnel dans un espace de recherche décrit par le model de variabilité du système traité. Cette nouvelle vision du problème de prise de décision nous permet de le traiter en utilisant les techniques aujourd'hui matures et efficaces de méta heuristiques et autre techniques d'intelligence artificielle. Cela permet à son tour de répondre efficacement aux préoccupations une et deux susmentionnées. La réutilisation du modèle de variabilité nous offre un cadre pour la vérification de la validité de la configuration cible et répondre ainsi à la préoccupation trois.
- Comme dernière contribution, nous proposons une adaptation de l'algorithme évolutionnaire proposé dans la contribution 2 au problème de l'auto-guérison dans les systèmes embarqués. Afin de garantir la préoccupation de temps de réponse (préoccupation six) nous couplons l'algorithme évolutionnaire de recherche avec une machine d'états finie afin d'anticiper les pertes de composants en prévoyant l'ensemble des réponses proches de l'optimal.

Méthodologie de recherche

Afin d'effectuer le travail de cette thèse, nous allons conduire un projet de recherche

suivant la méthodologie de conception relative à la recherche dans les systèmes d'information comme décrit par (Vaishnavi et al. 2004). La méthodologie de recherche implique l'analyse de l'utilisation et de la performance des artefacts conçus afin de comprendre, expliquer et, très souvent, améliorer le comportement des aspects des systèmes d'information (Vaishnavi et al. 2004).

Le cycle de conception se compose de 5 étapes : (1) identification du problème, (2) proposition d'une solution, (3) son développement, (4) son évaluation et (5) la conclusion. Le cycle de conception est un processus itératif; La connaissance produite dans le processus par la construction et l'évaluation de nouveaux artefacts est utilisée comme entrée pour une meilleure connaissance du problème.

D'après le cycle défini dans la méthodologie de recherche de conception, nous avons commencé par l'identification du problème (voir Figure A): nous avons convenu du problème à résoudre et nous l'avons clairement exprimé.

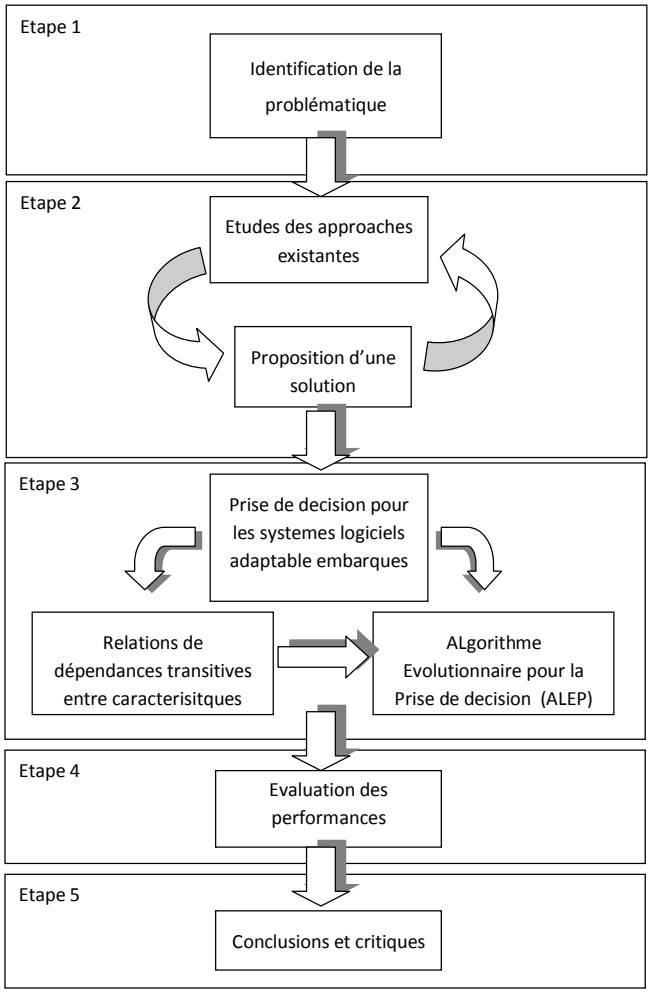


Figure A. La méthodologie de recherche adoptée

Ensuite, nous avons effectué la deuxième étape qui consiste à suggérer une solution au problème et à comparer les améliorations apportées par cette solution aux solutions déjà existantes. Pour ce faire, les approches les plus pertinentes ont été étudiées en détail. Une fois la solution au problème décrite, nous l'avons développé et validé (étapes 3 et 4). Ces deux étapes s'effectueront en plusieurs phases (voir Figure A).

Enfin, nous avons analysé les résultats de nos travaux de recherche afin d'en tirer plusieurs conclusions ainsi que d'identifier les perspectives de nos travaux futures (étape 5).

Plan de lecture

Le reste de cette thèse est structuré comme suit :

Dans la première partie, différentes notions relatives aux systèmes logiciels embarqués et à l'informatique autonome sont présentées. Plus particulièrement, le chapitre 1 expose, avec plus ou moins de détails, les systèmes embarqués dans leur globalité. Des différentes définitions existantes seront dégagés les singularités et les contraintes liées à ce type de système. Des illustrations diverses seront fournies en mettant l'accent sur la problématique étudiée qui est l'adaptabilité à l'exécution.

Dans le chapitre 2, l'approche de conception basée sur les lignes de produits logiciels est présentée. Les lignes de produits logiciels regroupent les techniques permettant une meilleure structuration du système afin d'en améliorer la réutilisabilité et la maintenance. Nous verrons que cette nouvelle approche est particulièrement intéressante dans le cas des systèmes embarqués. Nous baserons nous même notre démarche pour la prise de décision sur cette approche.

Le chapitre 3 introduit le domaine de l'informatique autonome. L'accent sera porté sur l'adaptation par les lignes de produits dynamiques qui sert de fil conducteur à notre travail. Une critique objective des travaux existant clôturera le chapitre.

Dans la deuxième partie de cette thèse, nous présenterons notre propre contribution. Dans le chapitre 4 nous introduirons le concept de variabilité liée et les relations de dépendance transitive qui en découlent. Les différentes relations seront formulées, la question, centrale de leur représentation sera abordée et leur exploitation via des operateurs de sélection transitive sera également exposée. Différents algorithmes de calcul de ces dépendances ainsi que leur exploitation pour des raisonnements sur les modèles de caractéristiques seront enfin rapportés.

Dans le chapitre 5 nous poserons le problème de l'adaptabilité comme un problème d'optimisation combinatoire sur un espace de recherche défini par le modèle de

caractéristiques du système. Se faisant, nous ouvrons la voix à l'exploitation des techniques de l'intelligence artificielle pour une planification intelligente, simple et efficace. Nous exploitons par la suite cette idée ainsi que les relations de dépendances décrites dans le chapitre 4 pour proposer un algorithme évolutionnaire pour la prise de décision.

Le chapitre 6 adresse la problématique de l'auto-guérison dans les systèmes embarqués. Une adaptation de l'algorithme évolutionnaire présenté chapitre 5 est donnée. La combinaison de l'algorithme évolutionnaire et d'une machine d'états finie permet d'anticiper les pannes de composants du système et de sélectionner un plan d'adaptation dans un temps borné.

Enfin nous concluons notre travail dans le chapitre 7. Les limites de l'approche et les perspectives futures y seront également discutées.

. Notes bibliographiques

Les travaux présentés dans cette thèse utilisent et étendent des articles et des publications des auteurs. Une liste non exhaustive en ai donnée ci-après :

1. A. Alidra, MT Kimour, "*Adapting large context-aware and pervasive systems. A new evolutionary-based approach*", International Journal of Knowledge-based and Intelligent Engineering Systems - Volume 21, issue 2. ISSN 1327-2314. Mars 2017.
2. A. Alidra, M.T. Kimour, *"Prototyping Software Product Lines analysis with Pharo"*, in **IWST'16, Prague,** Czech Republic. August 23 - 24, 2016, ACM 2016, DOI: http://dx.doi.org/10.1145/2991041.2991053
3. A Alidra, MT Kimour. «*Un nouvel opérateur de sélection de Fonctionnalités pour l'adaptation à l'exécution des systèmes logiciels* ». In the 27th International Conference on Software & Systems Engineering and their Applications. **ICSSEA Paris**, 25 - 27 Mai 2016
4. A. Alidra, MT Kimour, "*Towards a software factory for genetic algorithms*". Accepté à **ICCTS Dubai**. UEA. 2013. Publié par l' international Journal of Computer and Electrical Engineering (**IJCEE**, ISSN: 1793-8163). Février 2014.
5. A Alidra, MT Kimour. « *A new evolutionary approach to decision-making in autonomic systems* ». In Proceedings of the **3th IEEE control International Conference on System and control, Algiers** 2013
6. A Alidra, MT Kimour. « *Vers une prise de décision optimisée pour les systèmes embarqués auto-réparant*», international conference on embedded systems in telecommunications and instrumentation **(ICESTI'14) Annaba**, novembre 2014.
7. A Alidra, MT Kimour. « *Dealing with dynamic variability in embedded systems*», international conference on embedded systems in telecommunications and instrumentation **(ICESTI'12) Annaba**, Novembre 2012.
8. A. Alidra, MT Kimour. « **Automatic derivation of self-healing behavior from variability models at runtime** ». In the 24th International Conference on Software & Systems Engineering and their Applications. **ICSSEA, Paris**, 20 - 23 Octobre 2012

Première Partie

État de l'Art

Chapitre I

Les Systèmes Embarqués et l'adaptabilité logicielle

1.1 Introduction

Les systèmes embarqués ont été incontestablement une tendance de fond au cours de la dernière décennie. Pourtant, les systèmes embarqués existent depuis bien plus longtemps. Il suffit d'observer notre environnement quotidien pour s'apercevoir que les systèmes embarqués sont partout: les téléphones cellulaires, les réveils électroniques, les assistants de données personnelles (PDA), les sous-systèmes automobiles tels que l'ABS et le régulateur de vitesse, Etc. Cette section donne un aperçu des systèmes embarqués, de leurs spécificités par rapport à d'autres systèmes logiciels, des tendances actuelles en matière de comportement autonome et adaptable et de la façon dont ils peuvent bénéficier de la recherche effectuée dans cette thèse.

1.2 Systèmes embarqués par rapport aux systèmes à usage général

Un système embarqué est habituellement classé comme un système qui a un ensemble de fonctions prédéfinies et spécifiques à exécuter et dans lequel les ressources sont contraintes (Stepper et al. 1999). Prenons par exemple, une montre-bracelet numérique. Il s'agit d'un système embarqué qui a plusieurs fonctions apparentées: indication du temps, peut-être plusieurs fonctions de chronomètre, et une alarme. Il a également plusieurs contraintes de ressources. Le processeur qui fait fonctionner la montre ne peut pas être très grand, sinon personne ne la porterait. La consommation d'énergie doit être minimale; Seule une petite batterie peut être contenue dans cette montre, et cette batterie devrait durer presque aussi longtemps que la montre elle-même. Et enfin, elle doit indiquer l'heure avec précision, de façon cohérente, car personne ne veut une montre qui soit inexacte. Chaque conception embarquée satisfait à son propre ensemble de fonctions et de contraintes. Selon (Stepper et al. 1999), on estime à 50 000 le nombre de nouveaux modèles embarqués par an.

Cela diffère des systèmes à usage général, tels que l'ordinateur de bureau. Le processeur qui fait fonctionner cet ordinateur est appelé un processeur "à usage général" car il a été conçu pour effectuer de nombreuses tâches différentes, par opposition à un système embarqué, qui a été construit pour effectuer quelques tâches spécifiques soit de manière optimale ou dans des conditions d'exécution très stricts. Les points suivants résument les principaux aspects propres aux systèmes et aux applications embarqués.

- **Structure et architecture fonctionnelle**

Un système embarqué est un dispositif électronique autonome qui contient du logiciel. Il est conçu pour exécuter une tache dédiée bien définie (Kadionik, 2006). Un système embarqué ne possède généralement pas des entrées/sorties standards et classiques comme un clavier ou un écran. La figure 1.1 présente une vue générale de l'organisation d'un système embarqué typique.

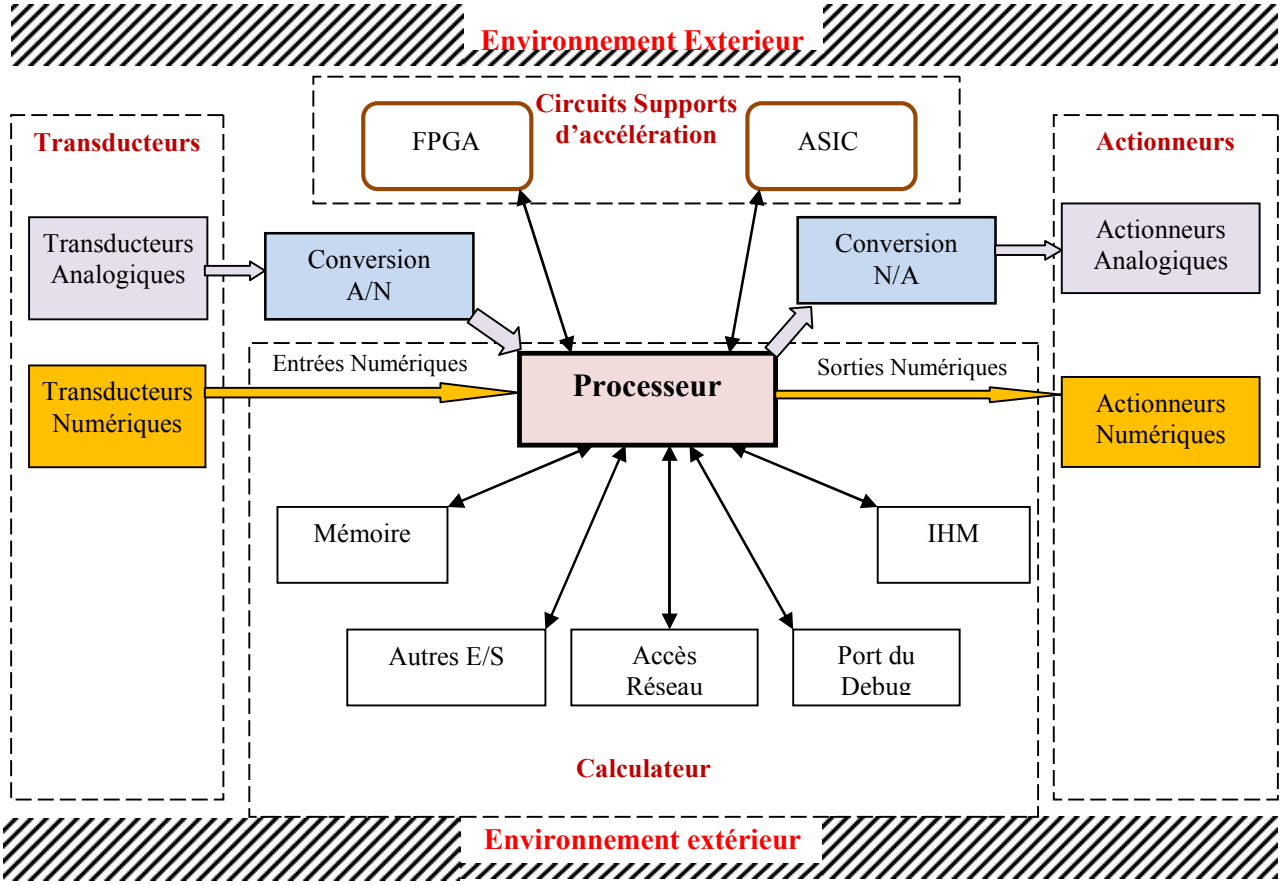


Figure 1.1. Organisation d'un Système embarqué typique

Le système matériel et l'application sont extrêmement attachés, le logiciel embarqué étant enfoui dans le matériel. Cependant, nous rencontrons maintenant des dispositifs tels que l'assistant numérique personnel (PDA) et les téléphones cellulaires portables qui sont des systèmes embarqués conçus pour être en mesure de faire une variété de fonctions primaires.

En outre, les derniers téléviseurs numériques comprennent des applications interactives qui effectuent une grande variété de fonctions générales sans rapport avec la fonction de télévision, mais tout aussi importantes, telles que le e-mailing, la navigation sur internet et les jeux.

- **Contraintes sur les ressources**

Les systèmes embarqués sont plus limités dans le matériel et/ou dans les fonctionnalités du logiciel que dans les ordinateurs personnels.

Selon (ESAPS, 2001), en termes de limitations matérielles, cela signifie des limitations dans la puissance du traitement, de la consommation d'énergie, de la mémoire et des fonctionnalités liées au matériel. Dans le logiciel, cela signifie généralement des limitations relatives c.à.d. avec moins d'applications, ou des applications à échelle réduite, ou encore avec une absence de système d'exploitation (OS) ou bien même avec un OS obtus ou avec moins de code au niveau abstraction.

- **Exigences sur les performances**

Un système embarqué est un système semblable à un système d'ordinateur mais avec des exigences de qualité et de fiabilité plus élevées (ESAPS, 2001).

- **Besoins de réactivité**

Les Systèmes embarqués sont en général réactifs. Un système embarqué temps réel est un système réactif devant fournir des sorties logiquement correctes tout en respectant strictement des contraintes temporelles explicites (Burns, 97) Il est considéré comme défaillant s'il ne respecte pas au moins l'une de ses spécifications logiques ou temporelles. Il est important de constater que la plupart des Systèmes Embarqués sont des Systèmes Temps Réel et que la plupart des systèmes Temps Réel sont embarqués.

Ces contraintes influent sur la conception et l'architecture des systèmes d'exploitation et le développement des applications des Systèmes embarqués. Etant donné que les systèmes embarqués sont dans leur grande majorité des systèmes temps réel, ils doivent respecter des contraintes temporelles fortes. C'est pour cette raison que l'on y trouve enfoui un système d'exploitation (ou noyau) Temps Réel (en anglais RTOS ou Real Time Operating System) (ESAPS, 2001). De plus, le Système embarqué utilise une famille de processeurs différente de celle du PC.

En termes de consommation d'énergie, il en consomme moins par rapport au PC car cet aspect reste primordial pour le SE. A partir de ce constat, un PC standard peut exécuter tout type d'applications car il est généraliste alors qu'un système embarqué n'exécute qu'une application dédiée et unique.

1.3 L'adaptabilité : une exigence clés pour les systèmes embarqués modernes

L'industrie des systèmes embarqués est confrontée à un grand nombre de défis, à différents niveaux de conception: de tels systèmes on exige des performances accrues tout en préservant une

consommation d'énergie aussi faible que possible. Par ailleurs, ils doivent pouvoir réutiliser le code logiciel existant et en même temps tirer parti de la logique supplémentaire disponible dans la puce, représentée par plusieurs processeurs travaillant ensemble. Afin de répondre efficacement à de tel exigences, le concept d'adaptabilité apparaît comme une solution a la fois inévitable et de choix. Dans ce qui suit, nous brossons un rapide tableau des principaux défis que les concepteurs de systèmes embarqués doivent gérer de nos jours et à l'avenir. En matière de contraintes et d'exigences, les systèmes embarqués doivent (ESAPS, 2001) :

- Satisfaire des contraintes de temps réel provenant des applications, qui peuvent être aussi bien des contraintes de temps réel souple (par exemple dans le cadre d'applications multimédia) que des contraintes de temps réel strict (par exemple dans le cadre de protocoles de communication).
- S'adapter ; à la nature périssable de certaines ressources (énergie) par le fonctionnement ; aux limites du matériel (surchauffe) ; aux capacités de stockage limitées. Ils doivent également intégrer l'instabilité des systèmes de communication utilisés (déconnexion, bande passante).
- Offrir une puissance de traitement suffisante pour satisfaire les contraintes de temps d'exécution des applications. Les processeurs utilisés dans les systèmes embarqués sont 2 à 3 fois moins puissants qu'un processeur d'un ordinateur PC.
- Revenir au moindre coût surtout lorsque le système est produit en grande série.
- Etre doté d'une grande sûreté de fonctionnement surtout pour les situations de défaillance qui mettent en péril des vies humaines où des investissements considérables (explosion de la fusée américaine Discovery).
- Ce type de systèmes sont dits « critiques » et ne doivent jamais faillir c.à.d. qu'ils doivent toujours donner des résultats corrects, pertinents et dans les délais attendus par les utilisateurs que ce soient des machines et/ou des humains.
- Avoir un poids efficient (un juste poids nécessaire au fonctionnement).

1.3.1 L'écart de Performance

La possibilité d'augmenter le nombre de transistors à l'intérieur d'un circuit intégré avec les années, en fonction de la loi de Moore, permet le maintien de la croissance de la performance au fil des années. Cependant, cette loi ne tiendra plus dans un proche avenir. La raison en est très simple: les limites physiques du silicium (Kim et al, 2003), (Thompson, et al, 2005).

Ainsi, les nouvelles technologies qui remplaceront complètement ou partiellement le silicium s'imposent. Toutefois, selon la feuille de route ITRS (ITRS, 2011), ces technologies ont, soit des niveaux de densité élevés et sont plus lentes que les CMOS, ou au contraire: les nouveaux

appareils peuvent atteindre des vitesses plus élevées, mais avec des exigences énormes en terme de consommation et de taille, même si l'on considère les technologies CMOS à venir.

En outre, les architectures de haute performance comme les machines super scalaires atteignent leurs limites. Selon (Borkar et al, 2011), (Flynn et al, 2005), et (Sima et al, 2004), il n'y a pas de nouveaux résultats de recherche dans de tels systèmes en ce qui concerne l'amélioration des performances. Les progrès de l'exploitation de l'ILP (Instruction Level Parallelism) stagnent: par exemple, dans la famille des processeurs Intel, l'efficacité globale (amélioration des performances des processeurs fonctionnant à la même fréquence d'horloge) n'a pas augmenté de façon significative depuis le Pentium Pro en 1995, Les nouvelles architectures Intel suivent la même tendance: la microarchitecture Core 2 n'a pas présenté une augmentation significative de son taux d'IPC (instructions par cycle), ainsi qu'il a été démontré dans (Prakash et al, 2008).

La stagnation des performances se produit parce que ces architectures défient certaines limites de l'ILP (Wall et al, 1991). Par conséquent, même de faibles augmentations de l'ILP sont devenues extrêmement coûteuses. L'une des techniques utilisées pour augmenter ILP est le choix judicieux de la largeur de répartition. Cependant, la largeur de l'expédition offre de graves répercussions sur la superficie totale du circuit. Par exemple, la zone de bloc de registres cubique augmente avec la largeur de la distribution, en tenant compte d'un processeur super scalaire typique tel que le MIPS R10000 (Burns, et al, 2002).

Dans (Austin et al, 2004), les «superordinateurs mobiles» sont abordés, qui sont des dispositifs embarqués qui devront effectuer plusieurs tâches de calcul intensif, telles que la reconnaissance vocale en temps réel, la cryptographie, la réalité augmentée, en plus de celles, classiques, comme le traitement de texte et l'e-mail. Même en tenant compte des processeurs informatiques de bureau, les nouvelles architectures pourraient ne pas répondre aux exigences des systèmes embarqués de l'avenir plus exigeant en puissance de calcul, donnant lieu à un écart de performance.

1.3.2 Les Contraintes de temps

A cause de la demande toujours croissante de puissance de traitement des applications, la performance globale du système embarqué liée au temps est un critère de conception important.

La performance temporelle

Dans la conception d'un système embarqué, la performance temporelle (exécution des taches en un temps court voire très court) des dispositifs est une préoccupation constante. C'est au niveau logiciel que cette performance s'exprime. En vue d'obtenir une meilleure performance

temporelle, il y a lieu d'agir sur les algorithmes en vue de réduire leur complexité, ainsi que sur les séquences fréquentes de code en les optimisant (la réutilisabilité).

Les systèmes embarqués temps réel

Beaucoup de systèmes embarqués interagissent directement avec leur environnement via des capteurs/actionneurs ou un réseau de communication sans fil. Ces interactions contraignent les temps de réponse du système embarqué de manière plus ou moins forte selon le domaine d'applications visé. Ce sont les systèmes temps réel ou plus précisément les systèmes embarqués temps réel (Benatitallah et al. 2006), (Falk et al. 2000) dans le sens où la durée de livraison des résultats d'un calcul fait partie intégrante de la spécification de ce dernier, au même titre que le résultat global. Il existe deux types de Systèmes Embarqués, les Systèmes Embarqués *temps réel strict* et les Systèmes Embarqués *temps réel souple*.

a. Les Systèmes embarqués temps réel strict

Dans les systèmes 'temps réel strict', ou dur, le non-respect des contraintes de temps du système, le plus souvent exprimées sous la forme d'échéances de terminaison, constitue une défaillance de l'application (Falk et al. 2000). Dans le cadre d'applications critiques, telles que par exemple certaines applications dans l'avionique, une telle défaillance peut avoir des conséquences catastrophiques, telles que la mise en danger de vies humaines ou des pertes financières importantes.

Etant donné les conséquences importantes du non-respect d'une échéance dans les systèmes temps réel strict, il est fortement recommandé pour de tels systèmes de pouvoir vérifier que toutes les échéances soient toujours respectées avant leur exécution.

Pour cela, des méthodes d'analyse d'ordonnançabilité (Selic et al. 1999) (Gerard et al. 2002) doivent être utilisées afin de procéder à cette vérification.

b. Les Systèmes embarqués temps réel souple

Dans les systèmes 'temps réel souple', bien que l'instant d'obtention du résultat soit important, la violation des contraintes de temps du système est tolérée même si elle demeure exceptionnelle (Falk et al. 2000), (Selic et al. 1999), (Gerard et al. 2002).

Cette tolérance est due au fait que les applications concernées ne relèvent pas du domaine des applications critiques. Les exemples typiques d'applications ayant des contraintes temps réel souples sont les applications multimédias à flux continus (transmission télévisée en direct… : le système vise au respect des contraintes de temps dans la délivrance des flux de données afin de garantir la qualité des images et du son). Toutefois, les contraintes de temps peuvent être adaptées (Falk et al. 2000) puisque une dégradation de la qualité des données ne sera que faiblement perçue

par l'utilisateur.

1.3.3 Les contraintes sur la consommation d'énergie

En plus de la performance, il faut tenir compte du fait que le plus grand problème potentiel dans la conception des systèmes embarqués est la consommation d'énergie excessive. Les systèmes embarqués futurs ne devraient pas dépasser 75 mW, car les batteries n'obéissent pas une loi équivalente à la loi de Moore (Austin et al, 2004). En outre, la fuite d'énergie est de plus en plus importante et, lorsqu'un système est en mode veille, la dissipation de puissance sera la principale source de consommation d'énergie. De nos jours, dans les microprocesseurs d'usage général, la dissipation de puissance se situe entre 20 et 30 W (en tenant compte d'un budget total de puissance de 100 W) (Thompson, et al, 2006).

On peut observer que, pour respecter les contraintes de puissance, les entreprises optent pour des puces multiprocesseurs afin de tirer tout le profit de la surface disponible, même s'il reste encore un énorme potentiel pour accélérer les programmes mono-thread. En fait, la stagnation de l'augmentation de la fréquence d'horloge, la consommation excessive d'énergie et les coûts matériels liés à l'exploitation des ILP, avec des technologies plus lentes, sont de nouveaux défis architecturaux qui doivent être traitées.

1.3.4 Les contraintes de Mémoire

Les mémoires ont été une préoccupation depuis les débuts des systèmes informatiques. Que ce soit du point de vue de la taille, du coût de fabrication, de la bande passante, de la fiabilité ou de la consommation d'énergie, une attention particulière a toujours été accordée lors de la conception de la structure mémoire d'un système. L'écart historique et toujours croissant entre le temps d'accès des mémoires et le débit de processeurs a également entraîné le développement de mémoires cache très avancées et très grandes, avec des systèmes de répartition et de remplacement complexes.

En outre, la capacité toujours croissante d'intégration des processus de fabrication a également alimenté l'utilisation de grandes caches sur puce, qui occupent une fraction importante de la surface de silicium pour la plupart des conceptions de circuits intégrés actuels. Ainsi, les mémoires représentent aujourd'hui une part importante du coût, des performances et de la consommation électrique globale de la plupart des systèmes, créant le besoin pour une conception soignée et le dimensionnement des sous-systèmes de mémoire.

Le développement de mémoires pour les systèmes embarqués actuels est soutenu principalement par la mise à l'échelle des transistors. Ainsi, les mêmes cellules de SRAM, DRAM et Flash ont été utilisées, génération après génération avec des transistors de plus en plus petits.

Bien que cette approche améliore la latence et de la densité, elle apporte aussi plusieurs nouveaux défis. Comme la fuite de courant ne diminue pas au même rythme que l'augmentation de la densité, la dissipation de puissance statique devient une préoccupation majeure pour les architectures de mémoire, ce qui conduit à des efforts communs à tous les niveaux de conception. Bien que la recherche au niveau de l'appareil tente de fournir des cellules de faible fuite (Yang et al, 2011), la recherche au niveau de l'architecture tente d'éteindre les banques de mémoire à chaque fois que c'est possible.

De plus, la réduction de la charge critique augmente les taux d'erreurs et exerce une pression accrue sur les techniques de correction d'erreurs, en particulier pour les applications critiques. Les tailles réduites augmentent également la variabilité du procédé, entraînant des pertes de rendement. Ainsi, des recherches approfondies sont nécessaires pour maintenir les performances et les économies d'énergie attendues pour prochaines générations de systèmes embarqués, tout en ne mettant pas en péril le rendement et la fiabilité.

Un autre grand défi se pose avec les difficultés croissantes dans la mise à l'échelle CMOS. De nouvelles technologies de mémoire devraient remplacer à la fois les mâtereaux volatils et non volatils utilisés de nos jours. Ces technologies devraient fournir une faible consommation d'énergie, une faible latence d'accès, une fiabilité élevée, une densité élevée et, plus important encore, un coût très faible par bit (ITRS, 2011). Comme le couplage des caractéristiques requises sur les nouvelles technologies est une tâche très exigeante, plusieurs concurrents se présentent comme des solutions possibles, comme les cellules ferroélectriques, nano électromécaniques et organiques (ITRS, 2011). Chaque type de mémoire a des tâches spécifiques dans un MPSoC. Étant donné que la mémoire est une grande partie de tout système aujourd'hui, apportant des coûts évidents et des problèmes de dissipation d'énergie, le défi est de rendre son utilisation aussi efficace que possible, en utilisant peut-être des informations recueillies a l'exécution ou fournie par l'applications et non disponibles au moment de la conception.

1.3.5 Réutilisation de code binaire existant

Parmi les milliers de produits lancés par les sociétés d'électronique grand public, on peut observer ceux qui rencontrent un grand succès et ceux qui ne réussissent pas. L'explication n'est peut-être pas seulement liée à leur qualité, mais également a leur standardisation dans l'industrie et le souci que peut avoir l'utilisateur final a propos de la fréquence des mises à jour du produit en cours d'acquisition.

L'architecture x86 est l'un de ces grands exemples. Compte tenu des normes actuelles, le x86

ISA (Instruction Set Architecture) lui-même ne suit pas les dernières tendances des architectures processeur. Il a été développé à une époque où la mémoire a été considéré comme très coûteuse. Le x86 ISA est un exemple typique d'une machine CISC traditionnelle. De nos jours, les architectures compatibles x86 récentes passent par des pipelines supplémentaires ainsi qu'un espace considérable dans la logique de contrôle et la ROM micro programmable juste pour décoder ces instructions CISC en instructions RISC. De cette manière, il est possible de mettre en œuvre des pipelines profonds et toutes les autres techniques RISC haute performance, tout en conservant le jeu d'instructions x86 et, par conséquent, la compatibilité de logiciel descendante.

Bien que de nouvelles instructions ont été inclus dans le jeu d'instructions x86 d'origine, comme ceux des SIMD, MMX, SSE (Conte et al, 2000), pour cibler les applications multimédia, il ya toujours un support pour les 80 instructions d'origine mises en œuvre dans le premier processeur x86. Cela signifie que n'importe quel logiciel écrit pour tout processeur x86 dans le passé, même ceux lancés à la fin des années 1970, peut être exécutée sur le dernier processeur Intel. C'est l'une des clés de la réussite de cette famille: la possibilité de réutiliser le code binaire existant sans aucune modification. Ce fut l'une des principales raisons pour lesquelles ce produit est devenu le leader dans son segment de marché. Intel pourrait garantir à ses consommateurs que leurs programmes ne seraient pas obsolètes au cours d'une longue période de temps et, même en migrant pour un système plus rapide, ils auront toujours la possibilité de réutiliser et exécuter à nouveau le même logiciel.

Par conséquent, des entreprises comme Intel et AMD continuent à mettre en œuvre des techniques super scalaires plus consommatrice d'énergie, afin d'augmenter leurs fréquences à l'extrême (Koufaty et al, 2003). Toutefois, le principe de base utilisé pour les architectures haute performance est toujours le même: la super scalarité. Comme les systèmes embarqués sont de plus en plus basés sur une quantité énorme de développement de logiciels, le coût de maintien du code existant devra être pris en considération lorsque de nouvelles plates-formes viendront sur le marché.

1.3.6 Rendement et coûts de fabrication

De nos jours, il faut une grande quantité de ressources pour créer une conception ASIC d'un volume, complexité et économie d'énergie plus ou moins importants. Certaines entreprises de conception peuvent encore réussir à le faire parce qu'elles ont les concepteurs, les infrastructures et l'expertise nécessaires. Toutefois, pour les mêmes raisons, il ya des entreprises qui ne peuvent tout simplement pas se le permettre. Pour ces entreprises, un tissu plus régulier semble le meilleur compromis pour l'utilisation d'un processus avancé.

Les problèmes de l'augmentation des coûts de conception et du turnarround sont encore plus sensibles en raison des pressions du marché. Le temps disponible pour une société afin d'introduire un produit sur le marché est en baisse. De cette façon, la conception de nouveaux circuits est de plus en plus dictée par les préoccupations du temps d'accès au marché.

Néanmoins, il y aura un point où, si la société a besoin d'une implémentation silicium plus personnalisée, il sera nécessaire de payer les coûts des masques et de la production. Cependant, l'économie pousse clairement les concepteurs vers des structures plus régulières qui peuvent être fabriqués en grandes quantités. Les fabriques ordinaires résoudraient le coût du masque et de nombreuses autres questions telles que l'imprimabilité, l'extraction, l'intégrité de puissance, les tests et le rendement. La figure 1.2 représente le cycle de vie typique d'un système logiciel embarque. Ainsi, une certaine forme d'adaptation doit être présente pour assurer que, des appareils produits en masse a faible coût peuvent encore être optimisés pour des applications avec des besoins différentes, sans nécessite un redésigne et de nouveaux coûts de fabrication.

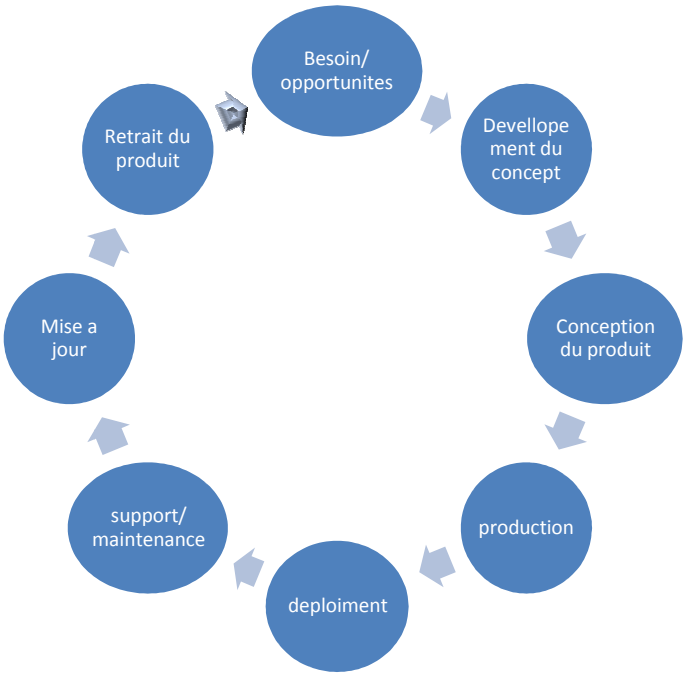


Figure 1.2 le cycle de vie d'un système embarqué

1.3.7 La communication

Compte tenu des limites croissantes de la consommation d'énergie et de la complexité croissante de l'amélioration des niveaux actuels d'exploitation des PIL, la tendance à intégrer de multiples noyaux de traitement dans une seule puce est devenue une réalité. Alors que l'utilisation de plusieurs processeurs fournit des ressources plus gérables, qui peuvent être désactivées

indépendamment pour économiser de l'énergie, par exemple (Isci et al, 2006), il est crucial qu'ils puissent communiquer entre eux d'une manière efficace, Avec un parallélisme au niveau des fils. On attend de l'infrastructure de communication une bande passante élevée, une latence faible, une faible consommation d'énergie, des coûts de fabrication faibles et une fiabilité élevée, avec plus ou moins de pertinence pour chaque fonctionnalité, selon l'application. Même si cela peut être une tâche simple pour un petit ensemble de processeurs, il devient de plus en plus complexe pour un plus grand ensemble de processeurs. De plus, outre les processeurs, les Soc intégrés comprennent des composants hétérogènes, tels que des accélérateurs dédiés et des interfaces de communication hors circuit, qui doivent également être interconnectés.

Comme les approches classiques telles que les bus ou les mémoires multiports partagées ont une mauvaise évolutivité, de nouvelles techniques de communication et des topologies sont nécessaires pour répondre aux exigences des nouveaux MPSoCs avec de nombreux noyaux et des limites strictes de zone et de puissance. Parmi ces techniques, les réseaux sur puce (NoCs) ont reçu une grande attention au cours des dernières années, car ils apportent une grande évolutivité et une bande passante élevée comme des actifs importants (Bjerregaard et al, 2006). Avec les NoCs comme une interconnexion prometteuse pour MPSoCs, plusieurs questions doivent être abordées, telles que l'organisation optimale de la mémoire, le mécanisme de routage, l'ordonnancement des threads et le placement, et ainsi de suite. En outre, comme tous ces choix de conception sont très dépendants de l'application, il existe une grande marge de manœuvre pour l'adaptabilité aussi sur l'infrastructure de communication, non seulement pour les NoC mais pour tout schéma choisi couvrant le tissu de communication.

1.3.8 Tolérance aux pannes

Certains systèmes embarqués doivent pouvoir remplir leurs fonctions malgré la présence de fautes [92], qu'elles soient d'origine physique ou humaine. La tolérance aux pannes a attiré de plus en plus d'attention au cours des dernières années en raison de la vulnérabilité intrinsèque que les technologies *deep-submicron* ont imposées. À mesure que l'on se rapproche des limites physiques de la technologie CMOS actuelle, l'impact des effets physiques sur la fiabilité du système est amplifié. Ceci est une conséquence de la susceptibilité qu'un circuit très fragile lorsqu'il est exposé à de différents types de conditions extrêmes, telles que des températures et tensions élevées, des particules radioactives venant de l'espace ou des impuretés présentes dans les matériaux utilisés pour l'emballage ou la fabrication du circuit, Etc. Indépendamment de l'agent qui cause le défaut, les prédictions sur

Les futurs circuits à l'échelle nanométrique indiquent un besoin majeur de solutions de tolérance aux pannes pour faire face aux taux de défauts élevés attendus (White et al, 2008).

Des solutions de tolérance aux pannes existent depuis 1950, d'abord dans le but de travailler dans des environnements hostiles et éloignés de missions militaires et spatiales. Plus tard, pour répondre à la demande de systèmes d'applications critiques à haute fiabilité, tels que les systèmes bancaires, le freinage automobile, les avions, les télécommunications, etc. (Pradhan et al, 1996). Le principal problème des solutions mentionnées est le fait qu'elles sont ciblées pour éviter qu'un défaut affecte le système à tout prix, puisque tout problème pourrait avoir des conséquences catastrophiques. Pour cette raison, dans de nombreux cas, il n'y a pas de problème avec la zone / puissance / performance que la solution de tolérance aux pannes peut ajouter au système

Dans ce sens, le principal défi est de permettre le développement de systèmes embarqués de hautes performances, en tenant compte de tous les aspects mentionnés précédemment, comme la consommation d'énergie, les applications à comportement hétérogène, la mémoire, etc. tout en offrant un système hautement fiable qui Peut faire face à un grand nombre de défauts. Par conséquent, ce besoin toujours croissant de systèmes à faible consommation d'énergie, tolérant aux pannes, à haute performance, à faible coût conduit à une question essentielle: quelle est la meilleure approche tolérante aux pannes pour les systèmes embarqués, suffisamment robuste pour gérer les faiblesses élevées et Causer un faible impact sur tous les autres aspects de la conception du système embarqué? La réponse change entre les applications, le type de tâche et la plate-forme matérielle sous-jacente. Encore une fois, la clé pour résoudre ce problème à différentes instances repose sur des techniques adaptatives pour réduire les coûts et maintenir la performance.

1.3.9 Hétérogénéité et évolutivité de plates-formes matérielles.

Le matériel adaptatif pose de véritables défis pour l'ingénierie logicielle, de l'élimination des besoins aux phases de développement de logiciels. Les difficultés pour l'ingénierie logicielle sont dues à la grande flexibilité et à l'espace de conception qui existe dans les plates-formes matérielles adaptables. Outre le comportement principal implémenté par le logiciel, c'est-à-dire les exigences fonctionnelles, une plate-forme matérielle adaptative dévoile un large éventail de conditions non fonctionnelles qui doivent être satisfaites par le logiciel en cours d'exécution et supportées par le processus d'ingénierie logicielle. Même aujourd'hui, les exigences non-fonctionnelles sont un fardeau pour le développement de logiciels, Bien qu'il soit possible de contrôler certaines exigences classiques, tels que la performance ou la latence, ceux plus spécifique au domaine étudié, tels que l'énergie et la puissance, restent des questions de recherche ouvertes.

Le logiciel embarqué a radicalement changé à un rythme rapide en seulement quelques années. Une fois hautement spécialisé pour exécuter seulement quelques tâches, telles que le décodage vocal, ou l'organisation d'un simple annuaire téléphonique dans le cas des téléphones mobiles, le logiciel que nous trouvons aujourd'hui dans n'importe quel smart phone de grand publique contient plusieurs API interconnectées et des Framework différents pour offrir une expérience complètement nouvelle à l'utilisateur. Le logiciel embarqué est maintenant multitâche et s'exécute en parallèle, puisque même les périphériques mobiles contiennent un ensemble distinct de microprocesseurs, chacun dédié à une certaine tâche, comme le traitement de la parole et les graphiques.

Ces architectures distinctes existent et sont nécessaires pour économiser l'énergie. Le gaspillage des ressources informatiques et énergétiques est un luxe que les ressources limitées ne peuvent pas se permettre. Cependant, le matériel complexe et hétérogène ci-dessus, qui prend en charge plus d'une architecture d'ensemble d'instructions (ISA), a été conçu pour être efficace sur le plan des ressources et ne pas faciliter la conception et la production de logiciels.

En outre, comme il existe potentiellement plusieurs nœuds informatiques, un logiciel parallèle conçu pour occuper efficacement le matériel hétérogène est également nécessaire pour économiser de l'énergie. Inutile de dire que la conception de logiciels parallèles est difficile. Si le logiciel n'est pas bien conçu pour tirer parti et utiliser efficacement toutes les ISA disponibles, le concepteur de logiciel manquera probablement un point optimal d'utilisation des ressources, produisant des applications très gourmandes en énergie. On peut facilement imaginer plusieurs d'entre eux fonctionnant simultanément, en provenance d'éditeurs de logiciels inconnus et distincts, implémentant des fonctionnalités imprévues. Il est alors claire que la conception et le développement de logiciels pour ces dispositifs peut être extrêmement complexe.

Si les plates-formes matérielles adaptables sont destinées à être des périphériques programmables dans un proche avenir, leur ingénierie logicielle doit traiter de façon transparente leur complexité intrinsèque, en supprimant cette charge du code. Dans le domaine des systèmes intégrés adaptatifs, le logiciel continuera d'être la véritable source de différenciation entre les produits concurrents et l'innovation pour les entreprises de consommation électroniques. Un nouvel environnement de langages de programmation, d'outils de développement de logiciels et de compilateurs peut être nécessaire pour soutenir le développement de logiciels adaptatifs ou, du moins, un profond remaniement des technologies existantes. L'industrie utilise une myriade de langages de programmation et de modélisation, de systèmes de versioning, de conception de logiciels et d'outils de développement, pour ne citer que quelques-unes des technologies clés, pour

continuer à offrir de l'innovation dans leurs produits logiciels. La grande question est de savoir comment faire en sorte que ces technologies évoluent en termes de productivité, de fiabilité et de complexité pour le nouveau scénario d'ingénierie logicielle créé par les systèmes adaptatifs.

1.4 Conclusion

L'industrie des logiciels embarqués a beaucoup évolués au cours des dernières décennies. Aussi bien à cause des contraintes liées au processus et technologies de développement qu'au contraintes du marcher ou à la démocratisation à grande échelle de ces systèmes, ils se trouvent aujourd'hui confrontés à de nouveaux défis. Dans ce chapitre, nous avons expliqué la nature de ces défis et nous avons montré que l'adaptabilité à l'exécution est devenue une prérogative majeure.

Dans la suite de ce document, nous présenterons notre travail qui va dans le sens de l'amélioration des capacités d'adaptation des systèmes logiciels embarqués. D'abord nous discuterons dans le chapitre suivant, les approches existantes, principalement basées sur le concept de lignes de produits dynamiques. Une étude critique de ces dernières nous permettra d'identifier leurs principales faiblesses et justifiera l'introduction de notre propre proposition dans la deuxième partie de ce manuscrit.

Chapi

II

Ingénierie des lignes de produits logiciels

2.1 Introduction

Dans ce chapitre, nous présentons brièvement l'ingénierie des lignes de produits logiciels ainsi que les principaux concepts qui s'y rapportent tels que la personnalisation, la réutilisation, et la variabilité. Nous décrivons notamment la notions centrale de séparation entre l'ingénierie du domaine et l'ingénierie des applications. En outre, nous présentons quelques approches, techniques et outils pour modéliser, gérer et réaliser la variabilité dans une famille de logiciels.

2.2 Les lignes de produits logiciels

Traditionnellement, le génie logiciel s'intéressait au développement des systèmes logiciels individuels, c'est-a-dire un système logiciel à la fois. Un processus de développement typique commence par l'analyse des exigences des clients, puis plusieurs étapes de développement (spécification, conception, Mise en œuvre et tests). Le résultat obtenu est un produit logiciel unique.

Au contraire, l'ingénierie des lignes de produits logiciels (LPL) vise à développer plusieurs systèmes logiciels similaires à partir d'une base de code commune (Bass et al. 1998, Clements et Northrop 2001 Pohl et al. 2005).

2.2.1 Personnalisation et réutilisation à grande échelle

L'ingénierie des lignes de produits logiciels repose sur l'idée de la personnalisation à grande échelle (Pine 1999) connu de nombreuses industries. Par exemple, dans l'industrie automobile, l'accent est mis sur la création d'une ligne de production unique à partir de laquelle de nombreuses variations personnalisées, mais similaires d'un modèle de voiture sont produites. La personnalisation à grande échelle tire profit du principe de similarité et de la conception modulaire pour produire massivement des produits personnalisés. De nombreuses industries, comme l'avionique, les télécommunications, ou de l'industrie automobile, construisent continuellement et massivement le même produit logiciel multiple et similaire. Il est donc possible de réutiliser des artefacts logiciels. La personnalisation massive de logiciel, et donc l'ingénierie des LPL, se concentrer sur les moyens de produire efficacement et de maintenir de multiples produits logiciels similaires, exploitant ce qu'ils ont en commun et gérant ce qui varie entre eux.

L'ingénierie des LPL Vise donc à développer de manière systématique et coordonnée des

variantes connexes et des solutions sur mesure pour différents clients. Au lieu de développer individuellement des variantes à partir de zéro, les points communs ne sont conçus qu'une seule fois.

La définition suivante donnée par Clements et Northrop saisit l'idée générale derrière une ligne de produits logiciels:

«Une ligne de produits logiciels est un ensemble de systèmes à forte intensité de logiciels qui partagent un ensemble commun de fonctionnalités qui répondent aux besoins spécifiques d'un segment de marché particulier ou qui sont développés à partir d'un ensemble commun de composants de base d'une manière ordonnée. " (Clements et Northrop 2001

Historiquement, l'idée de développer un ensemble de produits logiciels connexes peut être retracée à l'idée des familles de programme (Parnas 1976). Les LPL ont connu un véritable seulement au milieu des années 1990, lorsque le logiciel a commencé à s'intégrer massivement dans les familles de produits électroniques, principalement les téléphones mobiles étant le plus populaires, mais plusieurs autres domaines, tels que les systèmes automobiles, les systèmes médicaux, l'aérospatiale ou les télécommunications ont également été ciblés par les LPLs. (Maccari et Heie 2005)

Une caractéristique importante de l'ingénierie des LPL est l'accent mis sur la réutilisation du logiciel (Jacobson Et coll. 1997). Une définition générale de la réutilisation du logiciel est donnée dans (Krueger 1992) «La réutilisation du logiciel est le processus de création de logiciels à partir de logiciels existants plutôt que de construire des systèmes logiciels à partir de zéro. "

La réutilisation, en tant que stratégie logicielle visant à réduire les coûts de développement et à améliorer la qualité, n'est pas une idée nouvelle. Elle a été un sujet populaire depuis les débuts de la discipline du génie logiciel (McIlroy 1968 Parnas 1976). Les stratégies de réutilisation passées, axées sur la réutilisation de petits bouts de codes ou son clonage opportuniste d'un code conçu pour un système dans un autre, n'ont pas été rentables. La programmation orientée objet offre plusieurs mécanismes pour développer des unités génériques de fonctionnalité, mais la réutilisation est généralement effectuée a une petite échelle. Les Frameworks orientés objet fournissent des concepts réutilisables mais sont souvent grands, complexes Et / ou mal documentés de sorte que la localisation et l'instanciation de ces concepts pour l'implémentation un système logiciel est difficile même avec des compétences étendues. Les composants en tant qu'unités de composition - indépendantes et prêtes à l'emploi avec des interfaces contractuellement spécifiées promettent une réutilisation a grande échelle, mais le compromis optimal entre la fonctionnalité, la taille et la réutilisation des composants est très difficile à trouver. (Szyperski et al. 2002)

La réutilisation est toujours un problème difficile dans le génie logiciel. Le développement de familles de logiciels plutôt que des systèmes individuels peut faciliter la réutilisation, puisque les produits de la famille partagent des fonctionnalités communes et sont du même domaine. Ainsi :

:

«La réutilisation à grande échelle fonctionne le mieux dans les familles des systèmes connexes et est donc un dépendante du domaine." (Verre 2001)

L'ingénierie des LPL est basée sur cette idée. Il s'agit d'un changement de paradigme vers une approche systématique à la réutilisation des artefacts communs (également appelés composants de base ou core assets), ou les produits connexes sont traités comme une famille de produits. Leur co-développement peut alors être planifié dés le départ, au lieu de commencer à partir de zéro ou de copier et de modifier à partir d'un produit précédent. Les LPLs ont plusieurs avantages, en particulier pour les organisations qui développent des produits logiciels intensifs, mais qui nécessitent de plus en plus d'effort en raison de la diversité et de la complexité de ces produits.

2.2.2 Avantages escomptés

Par rapport au développement traditionnelle mono-produit, les LPLs promettent plusieurs avantages (Bass Et al. 1998, Clements et Northrop 2001 Pohl et al. 2005): en raison du co-développement et de la réutilisation systématique, les produits logiciels devraient être produits plus rapidement, avec des coûts plus faibles, et une meilleure qualité. La possibilité de personnaliser des produits logiciels dans différents contextes ouvre de nouvelles perspectives., dans les systèmes embarqués par exemple les lignes de produits logiciels permettent de réduire des contraintes traditionnelles liées à la rareté des ressources et à l'hétérogénéité du matériel, par ce que les variantes peuvent être efficacement adaptées à un ou plusieurs périphériques spécifiques (Beuche et al. 2004). Les succès de la personnalisation en masse de logiciels parviennent de domaines variés tels que les téléphones mobiles, les réseaux de télécommunications, les systèmes médicaux, les imprimantes, les voitures, les bateaux, les avions.

De nombreuses entreprises parmi lesquelles Alcatel, Hewlett Packard, Philips, Boeing et Robert Bosch GmBh7 ont rapporte leurs expériences lors de la conférence SPLC 2000 avec des avantages significatifs (Northrop 2002). Par exemple, Bass et al. rapporte qu'avec les LPL, Nokia peut produire 30 au lieu de 4 modèles de téléphone par an; Cummins, Inc. a réduit le temps de développement logiciel pour un nouveau moteur diesel d'une année à une semaine; Motorola a observé une hausse de 400% d'augmentation de la productivité, etc. (Bass et al. 1998).

2.2.3 Stratégies extractives, réactives et proactives

Bien que de nombreuses organisations soient conscientes des énormes avantages potentiels de la personnalisation, les coûts, les risques et les ressources associées sont une barrière d'adoption prohibitive pour beaucoup (Krueger 2001). Historiquement, les pratiques de l'ingénierie LPL induisent un énorme investissement. Dans le pire des cas, il s'agit d'analyser, d'archiver, de concevoir et d'implémenter toutes les variations de produits dans un horizon prévisible. Cette approche proactive pourrait convenir à des organisations qui peuvent prédire leurs besoins en LPL et qui disposent du temps et des ressources nécessaires pour un cycle de développement prolongé. Malheureusement, toutes les organisations ne peuvent pas se permettre de ralentir ou d'arrêter la production pendant quelques mois, même si le retour sur investissement peut potentiellement être énorme.

Clements et Krueger font valoir que les organisations en transition à l'ingénierie LPL doivent adopter un processus plus souple, avec des méthodes d'adoption peu onéreuses, qui vont au-delà d'une approche purement proactive (Clements et Krueger 2002). Krueger propose trois stratégies d'adoption : Proactives, réactives et extractives - qui peuvent éventuellement être combinées par les organisations LPL (Krueger, 2001; 2006).

Avec l'approche proactive, l'organisation analyse, conçoit et met en œuvre complètement la ligne de production pour soutenir la pleine gamme de produits nécessaires dans l'horizon prévisible. A partir de cela, un ensemble complet des exigences communes et variables, des définitions de produits, de code source, etc. sont mises en œuvre.

Avec l'approche réactive, l'organisation développe de manière incrémentale, leur ligne de production logicielle lorsque la demande se fait pour de nouveaux produits ou de nouvelles exigences sur les produits existants. Les artefacts communs et variables sont étendus progressivement en réaction aux nouvelles exigences. Cette approche incrémentale offre une solution plus rapide et moins coûteuse pour la transition vers la personnalisation de masse de logiciels.

Avec l'approche extractive, l'organisation capitalise sur des logiciels personnalisés existants en extrayant le code source commun et variable dans une seule ligne de production.

Ce niveau élevé de réutilisation des logiciels permet à une organisation d'adopter très rapidement la personnalisation de masse des logiciels. Par exemple, les développeurs adoptent souvent une approche extractive pour la refactorisation et la décomposition d'une ou de plusieurs applications héritées en unités réutilisables et composables.

Ces approches ne sont pas nécessairement mutuellement exclusives. Par exemple, une

approche commune consiste à amorcer un effort de personnalisation de masse en utilisant l'approche extractive et ensuite passer à une approche réactive pour développer progressivement le LPL au fil du temps.

Northrop propose une classification à ce sujet et distingue les différentes stratégies de production pour la réalisation des composants de base (Northrop 2002). La stratégie de production peut être de haut en bas (à partir d'un ensemble de composants de base et générant des produits a partir d'eux), ou de bas en haut (En commençant par un ensemble de produits et en généralisant leurs composants pour produire la Ligne de produit), ou un mélange des deux.

2.2.4 Ingénierie des domaines et ingénierie des applications

L'ingénierie des LPLs est séparée en deux phases complémentaires. L'ingénierie du domaine est concernée par le développement pour la réutilisation alors que l'ingénierie d'application est vise au développement par la réutilisation.

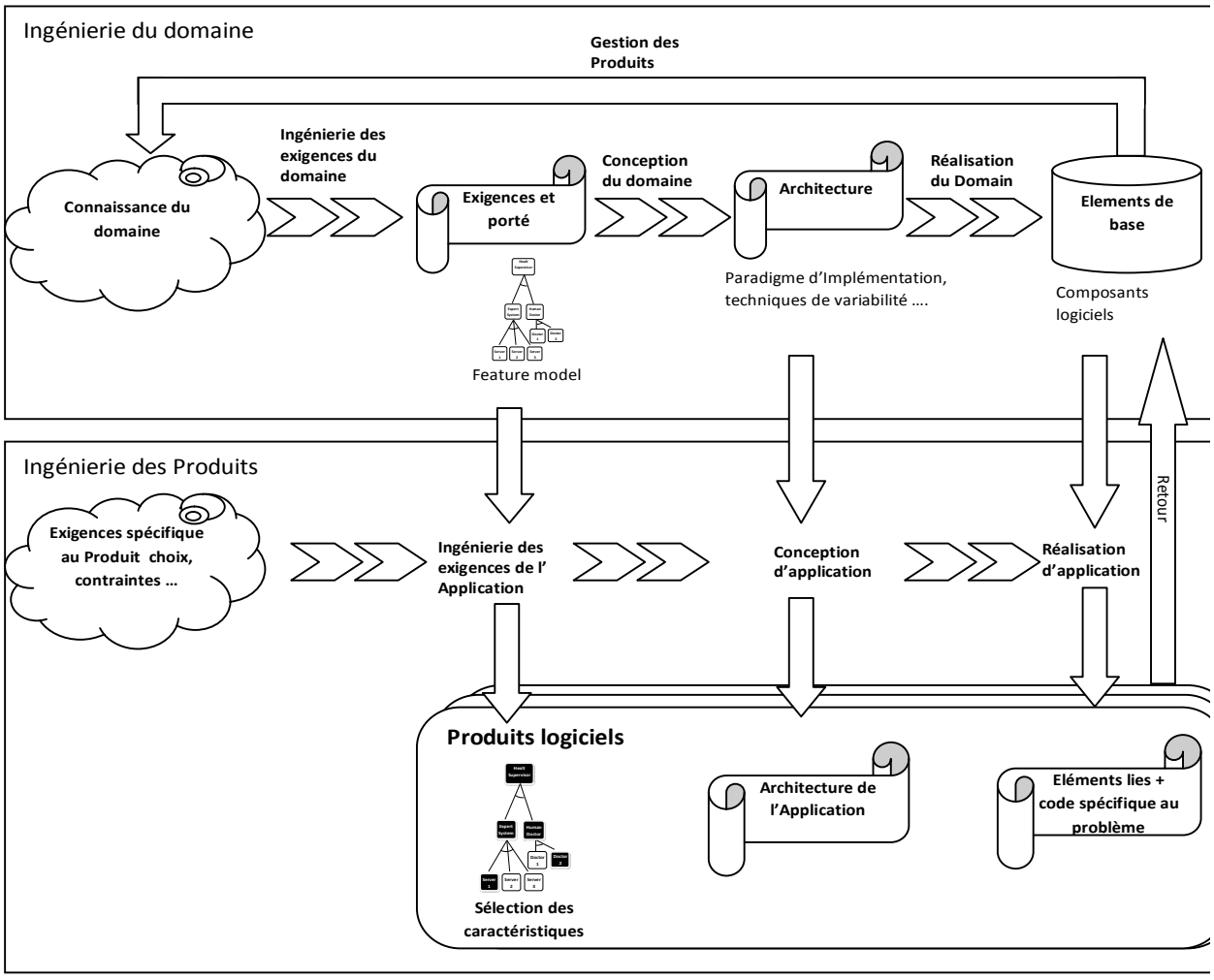


Figure 2.1 l'ingénierie du domaine et l'ingénierie d'application (Van der Linden, 2002)

L'idée derrière cette approche de l'ingénierie LPL est que les investissements requis pour développer les artefacts réutilisables pendant l'ingénierie du domaine, sont compensés par les avantages de la dérivation des produits individuels lors de l'ingénierie d'application (Deelstra et al. 2004; 2005). Une raison fondamentale pour la recherche et l'investissement dans les technologies sophistiquées pour les LPLs est d'obtenir le maximum de bénéfice de cet investissement initial, autrement dit, pour réduire au minimum la proportion des coûts d'ingénierie d'applications (voir Figure 2.1).

Ingénierie du domaine. Le processus de développement d'un ensemble de produits connexes (c.-à-d. Un LPL) au lieu d'un seul est appelé ingénierie de domaine. Une LPL doit remplir non seulement les exigences d'un seul client, mais les exigences de plusieurs clients dans un domaine, y compris les clients actuels et potentiels. Par conséquent, l'ensemble du domaine et ses besoins potentiels sont analysés, par exemple, pour identifier les différences entre les produits, les artefacts réutilisables et pour planifier leur développement, etc. L'ingénierie de domaine est le développement de la réutilisation: Les artefacts (exigences, composants, cas de test, etc.)Communs et réutilisables sont factorisés afin que leur réutilisation soit facilitée. Il est généralement composé de quatre activités: analyse de domaine, conception de domaine, le codage de domaine et le test de domaine. Dans l'analyse du domaine, les points communs et les différences entre les variantes potentielles sont identifiés et décrites, par exemple en termes de caractéristiques (fonctionnalités ou encore features). Dans ce contexte, une caractéristique est une abstraction de domaine de première classe, généralement un incrément visible dans la fonctionnalité de l'utilisateur final (Apel et Kästner 2009) (voir la section 2.4 pour une analyse détaillée du terme caractéristique). Ensuite, les développeurs conçoivent et mettent en œuvre la LPL tel que différentes variantes peuvent être construites à partir de parties communes et variables.

Ingénierie d'application. C'est le développement par la réutilisation (aussi appelé Développement de produits). Les produits concrets sont dérivés à l'aide de composants communs et réutilisables. Les artefacts développés dans l'ingénierie du domaine. Il se compose de quatre activités, parallèles aux activités d'ingénierie du domaine: l'ingénierie des exigences applicatives, conception de l'application, codage de l'application et test de l'application. Ce processus est basé sur l'ingénierie du domaine et consiste à développer un produit final, par la réutilisation des artefacts réutilisables et à adapter le produit final à des exigences spécifiques. Idéalement, les exigences du client peuvent être mappées à des éléments (par exemple, des caractéristiques) identifiés lors de l'ingénierie du domaine, de sorte que la variante peut être construite à partir de parties communes et variables existantes par de la mise en œuvre de la LPL. Le processus de construction de produits à

partir des artefacts du domaine est appelé dérivation de produits. Selon la forme de l'implémentation, il peut y avoir différents processus d'ingénierie des applications, de l'effort de développement manuel à des technologies sophistiquées, y compris la configuration et la génération de variantes automatisées.

2.2.5 Espace du problème et espace de la solution

L'un des défis les plus difficiles pour l'ingénierie logicielle, et par extension de l'ingénierie des LPL, est la complexité inhérente des systèmes logiciels modernes. Comme indiqué dans (Brooks 1987), Il existe deux types de complexité. D'une part, la *complexité essentielle* est inhérente au problème traité et ne peut pas être supprimé. D'autre part, la *complexité accidentelle* n'est pas essentielle au problème et est plus liée aux questions de mise en œuvre. En particulier, l'utilisation d'une technologie inadéquate à l'égard du problème ciblé (par exemple, l'utilisation du langage d'assemblage pour développer un moteur de jeu d'échecs) peut ajouter une complexité accidentelle importante pendant le développement du logiciel. L'invention des langages de programmation de haut niveau, par exemple, avait nettement amélioré la complexité accidentelle.

Les termes *problème* et *solution* (ou "modèle de problème" et "modèle de solution", " domaine du problème "et" domaine de la solution "," espace du problème "et" espace de la solution ", …) sont communément utilisé pour désigner le contraste entre le système étudié et son domaine d'application (Génova et al. 2009). Cela peut également être exprimé avec les termes " Analyse du problème "et" conception des solutions ". Plusieurs paradigmes (voir ci-dessous) distinguent explicitement l'espace du problème et l'espace de la solution et proposent de séparer les deux espaces au cours du développement du logiciel (ou de la ligne de logiciel)

L'espace de problème comprend des abstractions spécifiques au domaine qui décrivent les exigences sur un système logiciel et son comportement prévu. Par exemple, l'analyse de domaine prend place dans l'espace du problème, et ses résultats sont généralement documentés en termes de caractéristiques. L'espace des solutions comprend des abstractions orientées vers la mise en œuvre, tels que des artefacts de code. Dans l'espace des solutions, les abstractions ont été utilisées dans le domaine des langages de programmation allant du langage d'assemblage aux langages orientés objet qui assistent le programmeur à organiser des informations structurelles et comportementales constituant des logiciels.

Entre les éléments de l'espace du problème (par exemple, les exigences, les caractéristiques) et les éléments de l'espace des solutions, il existe un mappage qui décrit quels artefacts de mise en œuvre correspondent à quelles exigences ou caractéristiques. Selon la méthode de mise en œuvre et

le degré d'automatisation, cette cartographie peut avoir différentes formes et complexités allant des mappages implicites basés sur des conventions de nommage à des règles complexes codées dans des générateurs ou des transformations (Czarnecki et Eisenecker 2000).

Les mappages entre l'espace du problème et l'espace des solutions peuvent être enchaînés; Un mappage pourrait prendre deux ou plusieurs spécifications et les mapper à un (ou plusieurs) espace de solution (ceci est commun lorsque différents aspects d'un système sont représentés); Un mappage peut également implémenter l'espace du problème en termes de deux ou plusieurs espaces de solution (Czarnecki et Eisenecker 2000). En outre la transition entre l'ingénierie du domaine et l'ingénierie de l'application peut être plus ou moins complexe (par exemple, le degré d'automatisation peut varier)

2.3 Gestion de la variabilité

L'ingénierie du domaine et l'ingénierie des applications (et par voix de conséquence, l'espace du problème et l'espace de la solution) décrivent un cadre de processus très général (voir Figure 2.1) pour l'ingénierie des LPL. Ainsi, la gestion de la variabilité est un concept central et unique à l'ingénierie des LPL. C'est le processus de factorisation des points communs et de systématisation des variabilités de la documentation, exigences, code, artefacts de test et des modèles. Pour chaque étape des Frameworks proposés, différentes approches, formalismes et outils peuvent être utilisés. Par exemple, il existe différentes approches de spécification de la portée (John et Eisenbarth, 2009), différent mécanismes qui permet d'effectuer une analyse du domaine ou d'un modèle de la variabilité (voir la section 2.3.2) et différents mécanismes de mise en œuvre (voir la section 2.3.3).

2.3.1 Variabilité

Plusieurs définitions de la variabilité ont été données dans la littérature.

La variabilité dans le temps et la variabilité dans l'espace. Les travaux existants sur la gestion de la variabilité des logiciels peuvent être grossièrement divisés en deux catégories. La variabilité dans le temps et la variabilité dans l'espace sont généralement considérées comme des dimensions fondamentalement distinctes dans l'ingénierie des LPLs. Pohl et al. Définit la variabilité dans le temps comme "l'existence de différentes versions d'un artefact qui sont valables à des moments différents » et la variabilité dans l'espace comme « l'existence d'un artefact dans des formes différentes dans le même temps » (Pohl et al. 2005). La variabilité dans le temps est principalement liée à la gestion des variations du programme dans le temps et comprend les systèmes de contrôle des versions et plus généralement le domaine de la gestion des configurations

logicielles. L'objectif de l'ingénierie des LPL est principalement de faire face à la variabilité dans l'espace (Erwig 2010, Erwig et Walkingshaw 2011).

Régularité et variabilité. Weiss et Lai définissent la variabilité des LPL comme «une hypothèse sur la façon dont les membres d'une famille peuvent différer les uns des autres » (Weiss et Lai, 1999). Ainsi la variabilité spécifie les particularités d'un système correspondant aux attentes spécifiques d'un client, tandis que les points communs spécifient des hypothèses qui sont vraies pour chaque membre de la LPL. (Svahnberg et al. 2005) adopte un point de vue logiciel et définit la variabilité comme «la capacité d'un système logiciel ou d'un artefact à être efficacement étendue, modifiée, personnalisée ou configurée pour une utilisation dans un contexte particulier. » Ces deux définitions sont complémentaires pour capturer la notion de variabilité: la première est plus liée aux notions de domaine et de points communs tandis que la seconde se concentre davantage sur l'idée de la personnalisation. Néanmoins, il n'y a pas une perception ou une définition unique, de la variabilité: (Bachmann and Bass 2001) propose différentes catégories de variabilités, (Svahnberg et al. 2005) ont défini cinq niveaux de variabilités alors que certains auteurs distinguent la variabilité essentielle et technique (Halmans et Pohl 2003), la variabilité interne et externe (Pohl et al. 2005), ou encore la variabilité de la ligne de produit et celle du logiciel (Metzger et al. 2007).

2.3.2 Modélisation de la variabilité

La gestion de la variabilité devient une préoccupation primordiale pour maîtriser la complexité d'un LPL. Il est clair qu'il est nécessaire de modéliser des points de variation et leurs variantes pour exprimer la variabilité (Jacobson et al. 1997). Dans les phases initiales d'un projet LPL, une communication entre les concepteurs de la LPL et les clients sur les exigences et donc les variabilités est un facteur de succès critique. Une façon de communiquer efficacement sur la variabilité est de la modéliser. Les langages de modélisation de la variabilité sont donc utilisés pour produire des modèles de variabilité. Les langages de modélisation de la variabilité peuvent être graphiques, textuels ou un mélange des deux. Les modèles de variabilité décrivent les points de variation et leurs variantes, facilitent l'identification et la délimitation (ou la portée) de la variabilité. Ils sont également utilisés par de nombreuses approches d'implémentations et pour la génération automatisée de variantes ou pour le raisonnement et la détection des erreurs (voir section 2.6).

Les modèles de variabilité. Le modèle de caractéristique (feature model ou simplement FM) est actuellement la technique la plus populaire pour modéliser la variabilité. De nombreuses extensions et dialectes des features models ont été proposés dans la littérature (par exemple, FORM

(Kang et al. 1998), FeatureRSEB (Griss et al. 1998), (Riebisch 2003); (Beuche Et al. 2004), (Czarnecki et al. 2005b); (Schobbens et al. 2007), (Asikainen et al. 2006; 2007)).

Il existe également d'autres approches pour décrire séparément la variabilité (les modèles de variabilité (OVM) (Pohl et al. 2005) ou Covamof (Sinnema et al. 2006, Sinnema et Deelstra 2008)). Metzger et al. Ont montrer comment convertir un OVM en un modèle de caractéristiques, de sorte que les deux formalismes peuvent être combinés pour modéliser la variabilité (Metzger et al. 2007). Une approche similaire pour représenter la variabilité dans un modèle séparé peut également être les modèles de décision (Rabiser et al. 2007). Les features models peuvent également être comparés à des configurations dans l'industrie manufacturière. Une telle approche est illustrée par un configurateur appelé WeCoTin (Asikainen et al. 2004). Chen et al. A sélectionné et évalué 32 approches dans la littérature et rapporte un état des lieux sur l'utilisation des modèles de variabilité: quatorze approches utilisaient les features models, six approches utilisaient la modèles de décisions et douze approches utilisaient d'autres types de modèles de variabilité (Chen et al. 2009).

2.3.3 L'implémentation de la variabilité

L'implémentation de la variabilité est essentielle à l'ingénierie des LPLs. Plusieurs implémentations techniques de la variabilité ont été développées, supportées et évaluées. Pour réaliser la variabilité au niveau du code, les méthodes LPL classique préconisent l'utilisation de l'héritage, des composants, des Frameworks, des aspects ou des techniques génératives. Svahnberg et al. Présentent une taxonomie des techniques de réalisation de la variabilité (Svahnberg et al. 2005). On distingue deux grands groupes, les approches annotatives et de composition, qui peuvent être appliqué soit au niveau du code ou au niveau du modèle.

Les approches annotatives. L'idée est d'annoter des fragments de code dans une base commune de code enlevé afin de générer des variantes (les directives #ifdef et #endif. du préprocesseur C++ pour supprimer conditionnellement des fragments de code avant la compilation en est un exemple typique). Au niveau du modèle, Czarnecki et Antkiewicz (Czarnecki et Antkiewicz 2005 , Czarnecki et Pietroszek 2006), Heidenreich et al.(Heidenreich et al. , 2010), ou encore, (Lauenroth et al. , 2009 , Classen et al 2010b. ; 2011) utilisent des annotations sur les éléments du modèle. Dans ces cas, un modèle global est adapté à un produit particulier par l'activation ou la suppression des éléments de modèle à partir d'une combinaison de fonctions. De même, Saval et al. Utilise le terme *d'élagage* (Saval et al. 2009) et Voelter et Groher introduisent le terme de *variabilité négative* (Voelter et Groher 2007) pour décrire les approches d'annotation basé sur les modèles. Récemment, le Commun Variability Language (CVL) a été proposé pour fournir

une approche générique et distincte pour la modélisation de la variabilité dans tous les modèles définis par un méta modèle au moyen de facilités de Méta Objets (Svendsen et al. 2010).

Approches de composition. Au niveau du code, les caractéristiques sont mises en œuvre séparément dans des modules distincts (fichiers, classes, packages, plug-ins, etc.), qui peuvent être composées dans différentes combinaisons pour générer des variantes. Voelter et al. Décrit la variabilité mis en œuvre avec les approches de composition comme une *variabilité positive* (Voelter et Groher 2007) puisque des éléments variables sont additionnés les uns autres. De nombreuses techniques ont été proposées pour réaliser les approches compositionnelles (Frameworks, aspects (Mezini et Ostermann 2004), raffinement progressifs (Batory et al. 2004), etc.). Dans les LPL basé sur l'ingénierie des modèles, l'idée est que des modèles ou des fragments, chacun correspondant à une caractéristique, sont composés pour obtenir un modèle intégré d'une configuration du feature model. Ainsi, les techniques de modélisation orientée aspect ont été appliquées dans le cadre de l'ingénierie des LPL (Morin, Brice et Barais, Olivier et Jézéquel, Jean-Marc et Ramos, Rodrigo 2007, Morin et al. 2008). Apel et al. Proposent de revoir la technique de superposition et d'analyser sa faisabilité en tant que technique de composition de modèles (Apel et al. 2009). Perrouin et al. Proposent un processus flexible et un outil support, dans lequel un modèle de produit est généré par la fusion de fragments de diagramme de classes UML (Perrouin et al. 2008).

Langages et outils. En outre, plusieurs langues et outils ont été développées pour soutenir les deux approches. CIDE est un outil pour annoter des fragments de code avec des directives #ifdef et #endif. FeatureHouse est l'outil de composition dans lequel des artefacts logiciels écrits en plusieurs langues peuvent être composées, par exemple, le code source, des cas de test, des modèles, de la documentation, des makefiles (Kästner 2010). FeatureHouse et CIDE sont intégrés dans l'environnement de développement de FeatureIDE (Kästner et al. 2009a). FeatureMapper est un outil qui permet de définir les correspondances entre les features et leurs réalisations (Heidenreich et al. 2010 ; 2008). La communauté de la composition du logiciel (qui englobe le développement dirigé par les Aspects (AOSD), développement dirigé par les Features (fOSD), et le développement dirigé par les Composants (CBSD), etc.) propose des mécanismes de compositions différentes mais complémentaires, comme le tissage des aspects, la composition des features ou la composition de composants, pour composer les artefacts du logiciel, au niveau du code ou au niveau du modèle. Les outils sous-jacents peuvent être adaptés pour soutenir le développement LPL.

2.4 Les modèles de caractéristiques (features models)

Au cœur de l'Ingénierie des LPL est la gestion des points communs et des points de variation entre produits. Le modèle de caractéristique (feature model ou plus simplement FM) est une technique de modélisation de la variabilité, qui a généré beaucoup d'intérêt en ingénierie des LPL depuis leur introduction par (Kang et al. 1990) dans la méthode FODA. Les modèles de caractéristique sont actuellement la norme de facto pour représenter la variabilité.

Une caractéristique logicielle (software feature) est une caractéristique distinctive d'un produit logiciel (Cetina, 2009). Ainsi le FM représente l'arrangement hiérarchique des fonctionnalités du système ainsi que les relations entre elles (contraintes structurelles) mais également des dépendances inter-arbre tel que les relations *requière* et *exclu*. Ces relations déterminent les compositions valides de composants. Des informations additionnelle peuvent être incluses tel que la justification de la sélections des caractéristiques.

La figure 2.2 représente le FM d'une ligne de produits simplifiée pour les des systèmes de gestion des bases de données. Nous utiliserons la même notation graphique proposée dans (Czarnecki et Eisenecker 2000). L'exemple de la figure 2.2 montre différentes relations de variabilité tel que : optionnel, obligatoire (indispensable), choix exclusif ou multiple ainsi que des contraintes de sélection tel que l'implication ou l'exclusion entre caractéristiques.

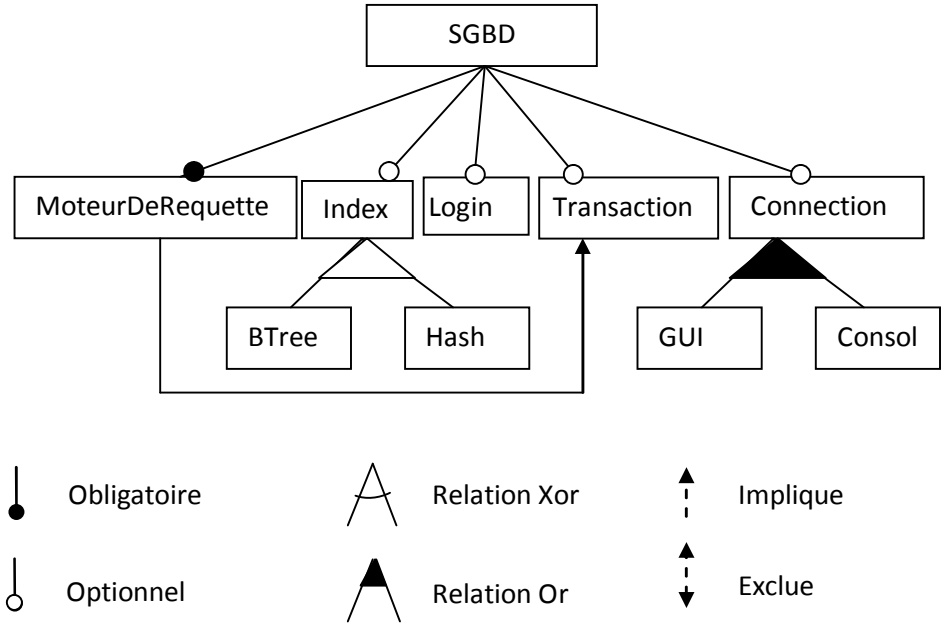


Figure 2.2 le modèle de fonctionnalité de la famille des systèmes de gestion des bases de données. inspiré de (Rosenmuller et al. 2011)

2.4.1 Sémantique des modèles de caractéristiques

Dans une perspective conceptuelle, un modèle de caractéristiques d'un concept décrit un ensemble de combinaisons de caractéristiques valides, chacun représentant une instance de ce concept. Par exemple, le modèle de caractéristiques représente par la figure 2.2, décrit le concept d'un système de gestion de bases de données (SGBD). Un exemple de ce concept est un SGBD qui incorpore un moteur de requêtes et un système de gestion des transitions mais pas une interface de connexion à distance et dont l'index est géré par une table de hachage.

Du point de vue de l'ingénierie des LPL, nous pouvons également considérer que le modèle de caractéristique représenté par la figure 2.2 décrit une ligne de produits logicielle de systèmes de gestion de bases de données où chaque produit de la LPL est un SGBD bien particulier.

L'objectif principal de la hiérarchie du FM est d'organiser les concepts (c.-à-d., Les caractéristiques) de la LPL en plusieurs niveaux de détail croissant. La hiérarchie est donc représentée comme un arbre, la racine étant le concept le plus général (ici: le Système de gestion de bases de données). Les arcs du modèle de caractéristiques modélisent les relations parent-enfant entre les caractéristiques et visent à agréger de nouveaux concepts (c.-à-d., les caractéristiques) ou à spécialiser un concept. Par exemple, un SGBD a cinq caractéristiques enfants (un moteur de requête, une interface de login, un système de gestion des transitions et un système d'indexation, identification). La caractéristique index est à son tour un concept général qui peut être spécialisé (Par exemple, la table de Hachage est un type particulier d'algorithme d'indexation).

Un autre aspect important des modèles de caractéristiques est la variabilité. La variabilité définit les combinaisons de caractéristiques autorisées (également appelées configurations), autrement dit, la variabilité restreint l'ensemble des instances valides d'un concept. La variabilité dans un modèle de caractéristique est exprimée à travers un certain nombre de mécanismes. Lors de la décomposition d'une Sous-caractéristique, les sous-caractéristiques peuvent être facultatives ou obligatoires. Par exemple, un SGBD à besoin d'un moteur de requête et donc la caractéristique $MoteurDeRequette$ est obligatoire. Cependant, un SGBD peut ne pas nécessiter une connexion identifié (ou login, donc la caractéristique $login$ est facultative). Notez qu'une caractéristique est obligatoire ou facultative par rapport a sa caractéristique parent (par exemple, une caractéristique peut être obligatoire et ne pas être inclue dans une configuration dans le cas où sa caractéristique parent n'est pas elle même inclue dans cette configuration). Les caractéristiques peuvent également former des groupes Or ou Xor. Les caractéristiques $Hash$ et $Btree$ forment un groupe Xor - ils sont mutuellement exclusifs de sorte qu'une caractéristique *index* ne peut pas être $Hash$ et $Btree$ en même temps. Les caractéristiques GUI et consol forment un groupe Or : un SGBD peut permettre la

connexion à la BD soit via une GUI, soit via une console ou encore permettre les deux modes en même temps. En outre, des contraintes d'implications et d'exclusions (appelées *règles de composition* dans (Kang et al. 1990)) traversant l'arborescence des caractéristiques peuvent être spécifiées pour exprimer des dépendances plus complexes entre les caractéristiques. Ainsi par exemple, la caractéristique $MoteurDeRequette$ implique la caractéristique Transaction.

Un modèle de caractéristique définit un ensemble de configurations de caractéristiques valides. La validité d'une configuration est déterminée par la sémantique des modèles de caractéristiques, par exemple, dans la figure 2.2, $Hash$ et $Btree$ sont mutuellement exclusives et ne peuvent pas être sélectionnés en même temps. Une configuration est obtenue en sélectionnant les fonctions de manière à respecter les règles:

• si une caractéristique est sélectionnée, son parent doit également être sélectionné (donc un arc entre deux caractéristiques dénote non seulement une relation conceptuelle mais également une dépendance logique);

- si un parent est sélectionné, les caractéristiques suivantes doivent également être sélectionnées: Au moins une sous caractéristique, toutes les sous- caractéristiques obligatoires, exactement une sous- caractéristique dans chacun de ses groupes Xor, au moins une dans chaque groupe Or. les contraintes liées aux caractéristiques et exprimant des *règles de composition* doivent être maintenues;

Il convient enfin de noter que les termes modèle de caractéristiques et diagramme de caractéristiques sont utilisés dans la littérature, généralement pour désigner la même chose. Le mot "diagramme" suggère une représentation graphique d'un modèle de caractéristiques.

Nous considérons qu'un modèle de caractéristiques possède les propriétés d'un modèle ; c'est-à-dire qu'il s'agit d'une abstraction objective d'une LPL, qu'il a une sémantique claire à partir de laquelle plusieurs types d'analyse et de raisonnement automatisés peuvent être effectués (voir la section suivante) et qu'il possède une représentation aussi bien textuelles que graphiques

2.4.2 Formalisme.

Trois notions clés sont définies ci-dessous: diagramme ou modèle de caractéristique, Hiérarchie de caractéristiques et configurations. Tout au long de cette thèse, nous nous appuierons sur ces notions. Plusieurs formalismes et notations ont été préposés dans la littérature. (Schobbens et al. 2007) répertorie la grande majorité des variantes des modèles de caractéristiques. Ils montrent que certaines variantes sont expressément complètes et donc équivalentes en termes d'expressivité et leur donnent une sémantique formelle.

2.4.3 Quelques remarques importantes

Premièrement, nous considérons que la hiérarchie d'un Modèle de caractéristiques est représenté comme un arbre (et non comme un graphe acyclique direct). En particulier, une caractéristique ne peut pas avoir deux parents et ne peut pas appartenir à plus d'un groupe de caractéristiques. Deuxièmement, nous considérons seulement deux types de groupe de caractéristiques (Xor et Or) - nous ne considérons pas les groupe Mutex (comme dans (Elle et al. 2011)) ou les groupes à cardinalités (comme dans (Riebisch et al. 2002, Schobbens et al.2007)) qui définissent un nombre minimum et maximum de caractéristiques à choisir parmi le nombre total de caractéristiques du groupe. Troisièmement, nous considérons uniquement les contraintes d'implication et d'exclusion. Enfin, les caractéristiques sont identifiées de façon unique par leur nom.

Définition 2.1 (Modèle ou Diagramme de caractéristiques, adaptée de (Acher et al, 2011)). Un modèle de caractéristiques $FM = < G, r, E_{MAND}, F_{XOR}, F_{OR}, Impl, Excl >$ est défini comme suit:

• $G = (F, E)$ est un arbre où F est un ensemble fini de caractéristiques et $E \subseteq F \times F$ est un ensemble fini des arêtes (les arêtes représentent une décomposition hiérarchique descendante des caractéristiques, c.-à-d., relations Parent-enfant entre eux);

• $r \in F$ est la caractéristiques racine;

• $E_{MAND} \subseteq E$ est un ensemble d'arêtes qui définit les caractéristiques obligatoires avec leurs parents;

• $F_{XOR} \subseteq \mathcal{P}(F) \times F$ et $F_{OR} \subseteq \mathcal{P}(F) \times F$ définissent des groupes de caractéristiques et sont des ensembles de paires de caractéristiques enfants avec leur caractéristique parent commune. Les caractéristiques enfants sont soit exclusives (Xor-groupes) soit inclusive (Or-groupes);

• les caractéristiques qui ne sont ni obligatoires ni impliquées dans un groupe de caractéristiques sont facultatives;

• une caractéristiques parent doit avoir un seul groupe de caractéristiques, et une caractéristiques doit appartenir à un seul groupe de caractéristiques.

• un ensemble de contraintes *implique* $Impl$ (resp. des contraintes *exclut* $Excl$), chaque contrainte implique (resp. contrainte exclut) est une formule propositionnelle dont la forme est un $A \Longrightarrow B$(resp. $A \Longrightarrow \neg B$) où $A \in F$ et $B \in F$.

Définition 2.2 (Hiérarchie de caractéristiques). La hiérarchie de caractéristiques d'un modèle de caractéristique FM est caractérisée par son arbre $G = (F, E)$ et sa caractéristique racine r.

Définition 2.3 (Configuration). Une configuration d'un modèle de caractéristiques FM est définie comme l'ensemble des caractéristiques sélectionnées. $[\![FM]\!]$ Désigne l'ensemble des configurations valides du modèle de caractéristiques FM et est donc un ensemble d'ensembles de caractéristiques. Une configuration c de FM est définie comme un ensemble de caractéristiques sélectionnées $c = \{f_1, f_2, \ldots, f_m\} \sqsubseteq \mathcal{F}_{FM}$.

Nous pouvons à présent définir la connexion entre le modèle de caractéristiques (resp. Configuration) et une LPL (resp. Produit) comme suit :

Définition 2.4 (LPL, Feature Model). Une ligne de produits logiciels SPL est un ensemble de produits décrit par un modèle de caractéristiques FM. L'ensemble des configurations de FM est notée $\mathcal{F}_{FM}$. Chaque produit de LPL est une combinaison de caractéristiques et correspond à une configuration valide de FM.

2.5 Propriétés des modèles de caractéristiques.

Les modèles de caractéristiques ont des propriétés importantes qui peuvent être extraites automatiquement par des techniques automatisées. En particulier, un modèle de caractéristique peut ne représenter aucune configuration valide (voir Définition 2.5), typiquement en raison de la présence de contraintes inter-arbres - Dans ce cas, le modèle de caractéristiques est dit *nul*

Définition 2.5 (modèle de caractéristiques nul ou *void FM*). Un modèle de caractéristiques est nul (ou insatisfaisable, invalide ou inconsistant) s'il ne représente aucune configuration.

Un modèle de caractéristiques peut comporter des caractéristiques qui ne peuvent être présentes dans aucune configuration valide (voir Définition 2.6), elles sont dites *caractéristiques mortes* (Benavides et al. 2010). Notez que toutes les caractéristiques (y compris la racine) sont mortes dans un modèle de fonction *nul*.

Définition 2.6 (caractéristiques mortes ou *Dead features*). Une caractéristique f de FM est morte si elle ne peut pas faire partie d'aucun des configurations valides de FM. L'ensemble des caractéristiques mortes de FM est noté $Deads\ (FM) = \{\, f \in \mathcal{F}_{FM} \,|\forall c \in [\![\mathrm{FM}]\!], f \notin \mathrm{c}\}$

Les caractéristiques peuvent également faire partie de toutes les configurations valides d'un modèle de caractéristiques.

Définition 2.7 (caractéristiques de base ou *Core features*). Une caractéristique f de FM est une caractéristique de base si elle fait partie de toutes les configurations valides de FM. L'ensemble

des caractéristiques de base de FM est noté $cores(FM) = \{ f \in \mathcal{F}_{FM} \mid \forall c \in [\![\mathrm{FM}]\!], f \in \mathrm{c}\}$

Les modèles de caractéristiques peuvent présenter des anomalies (c'est-à-dire des définitions incorrectes qui suggèrent que l'ensemble des produits décrits par un modèle de caractéristiques est différent de la LPL qu'il décrit) (Trinidad et al. 2008a). Ainsi, les caractéristiques mortes peuvent être considérées comme des anomalies.

Généralement, ces anomalies sont considérées comme une propriété négative d'un modèle de caractéristiques puisqu'elles peuvent facilement diminuer sa maintenabilité ou sa compréhensibilité. Ces anomalies peuvent être détectées automatiquement et des corrections peuvent être proposées et appliquées (Trinidad et al. 2008a, Benavides et al.2010).

2.6 Opérations de raisonnement.

L'analyse automatisée des modèles de caractéristiques consiste à extraire de l'information à partir des modèles de caractéristiques à l'aide des mécanismes automatisés (Benavides et al. 2010).

L'analyse des modèles de caractéristiques est une tâche fastidieuse et délicate, et il n'est pas envisageable de l'effectuer manuellement avec des modèles à grande échelle. C'est un domaine de recherche actif qui profite tant aux praticiens qu'aux chercheurs de la communauté des produits logiciels. Depuis l'introduction de modèles de caractéristiques, la littérature a contribué avec un certain nombre d'opérations d'analyse, d'outils, de paradigmes et d'algorithmes pour soutenir le processus d'analyse.

Mannion (Mannion 2002) a été le premier à proposer l'utilisation des techniques de logique propositionnelle pour raisonner sur les propriétés d'un modèle de caractéristiques. Dans (Batory 2005), la relation qui existe entre le modèle de caractéristiques, la logique propositionnelle et la grammaire a été établie. **Batory** suggère également d'utiliser le système d'entretien de la vérité logique (LTMS) pour déduire des choix pendant le processus de configuration (Batory 2005). **Schobbens et al.** Ont formulé certains opérations et leur complexité (Schobbens et al. 2007). **Benavides et al**. Ont Présenté une revue de la littérature sur les propositions existantes pour l'analyse automatisée des modèles de caractéristiques (Benavides et al. 2010). Des exemples d'analyses prolifèrent et incluent le contrôle de la cohérence et la détections des caractéristiques mortes (Marcilio Mendonca 2009), l'aide interactive au cours de la configuration (Tun et al. 2009, Mendonca et Cowan 2010), ou des modèles de fixation et de configurations (Trinidad et al. 2008a, White et al. 2008, Janota 2010).

Il convient de noter que la plupart des opérations de raisonnement (p. Ex., Satisfiabilité) sont des problèmes computationnels difficiles et sont NP-complets (Schobbens et al. 2007).

2.7 Algorithmes et Automatisation.

Différents types de support automatisé ont été proposés et peuvent être classés comme suit:

• **logique propositionnelle**: les solveurs SAT (pour la satisfaction) ou le diagramme de décision binaire (BDD) prennent une formule propositionnelle comme entrée et permettent de faire des raisonnements sur la formule (Validité, modèles, etc.).

• **programmation par contraintes**: un problème de satisfaction de contraintes (CSP) consiste en un ensemble de variables, un ensemble de domaines finis pour ces variables et un ensemble de contraintes limitant les valeurs des variables. Un CSP est résolu en trouvant des états (valeurs pour les variables) dont toutes les contraintes sont satisfaites. Contrairement aux formules propositionnelles, les solveurs CSP peuvent traiter non seulement des valeurs binaires (vrai et faux), mais aussi des valeurs numériques telles que des nombres entiers ou les intervalles.

• logique de description (DL): les logique de description sont une famille de langages de représentation de connaissances qui permettent un raisonnement dans les domaines de la connaissance en utilisant des raisonneurs logiques spécifiques. Un problème décrit en termes de logique de description est habituellement composé d'un ensemble de concepts (Par exemple, les classes), un ensemble de rôles (par exemple, les propriétés ou les relations) et un ensemble d'individus (Exemples). Un raisonneur de logique de description est un progiciel qui prend en entrée un problème décrit en DL et permet la vérification de la cohérence et de la correction ainsi que d'autres opérations de raisonnement.

• D'autres contributions non classées dans les précédents groupes proposant des solutions, des algorithmes ou des paradigmes existent également.

La plupart des techniques existantes s'appuient sur des outils de logiques propositionnels pour raisonner sur les modèles de caractéristiques propositionnelles (par exemple, voir (Czarnecki et Wasowski 2007, Mendonca et al.2008, Thüm et al. 2009, Marcilio Mendonca 2009, Janota 2010)). Benavides et al. Montrent que les solveurs CSP ou les solveurs DL peuvent également être utilisés, mais surtout pour d'autres extensions des modèles de caractéristiques (Par exemple, les attributs associés aux caractéristiques) (Benavides et al. 2010).

2.7.1 Les modèles de caractéristiques non propositionnels

Certaines extensions de modèles de caractéristiques ont été proposées (par exemple, les attributs associes aux caractéristiques (Benavides et al 2005), les modèles de caractéristiques à base de cardinalité (Czarnecki et al. 2005b)). Très tôt, Kang et al. Ont utilise un exemple d'attributs dans (Kang et al. 1990) tandis que Czarnecki et al. Utilisent le terme «attribut de caractéristique" dans

(Czarnecki et al. 2002). Cependant, comme rapporté dans (Benavides et al. 2010), la grande majorité de la recherche sur la modélisation de caractéristiques se fait sur les modèles de caractéristiques propositionnels « de base» (Czarnecki et al. 2006, Czarnecki et Wasowski 2007), Dans cette thèse, nous nous concentrons exclusivement sur ce type de formalisme. Dans le reste du présent document, le terme modèle de caractéristiques se réfère implicitement au modèle de caractéristiques propositionnel.

2.8 Utilisation des modèles de caractéristiques

Dans l'ingénierie LPL, les modèles de caractéristiques ont été employés dans des contextes variés, à différents niveaux d'abstraction et à des fins multiples. Ceci peu s'expliquée en partie par la vairées de notions qui peuvent être associées aux notions de caractéristiques et de variabilité Dans cette Section suivante, nous examinons les travaux pertinents, pour lesquels l'utilisation de modèles de caractéristiques est le principal intérêt.

2.8.1 Caractéristique et variabilité

Les modèles de caractéristiques visent à décrire la variabilité d'un LPL en termes de caractéristiques: Variabilité et caractéristique sont cependant des concepts abstraits pour lesquels il existe un grand nombre d'interprétations. Cela impact naturellement la façon dont les modèles de caractéristiques sont utilisés.

Qu'est-ce qu'une caractéristique? En raison de la diversité de la recherche en génie logiciel, il existe plusieurs définitions d'une caractéristique (Classen et al. 2008, Apel et Kästner 2009), par exemple

(Ordonné du plus abstrait au plus technique):

• (Kang et al. 1990): «un aspect qualité ou caractère d'un ou de plusieurs systèmes logiciels qui soit éminent ou distinctif et visible par l'utilisateur"

• (Kang et al. 1998): «une abstraction fonctionnelle distinctement identifiable qui doit être Implémentée, testée, fournie et entretenue "

• (Czarnecki et Eisenecker 2000): «une caractéristique distincte d'un concept (par exemple, Composant du Système, etc.) Qui est pertinente pour certains intervenants du concept "

• (Bosch 2000): "une unité logique de comportement spécifiée par un ensemble de propriétés et exigences fonctionnelles ou non fonctionnelles"

• (Chen et al. 2005): «une caractéristique du produit du point de vue l'utilisateur ou du client, qui consiste fondamentalement en un ensemble cohérent d'exigences individuelles "

• (Batory et al. 2003): "une caractéristique de produit qui est utilisé dans les programmes distinctifs du sein d'une famille de programmes connexes "

• (Zave et Jackson 1997): "une unité optionnelle ou incrémentielle de fonctionnalité"

• (Batory 2005): "une augmentation de la fonctionnalité du programme"

Variabilité: Catégories, Niveaux, Classification. La variabilité est une préoccupation du processus d'ingénierie LPL qui apparaît à chaque phase de développement logiciel. Par exemple lors de l'identification des variabilités, leur origine peut être identifiée de différentes perspectives. Dans (Bachmann and Bass 2001) les auteurs proposent de classer la variabilité en catégories: Variabilité de la fonction (une fonction particulière peut exister dans certains produits mais pas dans d'autres); Variabilité des données (une structure de données particulière peut être trouvée dans produit mais pas dans un autre); Variabilité des flux de contrôle (une interaction particulière peut se produire dans certains produits et pas dans d'autres); La variabilité de la technologie (OS, hardware, interface utilisateur, etc. peuvent varier d'un produit à l'autre); Variabilité des Objectifs de qualité (des dimensions qualitatives telles que la performance ou la sécurité peuvent être spécifique pour un produit), etc. (Svahnberg et al. 2005) ont défini cinq niveaux de variabilité (au niveau de la LPL, du , du composant, du sous-composant et du code) ou des problèmes de conception peuvent apparaitre. Une autre classification, basée sur la distinction entre la Variabilité essentielle et la variabilité technique, a été proposé dans (Halmans et Pohl, 2003). La Variabilité essentielle se réfère au point de vue du client (qui est aussi appelé la variabilité externe dans (Pohl et al.2005) ou variabilité de la ligne de produits dans (Metzger et al. 2007)). La variabilité technique (également appelée variabilité interne (Pohl et al. 2005) et la variabilité logicielle (Metzger et al. 2007)) Se réfère au point de vue des ingénieurs de la LP qui s'intéressent principalement aux détails de mise en œuvre (c.-à-d., Les points de variation, les variantes et les temps de liaison pour mettre en œuvre la variabilité).

Impact sur la modélisation des caractéristiques. La pléthore de définitions des caractéristiques et de variabilité suggère que les modèles de caractéristiques peuvent être utilisés à différents stades du développement de la LPL. Dès le début (par exemple, à l'étape d'identification des exigences) à la modélisation de plates-formes et des composants, les modèles caractéristiques peuvent être utilises sur n'importe quel type d'artefacts (code, documentation, modèles) et à n'importe quel niveau d'abstraction. Ils peuvent jouer un rôle central dans la gestion de la variabilité et de la dérivation des LPL (par exemple, voir (Czarnecki et Antkiewicz 2005, Ziadi et Jézéquel 2006, Sanchez et al. 2008, Voelter et Groher 2007, Heidenreich et al. 2010)). Dans cette thèse, les modèles de caractéristiques sont abordés dans une perspective générale et ne se limitent pas à une

étape de développement. En conséquence, un modèle de caractéristiques peut tout aussi bien décrire une famille de programmes, une famille d'exigences ou une famille de modèles.

2.9 Les outils de modélisations des caractéristiques

Les premiers outils de modélisation des modèles de caractéristiques sont apparus dés 2004. La plupart de ces outils prennent en charge la création de modèles de caractéristiques mais également un support au processus de configuration et de configuration partielle dans une démarche de configuration par étapes (Czarnecki et al. 2005b). L'outil. FMP (Czarnecki et al. 2004) a été la le point de départ d'un tel effort. FMP fournit des vues arborescentes pour créer des modèles de caractéristiques, les configurer ou les configurer partiellement. LPLOT est un autre exemple d'outil basé sur le Web offrant un éditeur de modèle de caractéristiques sous la forme d'une arborescence, un éditeur de configuration avec la fonction de propagation de décisions et l'analyse automatisée du modèles ainsi qu'une bibliothèque des modèles type (Mendonca et al. 2009a ; b). FAMA est un Framework Java qui met l'accent sur la comparaison des différents solveurs pour l'analyse automatisée des modèles de fonction (FaMa 2008, Trinidad et al. 2008b). Deux outils commerciaux intègrent la modélisation des caractéristiques et le processus de configuration ; pur :: variantes (pures :: variantes 2006, Beuche 2008) et Gears (BigLever - Gears 2006) et fournissent des extensions à plusieurs IDEs tels que Eclipse et Netbeans. FeatureIDE est un Framework basé sur Eclipse pour le développement de logiciels bases sur les modèles de fonctionnalités (Kästner et al. 2009a ; b). L'objectif principal de FeatureIDE est découvrir l'ensemble du processus de développement et d'intégrer des outils pour la mise en œuvre de LPL dans un environnement de développement intégré.

2.10 Conclusion

Les modèles de caractéristiques sont un formalisme fondamental dans l'ingénierie des LPL. Ces modèles structurent hiérarchiquement les caractéristiques et définissent de manière compacte l'ensemble des combinaisons légales de fonctionnalités, où chaque combinaison correspond à un produit dans une LPL. Dans ce chapitre, nous avons introduit l'idée et la sémantique des modèles de caractéristiques. Bien qu'il existe une panoplie de formalismes étendus de modèles de caractéristiques, (par exemple, des formalismes qui associent des attributs aux caractéristiques), nous avons choisi les modèles de caractéristiques classique (aussi appelé de base) car il s'agit du

formalisme le plus répondu dans l'industrie (Thüm et al. 2009)(Benavides et al.2010). Nous avons mis en évidence un certain nombre de techniques (automatisables) qui ont été développées pour faciliter l'analyse et la manipulation des modèles de caractéristiques. Nous avons également revu plusieurs travaux autour de l'utilisation des modèles de caractéristiques aujourd'hui exploités dans plusieurs domaines et utilisé dans une variété de contextes, à différents niveaux d'abstraction et à différentes fins.

Chapi

III

Informatique autonomes et lignes de produits logiciels dynamiques

3.1 Introduction

Un système autonome est un système logiciel capable de s'autogérer sans (ou avec très peu) l'intervention d'agents humains. Récemment, plusieurs travaux ont exploré l'idée d'exploiter le paradigme des LPL afin de réaliser des systèmes autonomes. Ceci est connu sous le nom de ligne dynamique de produits logiciels (DLPL) (Bencomo et al. 2010). Plus spécifiquement, une ligne dynamique de produits logiciels utilise les modèles de variabilité pour représenter et raisonner sur les variantes (ou configurations) du système à l'exécution afin de bénéficier des techniques et des outils existants à des fins d'adaptation à l'exécution. Dans ce chapitre les deux concepts centraux à notre travaille, à savoir, l'informatique autonome et les lignes de produit logiciel dynamiques, sont arborés. La terminologie adoptée est largement revisitée et une revue des principales techniques existantes est menée afin de permettre d'identifier les défis majeurs rencontrés dans ce domaine en pleine essor.

3.2 L'informatique autonome

En octobre 2001, IBM a publié un manifeste (IBM, 2001) décrivant une nouvelle vision de l'informatique autonome. Le but est de faire face à la complexité des systèmes logiciels en rendant les systèmes autonomes. Cependant, L'un des points soulevé fut que les systèmes doivent devenir encore plus complexes pour y parvenir. La complexité, doit donc être intégrée dans l'infrastructure du système, qui peut à son tour être automatisée. La similarité de l'approche décrite avec le système nerveux autonome du corps, qui le soulage du contrôle de base de notre conscience, a donné naissance au terme d'informatique (ou calcul) autonome (Autonomic Computing).

3.2.1 Définition

Inspiré par la biologie, l'informatique autonome a évolué comme une discipline pour créer des systèmes logiciels et des applications qui s'autogèrent dans le but de surmonter la complexité et l'incapacité à maintenir efficacement les systèmes actuels et à venir. À cette fin, les efforts au niveau de l'automatisation couvrent tous les domaines de l'informatique et démontrent déjà leur faisabilité et leur valeur.

En 2001, IBM a proposé le concept de l'informatique autonome. En fait, les systèmes informatiques complexes sont comparés au corps humain, un système complexe, mais doté d'un système nerveux

autonome qui s'occupe de la plupart des fonctions corporelles, soulageant ainsi notre pensé consciente de la tâche de coordonner toutes nos fonctions corporelles. IBM a suggéré que les systèmes informatiques complexes doivent être autonomes, c'est-à-dire devrait être capable de prendre en charge indépendamment les tâches de maintenance et d'optimisation régulières, réduisant ainsi la charge de travail des administrateurs système. IBM a également définit les quatre propriétés d'un système autogéré: auto-configuration, auto-optimisation, auto-guérison et autoprotection. Comme l'a déclaré Alan Ganek qui est l'un des responsables de l'informatique Autonome chez IBM:

«L'informatique autonome est la capacité des systèmes à mieux s'autogérer. Le terme autonomique vient du système nerveux autonome, qui contrôle de nombreux organes et muscles dans le corps humain. Habituellement, nous ne sommes pas conscients de son fonctionnement, car il fonctionne d'une manière involontaire, réflexive. Par exemple, nous ne remarquons pas lorsque notre cœur bat plus vite ou nos vaisseaux sanguins changent de taille en réponse à la température, la posture, la nourriture, ou d'autres changements auxquels nous sommes exposés." (Cetina et al. 2010)

3.2.2 Propriétés du calcul autonome

Les principales propriétés de l'informatique autonome telles que présentées par IBM sont l'auto-configuration, l'auto-optimisation, l'auto-guérison et l'autoprotection. Voici une brève description de ces propriétés (pour plus d'informations, voir (Kephart et al. 2003) (Bantz et al. 2003)):

• **Auto-configuration**. Un système informatique autonome se configure selon des objectifs de haut niveau, c'est-à-dire en spécifiant ce qui est désiré, et pas nécessairement la manière de le réaliser. Cela peut signifier être capable de s'installer et de se configurer en fonction des besoins de la plate-forme et de l'utilisateur.

• **Auto-optimisation**. Un système informatique autonome optimise son utilisation des ressources. Il peut décider d'initier une modification du système de façon proactive (en opposition au comportement réactif) afin d'améliorer la performance ou la qualité du service.

• **Auto-guérison**. Un système informatique autonome détecte et diagnostique les problèmes. Les types de problèmes détectés peuvent être interprétés de manière générale: ils peuvent être aussi bas-niveau qu'une erreur de bit dans une puce de mémoire (défaillance matérielle) ou au contraire de très haut niveau comme une entrée erronée dans un service d'annuaire (problème logiciel) (Paulson, 2002). Si possible, il devrait tenter de résoudre le problème, par exemple en basculant vers un composant redondant ou en téléchargeant et en installant des mises à jour logicielles. Cependant, il est important que le processus de guérison ne porte pas atteinte au système, par exemple par

l'introduction de nouveaux bugs ou la perte de paramètres vitaux du système. La tolérance aux fautes est un aspect important de l'auto-guérison. Autrement dit, un système autonome est dit réactif à des échecs ou des signes précoces d'un échec possible.

• **Autoprotection**. Un système autonome se protège des attaques malveillantes mais aussi des utilisateurs finaux qui effectuent par inadvertance des modifications de logiciels, par ex. En supprimant un fichier important. Le système s'accorde de manière autonome pour assurer la sécurité, la confidentialité et la protection des données. La sécurité est un aspect important de l'autoprotection, pas seulement dans le logiciel, mais aussi dans le matériel. Un système peut également être en mesure d'anticiper les violations de sécurité et de les empêcher de se produire en premier lieu. Ainsi, le système autonome présente des caractéristiques proactives.

Les concepts sous-jacents aux propriétés auto-* n'étaient pas entièrement nouveaux dans l'initiative d'informatique autonome d'IBM. Par exemple, un optimiseur de requêtes, un gestionnaire de ressources ou un logiciel de routage dans les systèmes de gestion de bases de données (SGBD), les systèmes d'exploitation et les réseaux, respectivement, permettent à tous ces systèmes de se gérer eux-mêmes. Cependant, la communauté des systèmes d'autogestion s'accorde sur le fait que le terme «informatique autonome» n'est pas utilisé pour décrire ces systèmes, mais ceux dans lesquels la planification des requêtes, la gestion des ressources ou la décision d'acheminement change pour refléter le contexte environnemental actuel ; Reflétant le dynamisme du système. Autrement dit, le plan de requête SGBD change à mesure que la requête est en cours d'exécution.

En outre, d'autres systèmes adaptatifs ont contenu certains éléments des propriétés ci-dessus pendant un certain temps, en particulier pour fournir l'auto-optimisation. Les premiers exemples de ce phénomène peuvent être observés dans les systèmes multimédias en streaming où le codec du flux change en fonction des fluctuations de la bande passante du réseau, l'objectif étant de maintenir la qualité musicale ou vidéo la plus élevée possible, (Maccann et al. 1998) (Lippmann et al. 1999). Cependant, de plus en plus, la communauté autonome identifie un système comme autonome s'il présente plus d'une des propriétés d'autogestion décrites plus haut (Ganek et al. 2006).

3.2.3 La boucle autonome MAPE-K

Pour obtenir un calcul autonome, IBM a suggéré un modèle de référence pour les boucles de contrôle automatique (IBM, 2003) qui est parfois appelé la boucle MAPE-K (Monitor, Analyse, Plan, Exécute, Knowledge) et est représenté dans la Figure 3.1. Ce modèle est de plus en plus utilisé pour communiquer les aspects architecturaux des systèmes autonomes.

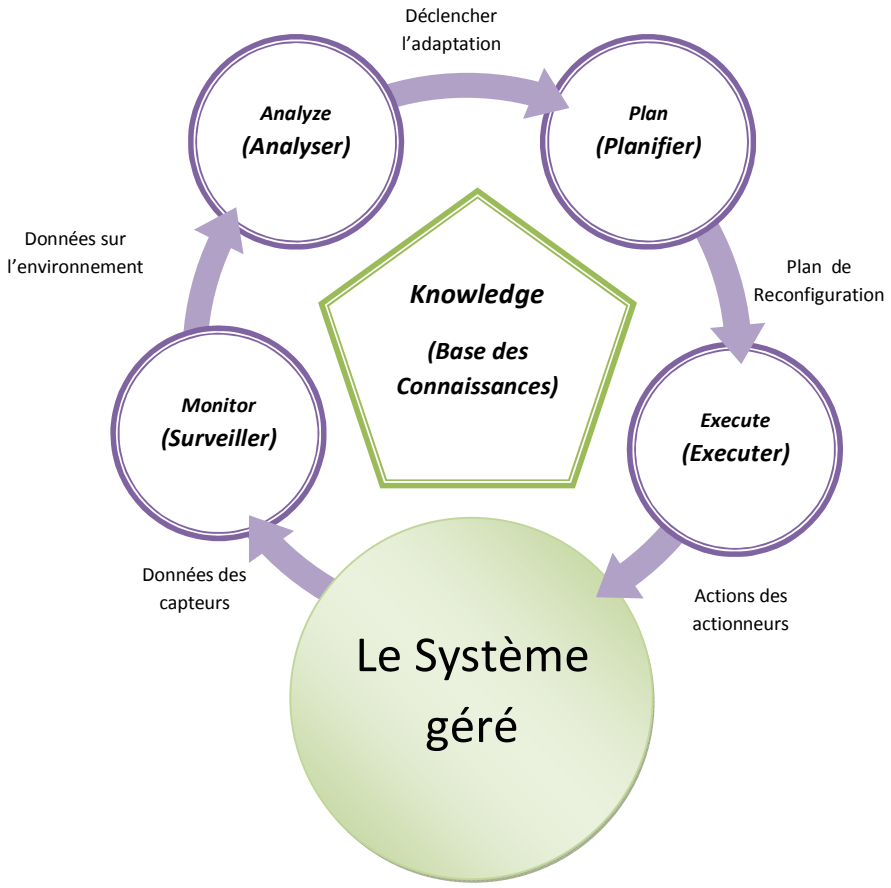


Figure 3.1 le modèle MAPE-K d'IBM et la terminologie correspondante

La boucle autonome MAPE-K est semblable au modèle d'agent générique proposé par Russel et Norvig (Russel et al.2003), et est probablement inspiré par celui-ci, dans lequel un agent intelligent perçoit son environnement par des capteurs et utilise ces percepts pour déterminer les actions à exécuter sur l'environnement.

Dans la boucle autonome MAPE-K, l'élément géré représente toute ressource logicielle ou matérielle à laquelle on attribue un comportement autonome en l'associant à un gestionnaire autonome. Ainsi, l'élément géré peut être par exemple un serveur web ou une base de données, un composant logiciel spécifique dans une application (par exemple l'optimiseur de requêtes dans une base de données), le système d'exploitation, un groupe de machines dans un environnement de grille, une pile de disques durs, Un réseau câblé ou sans fil, une CPU, une imprimante, etc.

Les capteurs, souvent appelés sondes ou jauges, recueillent des informations sur l'élément géré. Pour un serveur Web, cela peut inclure le temps de réponse aux demandes des clients, l'utilisation du réseau et du disque, l'utilisation du processeur et de la mémoire. Une grande partie de la recherche est impliquée dans la surveillance des serveurs (Roblee et al. 2005), (Strickland, et al. 2005), (Xu et al. 2005), (Sterritt, et al. 2005), (Diao et al. 2005).

Les actionneurs apportent des modifications à l'élément géré. Le changement peut être à plusieurs niveaux, par exemple L'ajout ou la suppression de serveurs à un cluster de serveurs Web (Schmerl

et al. 2002), ou à un niveau plus fins, telle que la modification des paramètres de configuration dans un serveur Web (Sterritt, et al. 2005) (Pilgriml et al. 2002)

La collecte d'information de l'environnement (surveillance)

La composante de surveillance de la boucle MAPE-K implique la capture des propriétés de l'environnement (physique ou virtuel, par exemple un réseau) qui sont importantes pour les propriétés auto-* du système. Les composants logiciels ou matériels utilisés pour effectuer la surveillance sont appelés capteurs. Par exemple, la latence du réseau et la bande passante mesurent les performances des serveurs Web, tandis que l'indexation des bases de données et l'optimisation des requêtes affectent le temps de réponse d'un SGBD, qui peut être surveillé. Le gestionnaire autonome requiert des données contrôlées appropriées pour connaître les défaillances ou les performances sous-optimales de l'élément autonome et effectuer les changements appropriés.
Les types de propriétés surveillées et les capteurs utilisés sont souvent spécifiques à une application, de même que les actionneurs utilisés pour exécuter des modifications à l'élément géré sont également spécifiques à l'application. Les systèmes informatiques autonomes sont basés sur deux types de surveillance comme suit.

- Surveillance passive. Les systèmes de surveillance passive ne nécessitent pas l'ajout d'un code de mesure dans le système, mais plutôt l'interaction réelle du système en marche. Par exemple, des outils de surveillance passive existent pour la plupart des systèmes d'exploitation, par ex. Windows 2000 / XP renvoie la mémoire et les statistiques d'utilisation des processeurs.
- Surveillance active. Une surveillance active requiert une ingénierie du logiciel à un certain niveau, par ex. Modifier et ajouter du code à la mise en œuvre de l'application ou du système d'exploitation, pour capturer des appels de fonction ou système. Cela peut souvent être dans une certaine mesure automatisé. Par exemple, ProbeMeister peut insérer des sondes dans le bytecode Java compilé.

Des travaux plus récents ont examiné comment décider quel sous-ensemble des nombreuses métriques de performance collectées à partir d'un environnement dynamique peut être obtenu à partir des nombreux outils de performance disponibles (par exemple, dproc). Il est intéressant de noter qu'un petit sous-ensemble de mesures fournit 90% de précision de classification des applications (Zhang., et al. 2006). Agarwala et al. (Agarwala et al. 2006) proposent QMON, un moniteur autonome qui adapte sa fréquence de surveillance et ses volumes de données afin de

minimiser les frais généraux de surveillance continue tout en maximisant l'utilité des données de performance. Autrement dit, un moniteur autonome pour les systèmes autonomes.

Planification

L'aspect planification de la boucle autonome implique de prendre en compte les données de surveillance des capteurs pour produire une série de modifications à effectuer sur l'élément géré sur la base d'un ensemble d'objectifs de haut niveaux. Ces objectifs sont parfois exprimés à l'aide des politiques de la condition d'événement (ECA), de politiques d'objectifs ou des politiques de fonction d'utilité (Kephart et al. 2004).

Les politiques basées sur les ECA prennent la forme «lorsque l'événement se produit et qu'une condition est vérifiée, alors exécuter une action». Par exemple,. Lorsque 95% du temps de réponse des serveurs Web dépasse 2s et qu'il ya des ressources disponibles, alors augmenter le nombre de serveurs Web actifs. Ils ont été intensément étudiés pour la gestion des systèmes distribués. Des exemples de ces langages et applications orientés politique dans l'informatique autonome incluent (Lymberopoulos et al. 2003) (Lobo et al. 1999) (Agarwal et al. 2005) (Batra et al. 2002) (Lutfiyya et al. 2001) (Ponnappan et al. 2002). Une difficulté avec les politiques d'ECA est que lorsqu'un certain nombre de politiques sont spécifiées, les conflits entre les politiques peuvent se produire qui sont difficiles à détecter. Par exemple, lorsqu'un système à plusieurs niveaux (par exemple, les niveaux de serveur Web et d'application) requiert une quantité accrue de ressources, mais que les ressources disponibles ne peuvent pas répondre aux demandes de tous les niveaux, un conflit survient. Dans un tel cas, il n'est pas clair comment le système doit réagir, et un mécanisme de résolution de conflit supplémentaire est nécessaire, par ex. Donnant une priorité plus élevée au serveur Web. En conséquence, une quantité considérable de recherches sur le règlement des conflits est apparue (Lupu, et al. 1999), (Matsuda, et al. 2005) (Gupta., et al. 2005). Par ailleurs, une complication supplémentaire est que le conflit ne devienne apparent qu'au moment de l'exécution.

- **Les politiques d'objectifs** sont plus abstraites dans la mesure où elles précisent des critères qui caractérisent les états souhaitables, mais laissent au système la tâche de trouver comment y parvenir. Par exemple, nous pourrions spécifier que le temps de réponse du serveur Web doit être inférieur à 2s, alors que celui du serveur d'applications est inférieur à 1s. Le gestionnaire autonome utilise alors des règles internes (c'est-à-dire connaissances) pour ajouter ou supprimer des ressources si nécessaire pour atteindre l'état souhaitable. Les politiques d'objectifs requièrent une planification de la part du

gestionnaire autonome et sont donc plus gourmandes en ressources que les politiques de la CEA. Cependant, ils souffrent toujours du problème que tous les Etats sont classés comme souhaitables ou indésirables. Ainsi, lorsqu'un état souhaitable ne peut être atteint, le système ne sait pas lequel des états indésirables est le moins mauvais.

- **Les fonctions utilitaires** résolvent le problème ci-dessus en définissant un niveau quantitatif de désirabilité pour chaque état. Une fonction d'utilité prend comme entrée un certain nombre de paramètres et répond avec l'opportunité de cet état. Ainsi, suivant notre exemple précédent, la fonction d'utilité pourrait prendre comme entrée le temps de réponse pour les serveurs Web et d'application et renvoyer l'utilité de chaque combinaison de temps de réponse du serveur Web et du serveur d'applications. Ainsi, lorsque les ressources sont insuffisantes, la partition la plus souhaitable des ressources disponibles parmi les serveurs Web et d'applications peut être trouvée. Le problème majeur des fonctions d'utilité est qu'elles peuvent être extrêmement difficiles à définir, car tout aspect qui influence la décision par la fonction d'utilité doit être quantifié. Des recherches sont en cours sur l'utilisation des fonctions d'utilité, en particulier dans l'allocation automatique des ressources (Tesauro., et al. 2004) ou l'adaptation du streaming de données aux conditions du réseau (Bhatti., et al. 1999).

Cependant, l'application de cette approche de manière sans état, c'est-à-dire où le gestionnaire autonome ne conserve aucune information sur l'état de l'élément géré et s'appuie uniquement sur les données de capteur actuelles pour décider d'exécuter un plan d'adaptation, est très limitée. En effet, il est bien meilleur pour le gestionnaire autonome de conserver des informations sur l'état de l'élément géré dans un modèle de contexte qui peut être mis à jour progressivement à partir de données de capteurs et raisonné.

En ce qui concerne l'information sur l'état que le gestionnaire autonome doit garder au sujet de l'élément géré, de nombreuses recherches ont examiné des approches fondées sur des modèles. Dans ces approches, une certaine forme de modèle de l'ensemble du système géré est utilisée par le gestionnaire autonome. Le modèle peut également représenter un aspect de l'environnement d'exploitation dans lequel les éléments gérés sont déployés, où l'environnement d'exploitation peut être compris comme toute propriété observable (par les capteurs) qui peut avoir une incidence sur son exécution, p. Ex. Les entrées de l'utilisateur, les périphériques matériels, les propriétés de connexion réseau.

Le modèle est mis à jour à l'aide des données du capteur et est utilisé pour raisonner sur le système géré pour planifier les adaptations. Le modèle architectural peut être utilisé pour vérifier que

l'intégrité du système est préservée lors de l'application d'une adaptation, c'est-à-dire que nous pouvons garantir que le système continuera à fonctionner correctement après l'exécution de l'adaptation prévue (Oreizy, et al. 1999). En effet, les modifications sont planifiées et appliquées d'abord au modèle, qui montrera l'état du système résultant de l'adaptation, y compris toute violation des contraintes ou des exigences du système présent dans le modèle. Si le nouvel état du système est acceptable, le plan peut alors être effectué sur le système géré, assurant ainsi que le modèle et sa mise en œuvre sont cohérents les uns par rapport aux autres.
En pratique cependant, il ya toujours un délai entre le moment où un changement se produit dans le système géré et cette modification est appliquée au modèle. En effet, si le retard est suffisamment élevé et que le système change fréquemment, un plan d'adaptation peut être créé et envoyé pour exécution avec l'hypothèse que le système réel se trouve dans un état particulier, par ex. un serveur web surchargé, alors qu'en réalité le système a déjà changé entre-temps et ne nécessite plus cette adaptation (ou nécessite un plan d'adaptation différent) (Garlan et al. 2001).

La base de Connaissances

La connaissance d'un système autonome peut provenir de sources diverses depuis les experts humains (dans les systèmes statiques basés sur les politiques (Bougaev et al. 2005)) jusqu'aux fichiers logs qui sauvegardent les données accumulées à partir de sondes traçant le fonctionnement quotidien d'un système pour observer son comportement, ce qui est utilisé pour former des modèles prédictifs (Salahshour et al. 2005) (Shivam et al. 2006). Cette section énumère quelques-unes des principales méthodes utilisées pour représenter la Connaissance dans les systèmes autonomes.

- **Le concept d'utilité**. L'utilité est une mesure abstraite d'utilité ou de pertinence pour un utilisateur par exemple. Typiquement, une opération de système exprime son utilité comme une mesure de choses telles que la quantité de ressources disponibles pour l'utilisateur (ou les programmes d'application utilisateur), et la qualité, la fiabilité ou la précision de cette ressource etc. Par exemple dans un système de traitement d'événement allouant des ressources matérielles pour les utilisateurs souhaitant exécuter des transactions, l'utilité sera fonction du taux alloué, de la latence admissible et du nombre de consommateurs, … . Un autre exemple est dans un système de provisionnement de ressources où l'utilité est dérivée du coût de la redistribution des charges de travail une fois attribuée ou de la consommation d'énergie en tant que portion des coûts d'exploitation (Osogami et al. 2005)(Sharma et al. 2003).

- **Apprentissage par renforcement**. L'apprentissage par renforcement est utilisé pour établir des politiques issues de l'observation des actions de gestion. A son niveau le plus basique, l'apprentissage par renforcement apprend les politiques en essayant des actions dans différents états du système et en examinant les conséquences de chaque action (Sutton et al. 1998). L'avantage de l'apprentissage par renforcement est qu'il ne nécessite pas de modèle explicite du système géré, d'où son utilisation dans le calcul autonome (Fenson et al. 2004) (Dowling et al. 2006). Cependant, il souffre d'une mauvaise mise à l'échelle si l'on souhaite représenter de grands espaces d'état, ce qui a également un impact sur son temps d'apprentissage. A cet effet, un certain nombre de modèles hybrides ont été proposés qui accélèrent la formation ou introduisent une connaissance du domaine pour réduire l'espace d'état, telle que proposé dans. (Tesauro et al. 2006)(Whiteson et al. 2006).
- Les techniques bayésiennes. Des techniques probabilistes ont été utilisées dans la littérature d'autogestion pour fournir le moyen de choisir une solution adéquate parmi un certain nombre de services ou d'algorithmes. Par exemple, Guo (Guo et al. 2003) montre comment les réseaux Bayesiens (ou Bayesian Networks BNs) sont utilisés dans la sélection de l'algorithme autonome pour trouver le meilleur algorithme. Par ailleurs, la classification et la rétroaction sensibles au coût ont été utilisées pour attribuer des coûts aux équations d'auto-guérison pour remédier aux défaillances (Nguye et al. 2003)].

En utilisant la connaissance de la configuration du système, un composant de diagnostic de problème (par exemple basé sur un réseau bayésien) analyserait les informations des fichiers journaux, éventuellement complétées par les données des moniteurs supplémentaires qu'il a demandées. Le système comparerait alors le diagnostic avec des correctifs logiciels connus (ou alertera un programmeur humain s'il n'y en a pas), installe le correctif approprié et recommence le teste.

Toutes les techniques présentées dans ce chapitre et dans les sections ci-dessus contribuent à la mise en place de gestionnaires autonomes sophistiqués pour les éléments gérés. En fin de compte, la distinction entre le gestionnaire autonome et l'élément géré peut être purement conceptuelle plutôt qu'architecturale, ou elle peut se fondre, laissant des éléments autonomes pleinement intégrés avec des comportements et des interfaces bien définis, mais aussi avec peu de contraintes sur leur structure interne.

3.3 Les lignes de produits logiciels dynamiques

Le principal objectif du paradigme des lignes de produits logiciels est de produire des logiciels, avec des coûts et des temps de mise sur le marché qui soient réduits par une réutilisation intensive des points communs et une gestion appropriée de la variabilité. Les produits sont généralement produits en sélectionnant les éléments qui font partie d'un produit et en enlevant ceux qui n'en font pas partie. Pour prendre cette décision, les fonctionnalités sont sélectionnées et / ou désélectionnées à différents moments (appelés temps de liaison ou *binding time*). Après la production, aucune activité automatisée n'est spécifiée dans l'ingénierie des lignes de produits « classiques » pour maintenir un produit en relation avec la LPL. Par conséquence, il ne pourra pas bénéficie des mises à jour éventuelles des fonctionnalités.

Dans les environnements modernes de calcul et de réseau, un degré élevé d'adaptabilité des systèmes logiciels est exigé. Les environnements informatiques, les besoins des utilisateurs et les mécanismes d'interface entre les périphériques logiciels et matériels tels que les capteurs peuvent changer de façon dynamique pendant l'exécution. Par conséquent, dans ces types d'environnements dynamiques, l'approche des LPL doit être changée d'une perspective statique à une perspective dynamique, où des systèmes capables de modifier leur propre comportement par rapport à des changements dans leur environnement opérationnel sont obtenus en relançant dynamiquement des points de variation au moment de l'exécution. C'est l'idée de lignes de produits logiciels dynamiques (LDPL) (Hallsteinsen et al. 2008) (Gomaa et al. 03).

Le développement de DLPL vise principalement à produire des produits configurables (Van der Linden, 2004) dont l'autonomie leur permet de se reconfigurer et de bénéficier d'une mise à jour d'une manière continue. Dans une LDPL, un produit configurable (PCo) est produit à partir d'une ligne de produits de façon similaire à la norme LPL. Cependant, la capacité de reconfiguration implique l'utilisation de trois artefacts pour le contrôler: L'analyseur, le décideur (planificateur) et le reconfigurateur. L'analyseur est chargé de capturer toutes les informations dans son environnement qui suggère une modification de ces informations à partir de capteurs externes ou même des utilisateurs. Le décideur doit connaître l'ensemble de la structure d'un PCo pour prendre une décision sur les caractéristiques qui doivent être activées et désactivées. Le reconfigurateur est responsable de l'exécution de la décision en utilisant les techniques de liaison à l'exécution standard du paradigme LPL. Un PCo peut être considéré comme une extension des produits LPL traditionnels où il n'y a pas de caractéristiques liées, mais plutôt un décideur et un reconfigurateur. Ainsi, les caractéristiques sont liées au moment de l'exécution. En conséquence, de nouvelles

caractéristiques peuvent être ajoutées à un produit existant ou même les caractéristiques existantes peuvent être mises à jour au moment de l'exécution.
L'intérêt pour les LDPL augmente à mesure que davantage de développeurs appliquent l'approche LPL aux systèmes dynamiques. Le premier atelier sur les LDPL a eu lieu lors de la 11e Conférence internationale sur les lignes de produits logiciels (SPLC) à Kyoto en 2007.

3.4 Les approches existantes à la prise de décision dans les systèmes adaptables

Dans la cette section, nous présenterons les différentes techniques de réalisation du comportement autonome basées sur les lignes dynamiques de produits logiciels. Nous mettrons l'accent sur l'aspect décisionnel du processus d'adaptation dans la mesure où nous l'avons identifié comme le point clé et le défis majeur de l'informatique autonome basée sur les lignes de produits logiciels dynamiques.

Premièrement, nous présenterons les principaux critères que nous avons utilisés pour évaluer et comparer les différentes approches. Ensuite, nous les présenterons par groupe d'approches connexes. Pour chaque groupe, nous discuterons leurs forces et leurs faiblesses et les comparons à notre propre proposition.

3.4.1 Critères de comparaison

Afin de comparer les différents travaux existants sur la planification dans les Lignes Dynamiques de Produits Logiciels (LDPL), nous avons sélectionné quatre critères qui peuvent être résumés comme suit:

- **Exploration**: Par exploration nous désignons à la fois la méthode utilisée pour explorer l'espace des configurations et le moment ou cette exploration est faite. Par conséquent, une approche de prise de décision peut explorer une partie ou la totalité de l'espace de recherche et / ou explorer l'espace des configurations à la conception, partiellement à l'exécution ou complètement à l'exécution.
- **Robustesse**: Les systèmes de logiciels embarqués modernes évoluent dans des environnements de plus en plus ouverts et donc imprévisibles. Il est, par conséquence important d'évaluer comment le sous système d'adaptation se comporte en présence de situations imprévues. Le terme robustesse désigne donc la capacité du système à s'adapter face à des scenarios qui n'ont pas été explicitement programmés au moment de la conception.

- **Optimisation**: de plus en plus, les utilisateurs des systèmes logiciels embarqués exigent de ces derniers d'offrir le meilleur service possible dans les différentes situations rencontrées. Souvent, l'optimisation des services fournis est motivée par des objectifs divergents. Ainsi, les techniques d'auto-adaptation peuvent soit maintenir le système dans un état cohérent (pas d'optimisation), soit optimiser son comportement par rapport à un objectif (optimisation mono-objectif) ou par rapport à plusieurs objectifs (optimisation multi-objectifs).
- **Évolutivité**: La mise à l'échelle (scalabilité) est une préoccupation importante dans les systèmes logiciels modernes. Pour être scalable, la stratégie d'adaptation doit être: a) facilement gérable lorsque la complexité des systèmes augmente; b) induire peu d'impact sur la consommation des ressources du système et les performances livrées; c) fournir de bons plans dans un laps de temps raisonnable. Ainsi nous distinguons trois niveaux de scalabilité : systèmes de petite tailles, systèmes de moyenne taille et systèmes de grandes taille.

Le tableau 3.1 résume les différences entre les méthodes discutées selon les critères définis ci-dessus.

Tableau 3.1 Résumé des approches à la prise de décision basées sur les LPLD

Approches	*Exploration*	*Robustesse*	*Optimisation*	*Mise à l'échelle*
(Cetina et al., 2008)	Règles ECA. Au moment de la conception	NON	NON	systèmes de petite taille
(Dinkelaker et al., 2010)(Shen et al., 2011)	Règles ECA/Strategies. Au moment de la conception	NON	NON	systèmes de petite taille
(Bencomo et al., 2008a)(Bencomo et al., 2008b)	Règles ECA. Au moment de la conception	NON	OUI	systèmes de petite taille
(Rosenmuller et al., 2011)	Règles ECA+ Solveurs SAT. Au moment de la conception	NON	NON	systèmes de petite ou moyenne taille
(Morin et al., 2009)(Morin et al., 2008)	Règles d'optimisation et adaptation manuelle. Au moment de l'exécution	OUI	OUI/NON	systèmes de taille moyenne
(White et al., 2007) (Rouvoy et al., 2009) (Svein et al., 2009) (Jian et al. 2010) (Perrouin et al. 2008).	Solveurs CSP	OUI	OUI	systèmes de taille moyenne
(Almeida et al. 2014)	Branch and Bound	OUI	OUI	systèmes de petite taille
(Pascual et al. 2015a)	Algorithme Génétique avec l'operateur de transformation	OUI	OUI	systèmes de taille moyenne
Notre approche	Algorithme Génétique avec l'operateur de sélection transitive	OUI	OUI	Systèmes de grande taille

Cetina et al ont proposé le moteur MoRe pour (Model-Based Reconfiguration Engine) pour mettre au point des systèmes autonomes (Cetina et al, 08). Leur proposition est basée sur le paradigme des modèles à l'exécution: Les modèles de caractéristiques sont utilisés pour raisonner sur les configurations valides du système à l'exécution. Ils montrent comment l'utilisation de modèles de caractéristiques pour représenter l'évolution du système afin bénéficier de connaissances

et d'outils basés sur des modèles. En ce qui concerne la prise de décision, les auteurs ont défini des règles ECA simples pour générer de nouvelles configurations à la survenance d'événements spécifiques. Les configurations générées sont vérifiées quant à leur validité par rapport aux contraintes du modèle de caractéristiques avant leur déploiement dans le système en cours d'exécution. Ils illustrent leur proposition sur le domaine de la domotique.

Dans (Dinkelaker et al. 10) et (Shen et al. 11), les auteurs ont exploré l'idée d'utiliser la programmation orientée aspects pour réaliser l'adaptation logicielle. Plus précisément, Dinkelaker et al introduisent l'idée des modèles dynamiques de caractéristiques et ont combiné diverses tendances AOP pour les mettre en œuvre (Dinkelaker et al. 10). L'objectif principal de leur proposition est d'éviter de spécifier des reconfigurations pour chaque nouvelle configuration de caractéristiques. La proposition a été appliquée à une ligne de produits logiciels de vente en ligne pour montrer sa faisabilité. Dans (Shen et al. 11), le concept de modèle de rôle pour améliorer la traçabilité entre les caractéristiques et leurs implémentations est introduit. Les auteurs ont utilisé des modèles de caractéristiques afin de documenter la variabilité du logiciel et AOP pour la mettre en œuvre. Une étude de cas sur un système de sélection de parcours a été rapportée. (Dinkelaker et al. 10) et (Shen et al. 11) utilisent des scénarios d'adaptation explicites basés sur les règles ECA pour la prise de décision.

L'approche Genie définit un middleware à base de composants basés sur des modèles architecturaux pour réaliser l'adaptation à l'exécution (Bencomo et al. 08.a) (Bencomo et al. 08.b). La prise de décision est à nouveau réalisée grâce à des règles ECA qui sont dérivées de machines d'état. Les auteurs ont soutenu que l'utilisation de machines d'état offre une représentation visuelle plus complète de la logique d'adaptation. Pour illustrer leur proposition, les auteurs ont rapporté une étude de cas sur une LDPL de Protocol de découverte de Services.

Dans (Rosenmuller et al. 11), Rosenmüller et al ont utilisé la Programmation Orientée Feature afin de réutiliser les connaissances de la LPL pour réaliser des systèmes adaptables. Pour réduire la complexité de la prise de décision, une partie de la variabilité de la gamme de produits est résolue au moment du design pour générer des gammes de produits logiciels dynamiques sur mesure. Du point de vue de la prise de décision, les règles ECA sont utilisées pour déclencher l'adaptation du système, puis un solveur SAT vérifie la validité de la nouvelle combinaisos. La proposition a été illustrée sur une famille de logiciels de réseau de capteurs.

On peut remarquer que toutes les approches mentionnées ci-dessus partagent une limite commune: la logique d'adaptation est câblée au moment du design, exigeant que tous les scénarios d'adaptation possibles aient été correctement prédits par les développeurs du système. Bien que

cette solution soit assez simple et puisse être intéressante pour les petits systèmes ou à des fins de preuve de concept, son application aux systèmes moyens et grands est prohibitive. En effet, pour de tels systèmes, l'espace des scénarios possibles est très important et il n'est pas possible de les prévoir au moment de la conception. De plus, le maintient et l'évolution de la logique d'adaptation peuvent rapidement devenir problématiques.

Pour faire face à ces limitations, une autre famille d'approches telles que (Pascual et al. 15.a) (Morin et al. 09) (White et al. 07) (Rouvoy et al. 09) (Almeida et al. 14) a vue le jour. Une valeur de coût ou d'utilité qui dépend du contexte d'exécution actuel est calculée à l'exécution pour chaque configuration du système. Cela évite d'énumérer tous les scénarios ou plans d'adaptation possibles. La prise de décision consiste plutôt à sélectionner la configuration ayant la valeur la plus élevée. En ce sens, cette famille de techniques est plus robuste car des scénarios imprévus sont naturellement anticipés à l'aide d'un algorithme de décision basé sur l'évaluation des configurations possibles.

Par exemple, Morin et al ont utilisé des règles d'optimisation pour optimiser les propriétés de QoS (Morin et al. 09) (Morin et al. 08). Les aspects sont étendus avec des informations de qualité de service et sont tissés à l'exécution pour optimiser la qualité de service de la propriété correspondante. Le procédé peut être partiellement manuel dans le sens que les aspects peuvent être tissés par l'utilisateur si aucune information de qualité de service ne leur est associée. Les auteurs ont démontré la faisabilité de leur proposition sur un système domotique dans le cadre du projet **Diva**.

Le principal inconvénient de ce travail est que l'optimisation est locale aux aspects et ne considère pas l'adéquation de la configuration globale. De plus, aucun modèle de variabilité n'est utilisé. Ainsi, les auteurs ont utilisé des invariants exprimés sous forme de méta-aspects qui doivent être spécifiés au moment du design. Enfin, si une configuration cible est jugée invalide, les auteurs ne proposent aucun mécanisme de correction et la configuration est simplement abandonnée.

Dans (White et al. 08), l'approche proposée dans (Benavides et al. 05) pour la reconfiguration a l'exécution. L'idée clé est d'utiliser les solveurs CSP pour résoudre la variabilité, considérant que la sélection des variantes (au moment de la conception ou au moment de l'exécution) peut être considérée comme un problème de satisfaction des contraintes (Russell et al. 10). Les auteurs ont développé l'outil **Scatter** pour sélectionner automatiquement les variantes du système qui optimisent une fonction de coût. Leur proposition a été appliquée aux systèmes mobiles.

Une autre approche fondée sur les buts est explorée dans le projet **MUSIC** (Rouvoy et al. 09) (Svein et al. 09) (Jiang et al. 10) (Perrouin et al. 08). **MUSIC** est l'extension du modèle

d'architecture du middleware **Madame** (Geihs et al. 09) pour l'informatique ubiquitaire. **MUSIC** utilise le Framework de planification basé sur les composants et les architectures orientées services. Du point de vue de la prise de décision, il utilise un algorithme de force brute ou certaines heuristiques pour évaluer, au moment de l'exécution, l'utilité de toutes les configurations valides. Ici, la fitness est une mesure qui reflète la satisfaction de l'utilisateur. La proposition a été illustrée sur un system d'assistance de voyage avec divers scénarios dans le métro de Paris.

Le principal inconvénient de ces approches est l'explosion du nombre de configurations possibles qui rend difficile leur utilisation dans le cas de grands systèmes complexes. En outre, bon nombre de ces approches sont mono-objectif et l'utilisation de solveurs CSP par exemple ne prend pas en charge l'optimisation multi-objectifs.

Dans (Almeida et al. 14), un algorithme de **branch and bound** est proposé pour faire face à ces limitations. La proposition a été validée dans le cadre d'applications multi-cloud pour assurer un service de cloud optimal en présence de fluctuations dans la QoS de différents fournisseurs de cloud. En dépit du fait que les auteurs rapportent une amélioration substantielle dans le temps de calcul des configurations optimales, il reste de complexité exponentielle ce qui signifie qu'il ne convient pas pour des systèmes à grande échelle complexes.

Une dernière catégorie d'approches de prise de décision sont les approches basées sur les méta heuristiques (Harman et al. 14). La principale préoccupation de ces travaux est l'explosion du nombre de configurations possibles. L'idée centrale est alors d'utiliser des méta-heuristiques pour explorer partiellement l'espace des solutions et diriger la recherche vers des espaces de recherche prometteurs. Théoriquement, la complexité temporelle est réduite à une complexité polynomiale au lieu d'exponentielle. Par conséquent, ces techniques peuvent réduire le temps global nécessaire pour produire de bonnes solutions. Cela se produit cependant au détriment d'une perte d'optimalité plus ou moins petite (habituellement acceptable). Par exemple, Guo et al ont montré que l'utilisation d'un algorithme génétique pour dériver des variantes au moment de la conception dans les LPL moyennes et grandes peut produire des solutions allant jusqu'à 87% optimalité dans 45% à 99% moins de temps (Guo et al. 11).

Le travail présenté dans (Pascual et al. 15.a) est une extension de (Guo et al. 11) à l'informatique mobile au moment de l'exécution. Les auteurs envisagent également l'optimisation multi-objectifs (Pascual et al. 15.b). Ici, l'idée clé est que l'algorithme génétique étant une méta heuristique converge vers l'optimum en générant des individus aléatoires et en les évoluant. Cependant, ces individus générés au hasard sont souvent invalides. Ainsi, ils doivent être transformés pour se conformer aux contraintes du modèle de caractéristiques. A cet effet, un

opérateur **fmTransform** est mis en œuvre. L'opérateur **fmTransform** prend en entrée un individu généré de façon aléatoire et produit en sortie un individu valide. Par conséquent, l'algorithme génétique produit des solutions de haute qualité tout en garantissant qu'elles sont conformes aux contraintes du modèle de variabilité.

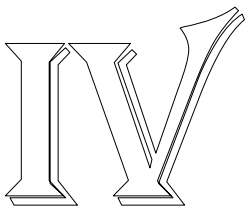
Bien que les auteurs de (Pascual et al. 15.a) aient affirmé que leur approche était bien adaptée aux grands systèmes, les résultats expérimentaux qu'ils ont rapportés tendent à montrer que le temps de convergence de l'algorithme génétique peut encore être important (plusieurs dizaines de secondes à plusieurs minutes) Dans le cas de systèmes moyens à grands (pas plus de 500 caractéristiques (Pascual et al. 15.a)). Il est clair que ces délais puissent être acceptables pour résoudre la variabilité statique au moment du design, ce n'est pas le cas pour l'adaptabilité au moment de l'exécution.

3.5 Conclusion

Dans ce chapitre nous avons introduit le domaine de l'informatique autonome et le paradigme des lignes dynamiques de produit logiciels. Nous avons survolé les principales approches à la prise de décision dans ce contexte et les avons comparés à notre propre approche. Nous avons notamment souligné l'insuffisance des travaux existant particulièrement en termes de complexité computationnelle et en temps de réponse. Afin d'y remédier nous définirons l'opérateur de sélection transitive qui vise à réduire le temps nécessaire pour générer les solutions valides.

Dans la suite de cette thèse, nous présentons notre opérateur de sélection transitive et nous montrons à travers une série d'expérimentations que son utilisation dans un algorithme génétique produit des solutions d'optimalité équivalente en moins de temps que l'algorithme génétique de (Pascual et al. 15.a) et (Guo et al. , 11).

Chapi



Les relations de dépendances transitives

4.1 Introduction

L'une des approches les plus populaires à l'adaptation des systèmes logiciels est l'utilisation des lignes de produits logiciels dynamiques ; la variabilité du système est exprimée par la model de variabilité et sa résolution est reportée à l'exécution. La sémantique formelle et la possibilité d'effectuer des raisonnements complexes sur les modèles de caractéristiques sont les clefs de leur adoption dans la discipline des systèmes autonomes et adaptables. Ainsi, les modèles de caractéristiques ont servi de base au raisonnement sur le comportement adaptable et les plans d'adaptation sont exprimés en termes de combinaisons de caractéristiques.

Bien que plusieurs travaux aient montré la pertinence de cette approche, certains défis restent toutefois à relever. Ainsi, la complexité des algorithmes de raisonnement sur les modèles de caractéristiques constituent un frein majeur à leur large adoption particulièrement dans le contexte embarqué. Or, comme le nombre de configurations possibles croît exponentiellement avec le nombre de fonctionnalités du système, le temps nécessaire pour la recherche d'une combinaison optimale ou même satisfaisante devient très important. Ceci est d'autant plus vrai dans le cas des systèmes logiciels embarqués modernes qui sont de plus en plus complexes et sont soumis à des contraintes plus ou moins strictes sur la consommation des ressources (notamment la puissance de la CPU et de l'énergie) et le temps de réponse.

A cet effet, nous présentons ci-après un ensemble de nouvelles relations de dépendances entre les caractéristiques dont l'objectif essentiel est de réduire la complexité calculatoire des algorithmes de raisonnement sur les modèles de caractéristiques en résolvant une partie de la variabilité (que nous qualifions de liée) et ce par l'exploitation des contraintes structurelles qui lient entre elles les fonctionnalités du systèmes.

Dans cette perspective, nous proposons une nouvelle conception de la notion de variabilité afin de distinguer la variabilité libre et la variabilité liée. A la lumière de cette redéfinition, nous revisitons la notion de dépendance entre caractéristiques et nous étudions ses deux dimensions qui sont la transitivité et le déterminisme. En nous basons sur une formulation novatrice de la dépendance, nous proposons, un algorithme de calcul de ces dépendances. Enfin, nous présentons les deux operateurs de sélection transitive qui les exploitent et servirons à l'implémentation efficace

des algorithmes de raisonnement sur les modèles de caractéristiques. Quelques algorithmes de raisonnement sont enfin présentés pour illustrer le propos.

4.2 La variabilité revisitée

Rappelons d'abord quelques définitions de la variabilité vues plus haut. Weiss et Lai définissent la variabilité des LPL comme «une hypothèse sur la façon dont les membres d'une famille peuvent différer les uns des autres » (Weiss et Lai, 1999), tandis que (Svahnberg et al. 2005) définit la variabilité comme «la capacité d'un système logiciel ou d'un artefact à être efficacement étendue, modifiée, personnalisée ou configurée pour une utilisation dans un contexte particulier. ». En combinant ces deux définissions et en se rapprochant du sens littéraire du terme français de la variabilité, nous pouvons proposer une nouvelle définition de la variabilité afin de dégager une caractéristique majeure de cette dernière qui est la liberté.

Définition 4.1 (La variabilité logicielle). La variabilité dans les lignes de produits logiciels est la capacité du système logiciel à se diversifier par l'extension, la modification ou la personnalisation d'un ensemble de ces artefacts (appelés points de variation) afin de différentier ces membres les des autres.

A partir de la définition 4.1, la variabilité apparait comme la possibilité offerte par une LPL de choisir une variante donnée parmi un ensemble de choix définissant un point de variation. En pratique, un choix dans un point de variation peut être soit :

- Indépendant des autres points de variation (point de variation libre): Le choix d'une variante dans le point de variation n'est ni déterminée ni conditionnée par le choix dans un autre point de variation.
- Dépendant d'un ou plusieurs points de variations (point de variation lié) : Le choix d'une variante dans le point de variation est conditionné ou déterminé par un autre choix dans un autre point de variation. Aussi, les deux points de variation sont dit liés.

Reprenant afin d'illustrer le propos, l'exemple de la ligne de produit des SGBD introduit au chapitre 2. La figure 4.1 récapitule cet exemple en le simplifiant légèrement.

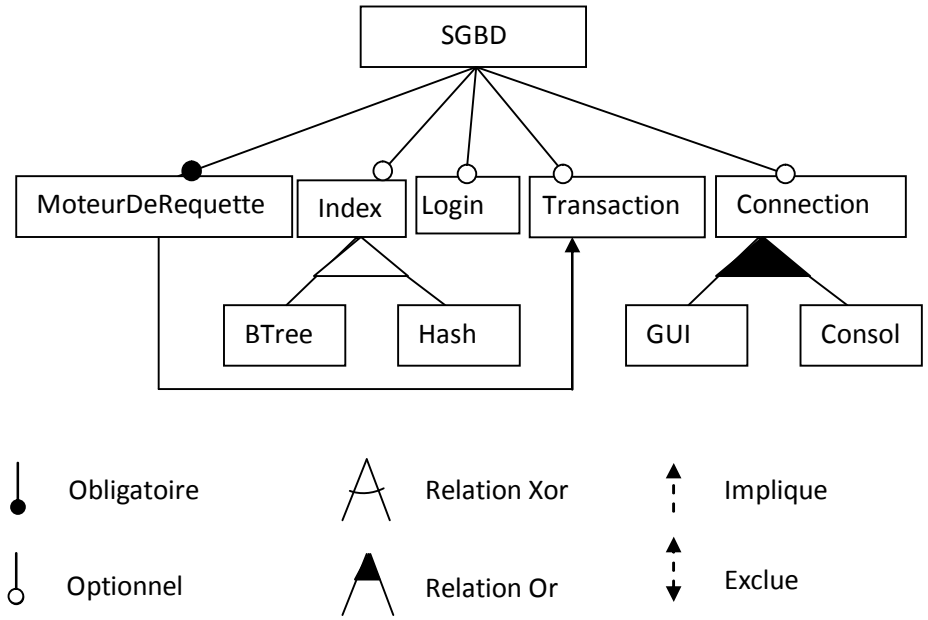


Figure 4.1 le modèle de fonctionnalité des systèmes de gestion des bases de données

Si nous considérons le point de variation *login* qui exprime la possibilité d'inclure ou ne pas inclure la fonctionnalité d'authentification au système, nous voyons que la décision d'inclure cette fonctionnalité n'est déterminée par aucun autre choix. Nous disons alors que $login$ est un point de variation libre. Au contraire, le point de variation *BTree* est lié dans la mesure où le choix de l'inclure dans un produit donné peut être déterminé par le choix opéré au niveau du point de variation *Hash* car les deux fonctionnalités sont exclusives.

4.2.1 Résolution partielle de la variabilité liée

Ainsi que nous l'avons expliqué plus haut, la variabilité se présente comme la capacité d'un système logiciel à varier par la réalisation de décisions sur un ensemble de points de variation pouvant être soit libres soit liés. Par métonymie, nous qualifierons la variabilité engendrée par les points de variation libre de variabilité libre et celle engendrée par les points de variation liée de variabilité liée.

Il est important à ce sujet de noter qu'en pratique, les points de variation liés sont beaucoup plus fréquents que les points de variation libres. Il suffit pour s'en rendre compte de considérer la structure même des modèles de caractéristiques qui sont le mécanisme le plus populaire pour la modélisation de la variabilité dans les LPLs. En effet, les différentes relations entre les

caractéristiques définissent en réalité des dépendances entre les points de variations correspondants. Ainsi par exemple, la relation hiérarchique entre caractéristiques exprime la contrainte que la sélection de la caractéristique fille est conditionnée par la sélection de sa caractéristique parent. De même la sélection d'une caractéristique obligatoire conditionne celle de la caractéristique parent. Le groupe de choix exclusif signifie que la sélection d'une caractéristique du groupe exclue (détermine dans le sens de l'exclusion) toutes celles qui sont dans le même groupe. Enfin les contraintes d'implication et d'exclusion lient les points de variations des deux caractéristiques impliquées dans la contrainte.

C'est dans ce sens qu'il convient de souligner l'importance de la notion de variabilité liée dans le processus d'analyse des lignes de produits (dynamiques) tel que représenté par le raisonnement sur les modèles de caractéristiques. En effet, si nous concevons ce processus comme une opération de recherche dans l'ensemble des produits de la ligne de produits (recherche de l'ensemble des produits dérivables, recherche de l'ensemble des produits non dérivables, recherche du sous ensemble des produits dérivables qui incluent/excluent une fonctionnalité donnée, recherche du produit le plus appropriée pour un besoin spécifique, …) la variabilité liée permet de restreindre l'espace de recherche global (tel que défini par la combinaison aléatoire des points de variations) à l'espace des produits dérivables uniquement (voire figure 4.2).

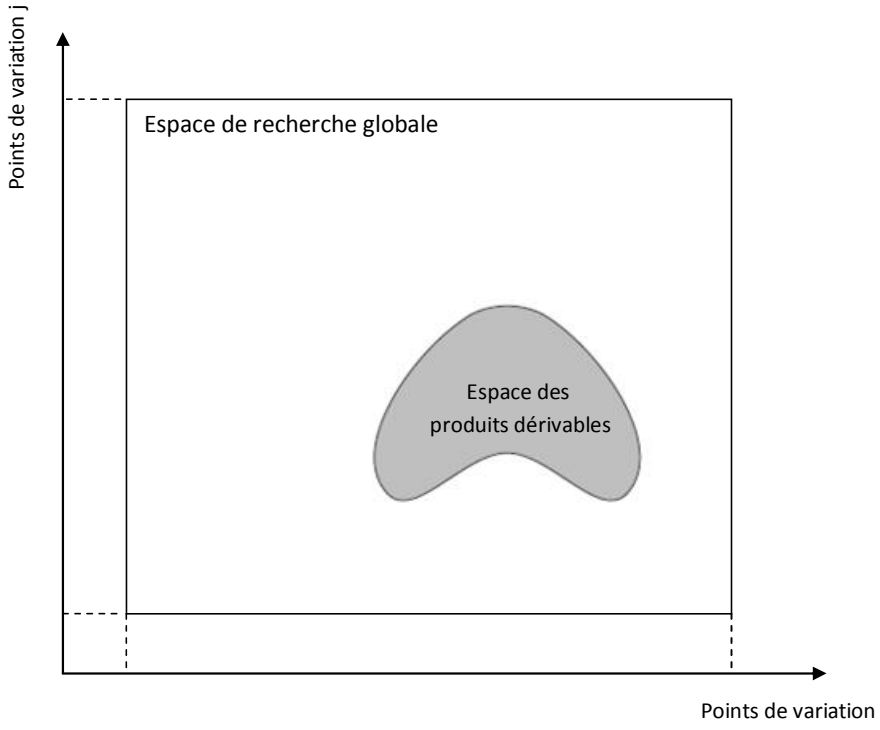


Figure 4.2. Impact de la variabilité liée sur l'espace de recherche

Ainsi, la variabilité liée permet de réduire significativement l'espace de recherche des algorithmes d'analyse et donc réduire sensiblement la complexité de calcul nécessaire pour le parcourir. Il suffit pour s'en convaincre de considérer le tableau 4.1. Le tableau 4.1 récapitule pour un nombre de modèles de caractéristiques de la littérature (Pascual el al. 2015b), le nombre de combinaisons aléatoires de **n** caractéristiques (ce nombre est donc égale à **2^n** et il reflète le volume de l'espace de recherche global) ainsi que le nombre de configurations valides (reflétant le volume de l'espace des produits dérivables de la LPL). Il peut être facilement constaté que ce dernier et très considérablement inferieur au premier (64 fois dans le cas de *x264* et ≈66621296 fois dans le cas de *Mobile Visit Guide*)

Tableau 4.1. Espace de recherche des produits valides et espace de recherche global

Nom du modèle	**Nombre de caractéristiques**	**Volume de l'espace de recherche global**	**Volume de l'espace des produits dérivables**
x264	17	2^17	2048
Wget	17	2^17	8192
Berkeley DB Memory	19	2^19	3840
Sensor Network	27	2^27	19152
Mobile Game	33	2^33	9198
Tank War	37	2^37	1,741,824
Mobile Media	43	2^43	2,128,896
Mobile Visit Guide	51	2^51	33,800,000
LPLOT-3CNF-500	500	2^500	3.779^e15
LPLOT-3CNF-1000	1000	2^1000	1.638^e131

Définition 4.2 (La variabilité partielle). Nous appelons résolution partielle de la variabilité liée, la délimitation stricte de l'espace des produits dérivables par le calcul à priori des point de variation liées.

La section suivante concrétise l'idée de la variabilité liée sur le formalisme des modèles de caractéristiques et développe la notion de résolution partielle par la proposition d'un modèle mathématique et d'un algorithme de calcul.

4.3 Les relations de dépendances entre caractéristiques

Dans le but de rendre l'idée de la variabilité liée exploitable dans le processus d'analyse des lignes de produits logiciels dynamiques, nous la concrétisons sur le formalisme des modèles de caractéristiques. A cet effet, nous projetons la notion de liberté des points de variation sur les

caractéristiques qui sont leur pendant dans le formalisme des modèles de caractéristiques. A cette fin, un ensemble de relations de dépendances dites transitives sont introduites. Nous exploitons le contexte formel de ce formalisme pour proposer une formulation mathématique consistante de la notion de variabilité liée basée sur les dépendances transitives entre caractéristiques.

4.3.1 Des points de variation liés aux dépendances entre caractéristiques

Les caractéristiques dans le modèle de caractéristiques expriment des points de variations dans le sens où la sélection/désélection d'une caractéristique reflète un choix qui résolve un point de variation dans un sens ou dans un autre (inclusion ou exclusion de la fonctionnalité dans un produit donné).

Ainsi, la transposition sur le formalisme des modèles de caractéristiques de la notion de liberté/liaison introduite plus haut sur les points de variation, nous amène à considérer sous une optique nouvelle la sémantique de ces modèles. En effet, nous envisageons le modèle de caractéristiques comme un ensemble de contraintes sur la sélection des fonctionnalités du système. Typiquement, cela s'exprime de la manière suivante :

Si $f_1$est sélectionnée pour être incluse dans une configuration alors f_2 doit être également sélectionnée.

Un autre exemple concerne la relation parent-fils qui peut être exprimée de la manière suivante:

Si la f_{Parent} est sélectionnée alors au moins l'une de ses caractéristiques filles doit être également sélectionnée.

Ainsi une première définition de la notion de dépendance peut être donnée comme suit :

Définition 4.3 (Dépendance entre caractéristiques). Nous disons qu'une caractéristique $f_2$dépend d'une autre caractéristique f_1 (ou qu'il existe une dépendance de f_1 vers f_2), si et seulement si, un choix sur la sélection de $f_1$implique un choix (différent ou le même) sur la sélection de $f_2$ou d'un groupe de caractéristiques dont $f_2$fait partie.

4.3.2 Les deux dimensions de la relation de dépendance

L'exemple précédent montre un aspect important de la relation de dépendance entre caractéristiques qui est le déterminisme. En effet, le choix sur la sélection d'une caractéristique peut déterminer (dans un sens ; sélection, ou dans l'autres ; désélection) le choix sur la sélection d'une autre caractéristique, mais il peut aussi le contraindre par rapport au choix de sélection d'un ensemble de caractéristiques. Ainsi par exemple, le cas de la relation parent-fils. La sélection de la

caractéristique parent ne suffit pas à elle seule à déterminer la sélection d'une sous-caractéristique mais implique une contrainte de sélection qui doit être vérifiée par au moins l'un des éléments de l'ensemble des sous caractéristiques.

L'autre aspect important de la relation de dépendance est la transitivité. Ainsi, une caractéristique peut dépendre d'une autre caractéristique directement, i.e. par l'application directe d'une contrainte structurelle du modèle de caractéristique (citons a titre d'exemple la contrainte d'implication entre caractéristiques) comme elle peut en dépendre de manière indirecte, i.e. par l'application de deux ou plusieurs contrainte structurelles du modèle de caractéristiques ; il est claire par exemple que si $f_1$requiert $f_2$et que $f_2$requiert $f_3$alors $f_1$requiert f_3. Le tableau 4.2 récapitule les deux dimensions de la relation dépendance entre Caractéristiques. Ce tableau est d'avantage développé dans les deux sous sections suivantes.

4.3.3 Déterminisme et indéterminisme des relations de dépendance

Définition 4.3 (Dépendance déterministe et non déterministe). Une relation de dépendance entre deux caractéristiques $f_1$et $f_2$est dite déterministe si et seulement si, le choix sur la sélection de $f_1$détermine à lui seul le choix de la sélection de f_2. Elle est dite non déterministe dans le cas ou le choix sur la sélection de $f_1$entre **en partie** seulement dans le choix sur la sélection de f_2.

Par exemple, considérant l'exemple du modèle de caractéristiques des SGBD illustré par la figure 4.1, nous pouvons voir que *BTree* et *Hash* dépendent indéterministiquement de *Index* car elles sont ses sous caractéristiques. Ainsi si *Index* est sélectionnée, au moins l'une des caractéristiques *BTree* ou *Hash* doit l'être également. Il existe une dépendance dans le sens inverse (c'est-à-dire, *Index* qui dépend de *BTree*, respectivement de *Hash*) qui est déterministe, i.e. la sélection de *Hash* (resp. *BTree*) entraine automatiquement celle de *index*.

Il est important de noter que dans le cas des modèles de caractéristiques classiques, seule la relation de sous-caractéristique donne lieu à des relations de dépendances non-déterministes.

La distinction entre relations déterministes et non déterministes est majeure car contrairement aux premières, les dernières impliquent un caractère aléatoire (le choix a priori arbitraire entre un ensemble de caractéristiques) qui nécessite des algorithmes appropriés (voir plus loin l'algorithme de sélection transitive pseudo-aléatoire).

Ces algorithmes sont donc source d'une complexité calculatoire supplémentaire par rapport aux algorithmes qui exploitent les dépendances déterministes. Prenons pour s'en convaincre l'opération de sélection de la caractéristique *BTree* dans l'exemple de la figure 4.1. Comme *BTree* requiert déterministiquement *Index* (considérant pour simplifier les dépendances directes

uniquement) la sélection de *BTree* requiert la sélection de $Index$. Si nous considérons à l'inverse la sélection de la caractéristique $Index$, Celle-ci implique la sélection de *BTree* ou *Hash* (non déterminisme). Ainsi il est nécessaire de prendre les deux chemins pour couvrir l'ensemble des solutions possibles. Si chacune de ces caractéristiques ont-elles mêmes des sous caractéristiques, le nombre de chemin à explorer devient très grand (exponentiel à la profondeur de la descendance)

Tableau 4.2. Les deux dimensions de la relation de dépendances

		Transitivité	
		Directe	Transitive
Déterminisme	Déterministe	Implication directe Exclusion Directe Implication inverse	Implication transitive Exclusion transitive Implication inverse transitive
	Non-déterministe	Sous-Caractéristique	Descendance transitive Descendance par implication transitive

4.3.4 Les dépendances directes entre caractéristiques

Définition 4.4 (Dépendance directes). Une relation de dépendance entre deux caractéristiques f et e est dite directe si et seulement si il existe **une** contrainte structurelle du modèle de caractéristiques qui lie entre elles les f et e.

A partir de la définition 2.1 du chapitre 2, nous pouvons dégager les différents types de contraintes établissant une relation de dépendance directes entre caractéristiques dans les modèles de caractéristiques classiques. Il s'agit de :

- $f_1$est parent de f_2 (donc $f_2$est sous caractéristique de f_1),
- $f_1$est sous caractéristique obligatoire de f_2,
- $f_1$et $f_2$sont dans le même sous groupe *Xor*,
- $f_1$implique $f_2$par une contrainte d'implication, et
- $f_1$et $f_2$sont exclusives par une contrainte d'exclusion.

Il convient de noter que ni le choix multiple ni l'optionalité n'établissent de dépendance entre les caractéristiques. En effet, si $f_1$et $f_2$sont dans le même groupe Or, il n'est pas possible de déterminer le choix sur la sélection de f_1a partir du choix de sélection de f_2 (et inversement). De même si $f_1$est une sous caractéristique optionnelle de f_2, nous ne pouvons pas déduire l'état de sélection de f_1à partir de l'état de sélection de $f_2$si ce n'est par la relation de sous caractéristique

invoquée plus haut.

Reformulons à présent les contraintes du modèle de caractéristiques.

D'abord, considérons le modèle de caractéristique $FM =< G, r, E_{MAND}, F_{XOR}, F_{OR}, Impl, Excl >$ ou :

• $G = (F, E)$ est un arbre où F est l'ensemble des caractéristiques et $E \subseteq F \times F$ l'ensemble des arêtes qui représentent la décomposition hiérarchique descendante des caractéristiques, c.-à-d., relations Parent-enfant.

• $r \in F$ est la caractéristiques racine;

• $E_{MAND} \subseteq E$ est l'ensemble des arêtes qui définissent les caractéristiques obligatoires avec leurs parents;

• $F_{XOR} \subseteq \mathcal{P}(F) \times F$ et $F_{OR} \subseteq \mathcal{P}(F) \times F$ définissent les groupes de caractéristiques exclusives (Xor-groupes) et inclusives (Or-groupes);

• $Impl$ est l'ensemble des contraintes d'implication entre caractéristiques qui ont la forme $A \Longrightarrow B$ où $A \in F$ et $B \in F$ (nous disons : A *implique* B)

• $Excl.$ est l'ensemble des contraintes d'exclusion entre caractéristiques, qui ont la forme $A \Longrightarrow \neg B$ où $A \in F$ et $B \in F$ (nous disons : A *est exclusive avec* B)

Ainsi, nous pouvons exprimer les contraintes du modèle de caractéristiques qui établissent une relation de dépendance comme suit :

- **Paternité** : f est la caractéristique parent de e. Alors, à chaque fois que e est sélectionnée, f est sélectionnée également : $si\ (f, e) \in E, alors, \forall c \in [\![FM]\!], e \in c \Longrightarrow f \in c$
- **Obligatoire** : e est une sous-Caractéristique obligatoire de f. Alors, si f est sélectionnée alors e doit l'être également : $si\ (f, e) \in E_{MAND}, alors, \forall c \in [\![FM]\!], f \in c \Longrightarrow e \in c$
- **Alternative** : Soit $f_1, f_2,$ a $f_n$des caractéristiques qui appartiennent à un groupe Xor. Alors, une seule Caractéristique f_i peut être sélectionnée à la fois : si $\{f_1, f_2,$ a $f_n\}, \mathrm{f_{Parent}} \in \mathrm{F_{XOR}}$, alors, $\forall c \in [\![FM]\!], si\ f_i \in c \Longrightarrow \forall f_j \in \{f_1, f_2, .., f_n\} - \{\mathrm{f_i}\}, f_j \notin c$
- **Sous-Caractéristiques** : soit f_{Parent} la caractéristique parent de f_1 a f_n. Alors si f_{Parent}est sélectionnée alors au moins une sous-caractéristique f_i doit être sélectionnée : $si\ (f_{Parent}, f_1), \ldots (f_{Parent}, f_n) \in E, alors, \forall c \in [\![FM]\!], f_{Parent} \in c \Longrightarrow \exists f_i \in \{f_1, \ldots f_n\}, f_i \in c$

Figure 4.3. *Exemple illustratif de la notion de dépendance*

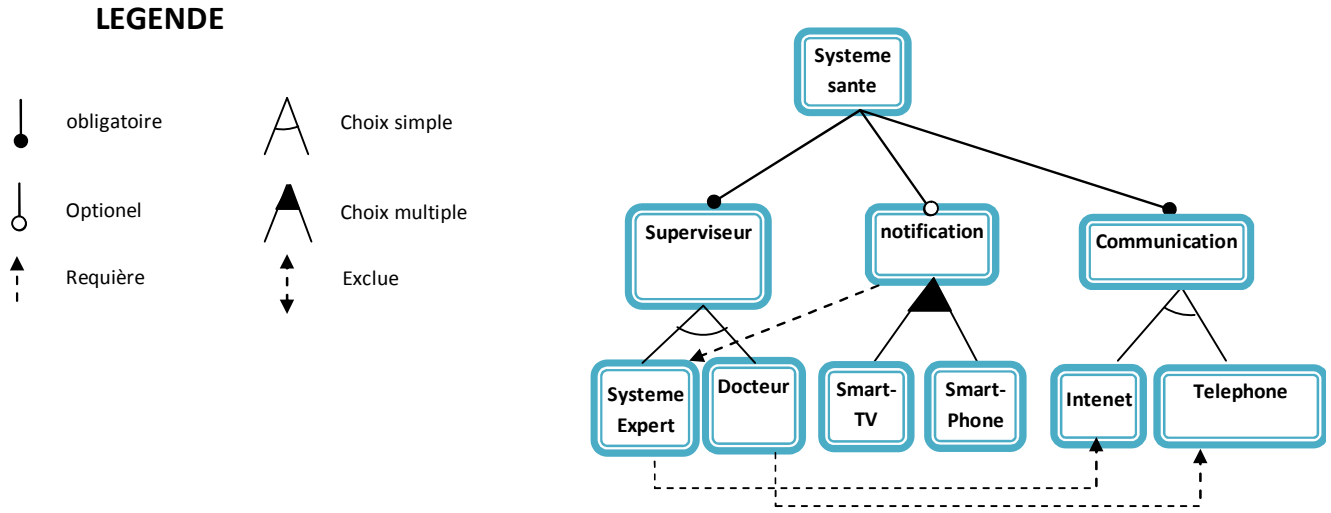


Tableau 4.3 les dépendances directes et transitives de *SystemeExpert*

Dependances directes (voisines) de Systeme expert	**dependences Transitives de System expert**
Implications directes *: Superviseur(parent), Internet(constraint* ***requires****)* ***Exclusions directes*** *: Docteur(meme groupe xor)* ***implicationInverses directes*** *: Notification(constraint requires)*	***Implications transitives*** *:systeme sante, Superviseur, Communication, Internet* ***Exclusions Transitives****: Docteur, Telephone* ***implicationInverses transitives****: Notification*

- **Implication** : s'il existe une contrainte d'implication : $f \Longrightarrow e$ alors e doit être sélectionnée à chaque fois que f est sélectionnée : $f \Longrightarrow e\{\text{alors}, \forall c \in [\![FM]\!], si\ f \in c \Longrightarrow e \in c$
- **Exclusion** : S'il existe une contrainte d'exclusion $f \Longrightarrow \neg e$ alors e doit être désélectionnée si f est sélectionnée et vis vers ca : $si\ f \Longrightarrow \neg e$, alors, $\forall c \in [\![FM]\!], si\ f \in c \Longrightarrow e \notin c\ et\ si\ e \in c \Longrightarrow f \notin c$

4.3.4.1 Les dépendances directes entre caractéristiques

A partir de ce qui précède, nous pouvons distinguer les trois relations de dépendances entre caractéristiques :

L'inférence *directe* : $f\ requiert\ directement\ e$

$$f \longmapsto^{d} e \overset{\Delta}{\Leftrightarrow} \begin{cases} f\ est\ la\ caracteristique\ parent\ de\ \ e\ ou \\ f\ une\ caracteristique\ obligatoire\ de\ e\ ou \\ f \Longrightarrow e\ ou \\ e\ est\ l'unique\ caracteristique\ selectable\ de\ f \end{cases}$$

L'exclusion directe: $f\ et\ e\ sont\ directement\ exculisifs$

$$f \leftrightarrow^{d} e \overset{\Delta}{\Leftrightarrow} \begin{cases} f\ et\ e\ appartiennent\ au\ meme\ groupe\ Xor\ ou \\ \exists\ une\ constrainte\ \ d'exclusion\ f\ \ \Rightarrow \neg\ e \end{cases}$$

L'ascendance directe :

$f\ est\ l'ascendant\ direct\ de\ e\ ou\ e\ est\ le\ descendant\ direct\ de\ f$

$$f \rightharpoonup e \overset{\Delta}{\Leftrightarrow} e\ est\ une\ sous\ caracteristique\ de\ f.$$

Nous ajouterons par commodité la relation inverse de l'inférence directe :

L'inférence Inverse direct : f est requises par e

$$f \leftarrow\!\dashv^{d} e\ (f\ est\ directement\ requise\ par\ e) \overset{\Delta}{\Leftrightarrow} e \longmapsto f$$

Il est alors facile établir pour toutes caractéristiques $e, f \in E$

$f \longmapsto^{d} e\ \Rightarrow \forall c \in [\![FM]\!], si\ f \in c \Longrightarrow e \in c$

$f \leftrightarrow^{d} e \Rightarrow \forall c \in [\![FM]\!], si\ f \in c \Longleftrightarrow e \notin c$

$\forall c \in [\![FM]\!], si\ f \in c \Longrightarrow \exists\ e\ est\ une\ sous\ caracteristique\ de\ f\ tel\ que\ e \in c$

4.3.5 Les dépendances transitives entre caractéristiques

Généralisons à présent les relations de dépendances directes introduites plus haut pour inclure les dépendances entre des caractéristiques qui ne sont pas liées par des contraintes directes. Cette généralisation exploite simplement le caractère transitif des relations de dépendances directes. Ainsi, Nous redéfinissons la notion de dépendance en prenons en compte celle de dépendance indirecte :

Définition 4.5. (Dépendance entre caractéristiques). Une caractéristique f est dite $transitivement\ dependante$ (ou simplement : $depandante$) de e s'il existe une suite de dépendances directes dont l'origine est la caractéristique e et la fin la caractéristique f.

La définition 4.5 reformule la notion abstraite de dépendance introduite dans la définition 4.2 (l'existence d'une causalité entre la sélection de deux caractéristiques f et e) par la recherche d'une suite de dépendances directes (que nous pouvons également qualifier de locales) qui, une fois considérées ensemble, permettent d'établir cette causalité. Ainsi, cette causalité ainsi que la dépendance qui en découle peuvent elles être qualifiées de globales.

Formellement, les dépendances entre caractéristiques se présentent alors comme suit :

L'inférence globale ou transitive: $f\ requiert\ globalement\ ou\ transitivement\ e$

$$f \longmapsto e \overset{\Delta}{\Leftrightarrow} \begin{cases} f\ \longmapsto^{d} e\ ou \\ \exists g \in F, f\ \longmapsto^{d} g\ et\ g\ \longmapsto e \end{cases}$$

L'exclusion transitive ou globale: f *et* e *sont globalement ou transitivement exculisifs*

$$f \leftrightarrow e \overset{\Delta}{\Leftrightarrow} \begin{cases} f \leftrightarrow^{d} e \ ou \\ \exists g \in F, f \mapsto g \ et \ g \leftrightarrow e \ ou \\ \exists g \in F, f \leftarrow\!\!\shortmid g \ et \ g \leftrightarrow e \end{cases}$$

Nous ajouterons par commodité la relation inverse de l'inférence directe :

L'inférence Inverse transitive ou globale: f est *globalement ou transitivement* requises par e

$$f \leftarrow\!\!\shortmid e \overset{\Delta}{\Leftrightarrow} e \mapsto f$$

Par ce qu'elle ne représente aucun intérêt immédiat du point de vue de l'efficacité calculatoire, la relation de descendance transitive n'est pas davantage explorée et nous nous contentons d'exploiter la relation de descendance directe introduite plus haut. De plus, par commodité, nous utiliserons les termes d'inférence, exclusion et inférence inverse pour designer les relations de dépendances transitive ou globale.

Les relations de dépendances transitives expriment une dépendance globale entre caractéristiques. Ainsi :

-***l'inférence*** traduit l'idée que, f requiert e si et seulement si e doit être sélectionnée à chaque fois que f est sélectionnée. Ceci se vérifie si et seulement si et seulement si :

- f requiert directement e
- f requiert directement g et g requiert e.

-***L'exclusion*** traduit l'idée que, f est exclusive avec e si et seulement si e doit être désélectionnée à chaque fois que f est sélectionnée et vice vers ca. Ceci se vérifie ssi :

- f et e sont directement exclusives.
- f requiert g et g est exclusive avec e
- f est requise par g et g est exclusive avec e

-L'inférence inverse est simplement la relation inverse de l'inférence. Elle est cependant importante à tracer pour chaque caractéristique dans la mesure où elle se traduit par le fait que *f* est requise par *e* implique que *f* doit être désélectionnée si *e* est désélectionnée.

4.3.6 Les ensembles de dépendances transitives

À partir des définitions des relations de dépendances directes et transitives introduites plus haut, nous pouvons définir pour chaque caractéristique les ensembles suivants :

$\boldsymbol{DescendancesDirectes(f)} = \{g \in F, (f, g) \in E\}$

$\boldsymbol{InferenceDirectes(f)} = \{g \in F, f \mapsto g\}$

$= \{g \in F, (g,f) \in E\}$ /* la caractéristique parent */

$\cup \{g \in F, (f,g) \in E_{MAND}\}$ /* les sous-caractéristiques obligatoires */

$\cup \{g \in F, f \Longrightarrow g \in Impl\}$ /* les caractéristiques impliquées */

$\cup \{g \in F, (f,g) \in E \quad et \; \forall e \in$

$\boldsymbol{DescendancesDirectes(f)} - \{g\} \; alors \; e \; est \; une \; caracteristique \; morte\}$

/* les sous-caractéristiques uniques */

Ou, une caractéristique morte se définie comme une caractéristique qu'il est logiquement impossible de sélectionner. Concrètement, une telle caractéristique requiert une autre avec laquelle

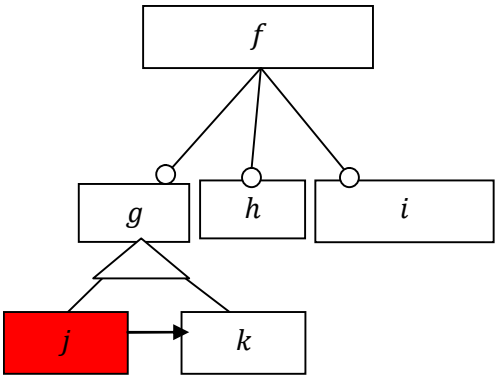


Figure 4.4 *: Exemple de caractéristique morte : j. j requiert k qui l'exclue*

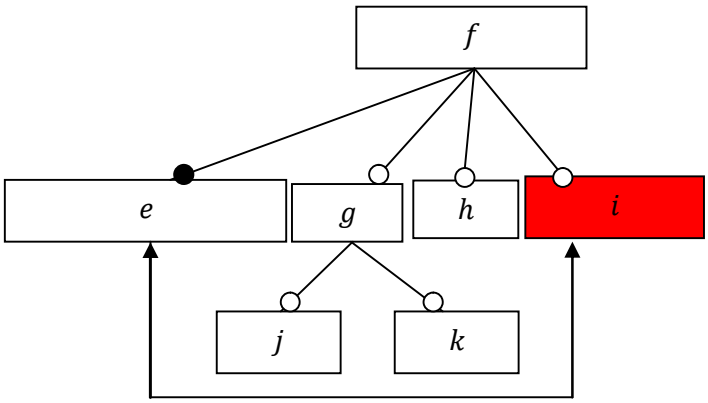


Figure 4.5 *: Exemple de caractéristique morte : i. i exclue e qui est obligatoire (donc est requise par i)*

elle est exclusive. La figure 4.4-7 illustrent quelques exemples de caractéristiques mortes.

$\boldsymbol{Inferences(f)} = \{g \in F, f \mapsto g\}$

$$= InferencesDirectes(f) \bigcup_{e \in inferenceDirectes(f)} inferences(e)$$

$\boldsymbol{ExclusionsDirectes(f)} = \{g \in F, f \leftrightarrow^{d} g\}$

$$= \{g, (g, .. f) \in F_{xor}\} U \{g, g \Longrightarrow \urcorner f \in Excl$$

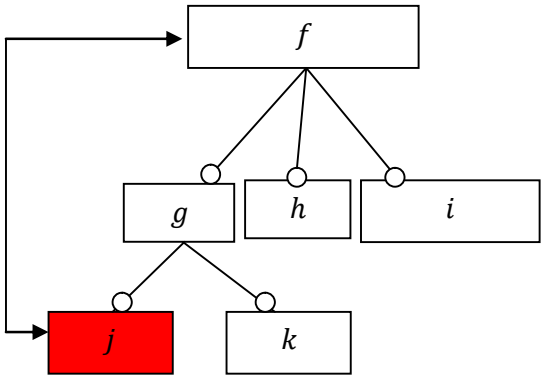


Figure 4.6 *: Exemple de caractéristique morte : j. j exclue la racine (qui est requise par toutes)*

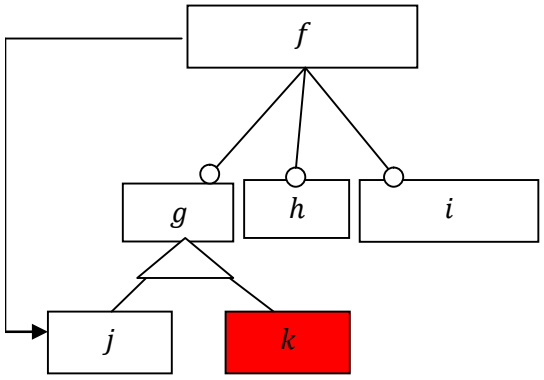


Figure 4.7 *: Exemple de caractéristique morte : k. k exclue j qui est requise par la racine (donc requise par k elle meme)*

$\boldsymbol{Exclusions(f)}$= $\{g \in F, f \leftrightarrow g\}$

$= ExclusionsDirectes\,(f) \bigcup_{e \in inferences(f)} exclusions(e) \bigcup_{e \in exclusion(f)} inferencesInverses(e)$

$\boldsymbol{inferencesInversesDirectes(f)} = \{g \in F, f \leftarrow\!\!\!\!\!\!\cdot^{d}\ g\}$

$=\{g \in F, (f,g) \in E\}$ /* les sous-caractéristiques*/

$\cup\ \{g \in F, (g,f) \in E_{MAND}\}$ /* le parent si f est obligatoire */

$\cup\ \{g \in F, g \Longrightarrow f \in Impl\}$ /* les caractéristiques impliquantes */

$\cup\ \{g \in F, (g,f) \in E \quad et\ \forall e \in$

$DescendancesDirectes\ (g) - \{f\} \quad alors\ e\ est\ une\ caracteristique\ morte$

/* f est une sous-caractéristiques unique */

$\boldsymbol{inferencesInverses(f)} = \{g \in F, f \leftarrow\!\!\!\!\cdot\ g\}$

$= inferencesInversesDirectes\,(f) \bigcup_{e \in inferencesInversesDirectes(f)} inferencesInverses(e)$

Ces differents ensembles seront calculées par un algorithme inspiré de Dijkstra (voire

section 4.4.2) et exploité par les différents opérateurs de sélection de caractéristiques que nous présentons dans la section suivante.

4.3.7 L'opérateur de sélection transitive de caractéristiques

Ci-après nous proposons d'exploiter la formulation donnée dans la section précédente pour accélérer le processus de génération de nouvelles combinaisons de caractéristiques. À cet effet nous introduisons l'opérateur de sélection transitive de caractéristiques dont le l'opération de sélection est donnée par l'algorithme 4.1.

Algorithme 4.1 L'opérateur de sélection transitive : L'opération de sélection

Paramètres d'entrée: $f, selectionnees, deselectionnees$

$SelectionnerTransitivement(f)$

$debut$

$selectionnees \leftarrow selectionnees\ U\ \{f\}$

$pour\ chaque\ caracteristique\ e\ \in inferences(f),$

$selectionnees\ \leftarrow selectionnees\ U\ \{e\}$

$pour\ chaque\ caracteristique\ e\ \in exclutions(f),$

$deselectionnees\ \leftarrow deselectionnees\ U\ \{e\}$

Fin

Il est d'abord important de noter que la sélection transitive est déterministe. En effet, les opérations de sélection d'une caractéristique sont propagées aux seules dépendances déterministes (voire section 4.3.3). Donc aucun calcul aléatoire ou de parcours exhaustif n'est nécessaire pour l'implémentation de ces opérations. Ceci n'est pas le cas lorsque nous tentons de répercuter l'action de sélection/désélection sur une dépendance non-déterministe (cela est toutefois nécessaire pour la génération de configurations valides). Dans ce cas, un algorithme aléatoire (dans le cas ou nous nous contentons de générer une solution possible) ou de parcours exhaustif (dans le cas ou nous voulons générer toutes les solutions possibles) est nécessaire d'où l'intérêt de l'opérateur de sélection pseudo aléatoire introduit plus bas.

L'intérêt de l'opérateur de sélection transitive réside dans le fait que la décision d'inclusion/exclusion de plusieurs caractéristiques est faite à la fois par une seule opération de sélection/désélection, réduisant ainsi l'espace de recherche d'une manière significative. Prenant pour s'en convaincre les exemples des modèles de caractéristiques des figures 4.8-10.

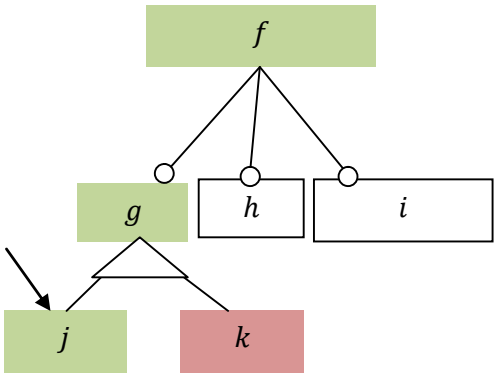


Figure 4.8 *: la sélection transitive de la caractéristique j*

Ainsi, la sélection transitive de la caractéristique j de la figure 4.8 induit-elle la sélection des caractéristiques $f\ et\ g \in inferences(j)$ et la désélection de la caractéristique $k \in exclusions(j)$. Aussi, l'espace de recherche se réduit-il à 2^2 choix valides au lieu de 2^5.

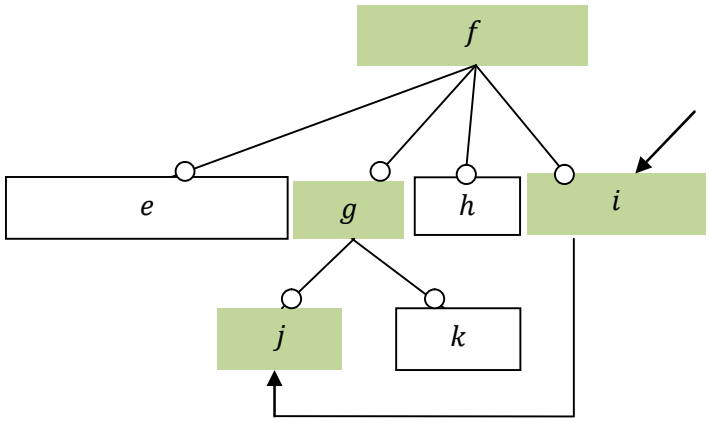


Figure 4.9 *: la sélection transitive de la caractéristique i*

L'exemple illustrée par la figure 4.9 montre que la sélection transitive de la caractéristique i induit celle des caractéristiques $f, j\ et\ g \in inferences(i)$. Le volume de l'espace de recherche passe ainsi a 2^3 au lieu de 2^6. Si nous supposons une relation d'exclusion entre j et k (le cas de la figure 4.10) alors l'espace de recherche se réduit a 2^2 choix valides.

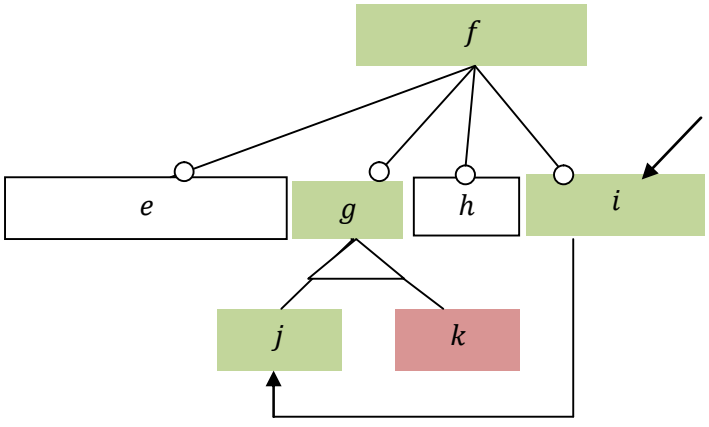


Figure 4.10 *: exclusion de k par la sélection transitive de la caractéristique i*

Le tableau 4.4 récapitule le gain relatif à l'exploitation de l'opérateur de sélection transitive par rapport à la sélection simple de caractéristiques des trois modèles vus plus haut.

Tableau 4.4. L'intérêt de l'utilisation des opérations de sélection transitives

	Volume de l'espace de recherche		
Le modèle de caractéristiques	Après une sélection transitive (en nombre de combinaisons restantes)	Après une sélection simple (en nombre de combinaisons restantes)	Gain relatif
De la figure 4.8	2^2	2^5	2^3
De la figure 4.9	2^3	2^6	2^3
De la figure 4.10	2^2	2^6	2^4

Intéressons nous à présent à l'opération de désélection transitive. La désélection transitive d'une caractéristique f consiste à désélectionner f puis désélectionner toutes les caractéristiques qui la requièrent (les éléments de l'ensemble$inferencesInverses(f)$). De plus, un soin particulier doit être accordé à la désélection de la caractéristique parent de f (ainsi qu'à celles des éléments de $inferencesInverses(f)$) si celle(s)-là n'ont plus aucunes sous caractéristiques « sélectables ». La figure 4.11 illustre le fonctionnement de l'opération de désélection dont l'algorithme est donné par les algorithmes 4.2 et 4.3.

Dans le cas de la figure 4.11, nous appliquons l'opération de désélection transitive à la caractéristique v. La fonction $parentaDeselectionner(v)$ retourne g. C'est donc cette caractéristique qui sera désélectionnée ainsi que ses inférences inverses dont notamment la caractéristique f. Par ailleurs, f atant été la seule caractéristique sélectable de e, cette dernière doit également être transitivement désélectionné. Dans le pire des cas, l'algorithme termine sur une ou plusieurs sous caractéristiques de la racine (la racine étant elle-même inclue dans toutes les

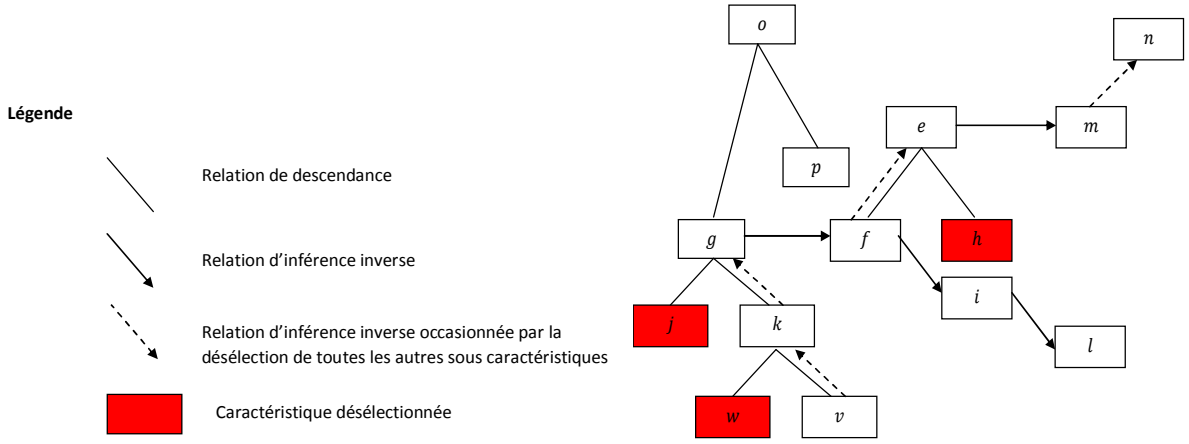


Figure 4.11 *: Fragment d'un modèle de caractéristiques avec impact de la désélection transitive de la caractéristique v*

combinaisons valides). Ainsi, si la racine est désélectionnée alors la combinaison est rejetée.

Algorithme 4.2 L'opérateur de sélection transitive : L'opération de désélection

Paramètres d'entrée: $f, selectionnees, deselectionnees,$

$DeselectionnerTransitivement\ (f)$"

$dedut$

$e \leftarrow parentaDeselectionner\,(f)$

$deselectionnees \leftarrow deselectionnees\ U\ \{e\}$ ***"Désélectionner e"***

$deselectionnees \leftarrow deselectionnees\ U\ inferencesInverses(e)$ ***"Désélectionner les inférences inverses de e y compris f***"

$Pour\ chaque\ g \in inferencesInverses(e) - descendances(e) faire$:

$h \leftarrow parentaDeselectionner\,(g)$

$si\ h \neq g\ alors$

$deselectionnerTransitivement(h)$

$finSi$

$finPour$

fin

Algorithme 4.3 la fonction de recherche du parent à désélectionner

$parentsaDeselectionner\,(f)$

"***Renvoi un ascendant (direct ou indirect) de f dont toutes les sous caractéristiques ont été désélectionnées.***

Retourne f si un tel parent n'existe pas"

$debut$

$e \leftarrow f$

$tant\ que\ \big(sousCaracterisitiquesDirectes\big(parent(e)\big) - \{e\}\big) - deselectionnees = \emptyset\ faire$

"toutes les autres sous caractéristiques de l'ascendant de e sont désélectionnées"

$e \leftarrow parent(e)$

$finTantQue$

$renvoyer\ e$

fin

4.3.8 L'opérateur de sélection transitive pseudo-aléatoire de caractéristiques.

L'opérateur de sélection transitive pseudo aléatoire de caractéristiques sélectionne de manière aléatoire une sous caractéristique pour chaque caractéristique sélectionnée par l'opérateur

de sélection transitive. Son fonctionnement est illustré par l'algorithme 4.4.

Algorithme 4.4 L'opérateur de sélection transitive : L'opération de sélection

Paramètres d'entrée: $f, selectionnees, deselectionnees$

$PA\_SelectionnerTransitivement(f)$

$debut$

$e_{finale} := ChoisirSousCaracterisitqueFinale(f, selectionnees, deselectionnees)$

$selectionnerTransitivement(e_{finale})$

Fin

Lorsqu'une caractéristique est sélectionnée, l'algorithme vérifié l'existence de sous-caractéristiques. Si tel est le cas, une série de sous- caractéristiques à partir de la caractéristique courante jusqu'à une caractéristique finale est choisie aléatoirement. Ensuite, la caractéristique finale est sélectionnée ainsi que ses dépendances en utilisant l'opérateur de sélection transitive de l'algorithme 4.1. Il convient de noter que la sélection transitive de la sous caractéristique finale implique la sélection de la caractéristique elle-même dans la mesure où celle-ci fait partie de l'ensemble inférences de la sous-caractéristique finale.

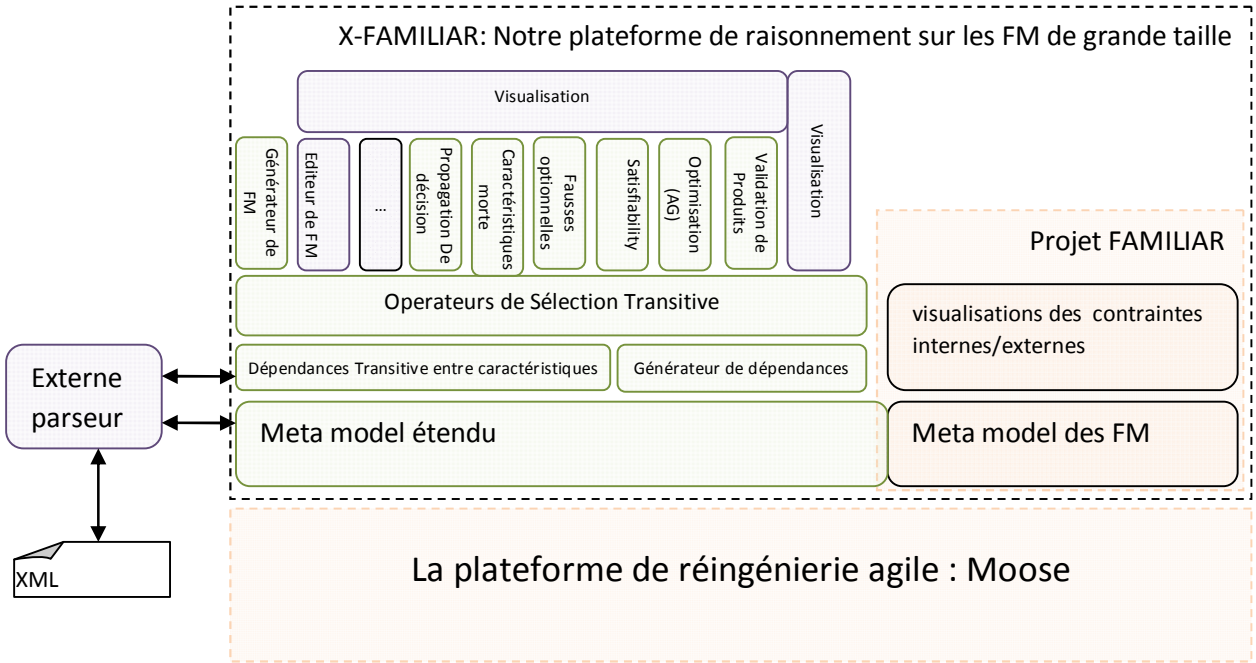


Figure 4.12. Architecture générale de la plateforme de raisonnement sur les Modèles de caractéristiques

4.4 Validation

Afin de valider la formulation mathématique des relations de dépendances transitives introduites dans la section précédente. Nous l'avons implémenté dans le cadre d'une plateforme de raisonnement sur les lignes de produits. La figure 4.12 représente une vue d'ensemble de la plateforme proposée pour raisonner sur les grandes lignes de produits. La plateforme proposée est basée sur ***Moose***, un projet open source pour le reengineering agile (*moosetechnoogie.org*). Les cases bleues de la figure 4.12 représentent les projets existants. Les boîtes vertes représentent des outils que nous avons déjà développés alors que les boîtes pourpres représentent ceux que nous projetons de développer prochainement.

4.4.1 Structures de données

Pour représenter les modèles de caractéristiques, nous avons réutilisé et étendu le code du projet FAMILIAR (FAMILIAR, 2014). FAMILIAR est une plateforme qui supporte la décomposition des LPL complexes en fournissant des visualisations des contraintes internes et externes entre les FM (Urli et al. 2015). Nous avons notamment étendu le projet avec des méthodes pour générer et accéder aux dépendances entre les caractéristiques.

La principale structure de données pour la représentation des dépendances entre les caractéristiques est $FMEncoder$ maintient un dictionnaire associant à chaque fonctionnalité ses dépendances transitives. Alors, l'accès aux dépendances d'une caractéristique f peut s'effectuer par les opérations suivantes:

$FMEncoder[f].inferences$, retourne des caractéristiques requises par f.

$FMEncoder[f].exclusions$, retourne les caractéristiques exclues par f.

$FMEncoder[f].inferencesInverses$, retourne les caractéristiques qui requièrent f.

Notez que les dépendances d'une caractéristiques (ainsi que les combinaisons de caractéristiques c'est-à-dire une configuration ou un produit de la LPL) sont codées comme des instances de la structure $BitSet$. $BitSet$ est simplement implémenté sous la forme d'un tableau de bits. Donc,

- $aBitSet[i] = 1$ signifie que $feature_i$ est contenu dans l'ensemble représenté par $aBitSet$.
- $aBitSet[i] = 0$ signifie que $feature_i$ n'est pas contenu dans l'ensemble représenté par $aBitSet$.

En outre, $BitSet$ implémente les méthodes suivantes:

$bitUnion(aBitSet1, aBitSet2)$. Mise à jour de $aBitSet1$pour coder l'union des ensembles représentés par $aBitSet1$ et $aBitSet2$. retourne *vrai* si $aBitSet1$ a été modifié.

$bitIntersection$: $(aBitSet1, aBitSet2)$. Met à jour $aBitSet1$pour coder l'intersection des ensembles représentés par $aBitSet1$ et $aBitSet2$. Retourne *vrai* si $aBitSet1$ a été modifié.

$EnsembleDesCaracteristiques\,(aBitSet1)$. Retourne la collection des caractéristiques codées par $aBitSet1$.

Par ailleurs, $FMEncoder$ maintient l'ensemble des caractéristiques mortes, lui-même codé sous la forme d'un BitSet et ci apres accede par $FMEncoder.caracteristiquesMortes$

4.4.2 Calcul des dépendances

L'algorithme 4.5 reproduit le processus de génération des dépendances entre les caractéristiques d'un modèle de caractéristiques $aFeatureModel$. Ce code est inspiré par l'algorithme bien connu de Dijkstra dont le principal point fort est une réduction significative de la complexité de calcul.

Algorithme 4.5. Algorithme de calcul des relations de dépendances.

Paramètres d'entrée: $aFeatureModel$

$generaterDependences\,(aFeatureModel)$

$debut$

$aFMEncoder \leftarrow initializerDependences\,(aFeatureModel)$.

$encodersOntChange \leftarrow vrai$.

$tantque(encodersOntChange)$

$encodersOntChange \leftarrow MettreaJourDependences\,(aFMEncoder\,, aFeatureModel)$

$Retourner\ aFMEncoder$

Fin

Le code de l'algorithme 4.5 commence par l'initialisation des dépendances de chaque caractéristique avec les dépendances directes de celles-ci. Pour rappel, les dépendances directes de la caractéristique f sont définies par les ensemble $inferencesDirectes(f)$, $exclusionDirectes\,(f)$, $inferencesInversesDirectes\,(f)$et $descendancesDirectes\,(f)$. Par exemple, les inférences directes d'une caractéristique sont initialisées avec:

1- Les caractéristiques impliquées par une contrainte $implique$,

2- La caractéristique $parent$.

3 Les sous – caractéristiques obligatoires.

De plus l'ensemble des caractéristiques mortes est initialisé à l'ensemble vide. C'est-à-dire que tout les bits de $FMEncoder.caracteristiquesMortes$ sont initialement à 0.

Ensuite, une boucle de mise à jour de la structure est exécutée afin d'ajouter les dépendances transitives des caractéristiques directement dépendantes (ci-après désignées par voisines), et ce conformément aux définitions de la section 4.3.6. Ce processus est répétée jusqu'à ce plus aucune dépendance supplémentaire n'est trouvée.

L'algorithme 4.6 représente la fonction mise à jour des dépendances transitives des caractéristiques du modèle dans la structure $FMEncoder$.

Algorithme 4.6. mise à jour des dépendances transitives

Paramètres d'entrée: $\boldsymbol{aFeatureModel}, \boldsymbol{aFMEncoder}$

$MettreaJourDependences\ (aFMEncoder, aFeatureModel)$

$debut$

$encodersOntChange \leftarrow faux.$

$pour\ cahque\ caracteristique\ f\ faire$

$encodersOntChange$

$\leftarrow encoderOntChange\ ou\ MaJDependancesde\,(f, aFMEncoder, aFeatureModel).$

$finPour$

$retourner\ encoderOntChange$

fin

Ou $MaJDependancesde$ est décrite par le code suivant : (algorithme 4.7)

Algorithme 4.7. mise à jour des dépendances d'une caractéristique f

Paramètres d'entrée: $\boldsymbol{f, aFeatureModel, aFMEncoder}$

$MaJDependancesde(f, aFMEncoder, aFeatureModel)$

$debut$

$encodersOntChange \leftarrow faux.$

$pour\ chaque\ caracteristique\ e \in inferencesDirectes(f)\ faire$

$encodersOntChange$

$\leftarrow encodersOntChange\ \boldsymbol{OU}\ bitUnion(aFMEncoder[f].inferences, aFMEncoder[e].inferences)$

$encodersOntChange \leftarrow encodersOntChange\ \boldsymbol{OU}$

$bitUnion(aFMEncoder[e].inferencesInverses, aFMEncoder[f].inferencesInverses)$

$encodersOntChange$

$\leftarrow encodersOntChange\ \boldsymbol{OU}\ bitUnion(aFMEncoder[f].exclusions, aFMEncoder[e].exclusions)$

finPour

$pour\ chaque\ caracteristique\ e \in exclusion(f)\ faire$

$encodersOntChange$

$\leftarrow encodersOntChange\ \boldsymbol{OU}\ bitUnion(aFMEncoder[f].exclusions, aFMEncoder[e].exclusions)$

$finPour$

$Si\ (aFMEncoder(f).inferences\ bitAnd\ aFMEncoder(f).exclusions\ contient\ des\ bits\ a\ 1\ alors$

$marquerCommeMorte(f)$

$encodersOntChange \leftarrow vrai$

$finSi$

$retourner\ encoderOntChange$

fin

La mise à jour est faite selon les définitions des ensembles de dépendances transitives donnés dans la section 4.3.6. A savoir, pour une caractéristique f:

- si f requiert directement e et e requiert g alors f requiert g
- si f requiert e et f est requise par g alors g requiert e
- si f requiert e et e Exclut g alors f Exclut g . Et enfin,
- Si f exclut e et e est requise par g alors f Exclut g

La fonction $marquercommeMorte(f)$ ajoute f à l'ensemble des caractéristiques mortes, ainsi que toutes les caractéristiques qui requiert f (c.-à-d. $inferenceInverse(f)$) puis verifie si l'ascendant de f a une seule sous-caractéristique non-morte. Si c'est le cas, cette derniere est ajouté

à l'ensemble des inférences directes de son ascendant. Ce fonctionnement est illustre par l'algorithme 4.7.

Algorithme 4.7. marquer une caractéristique comme morte

Paramètres d'entrée: $f, aFeatureModel, aFMEncoder$

$MarquerCommeMorte(f)$

$debut$

"Ajouter f ainsi que ces inférences Inverses a l'ensemble des caractéristiques mortes"

$bitUnion\ (CaracterisitquesMortes, aFMEncoder[f].inferencesInverses)$

"Quels sont les sous caractéristiques du parent de f qui sont morte"

$abitSet \leftarrow (aFMEncoder[Parent(f)].descandants\ bitIntersection\ CaracterisitquesMortes)$

"Quels sont les sous caractéristiques du parent de f qui ne sont pas mortes"

$abitSet \leftarrow abitSet\ XOR\ aFMEncoder[Parent(f)].descandants$

$DescendantsNonMorts \leftarrow ensembleDesCaracteristiques\,(aBitSet)$

"si une seule de ces sous caractéristiques du n'est pas morte"

$Si\ cardinalite(DescendantsNonMorts) = 1\ alors$

"alors l'ajouter aux dépendances directes du parent"

$e \leftarrow choisirLaCaracteristique(DescendantsNonMorts)$

$inferencesDirectes(Parent(f)) \leftarrow inferencesDirectes\,(Parent(f)) \cup \{e\}$

$finSi$

fin

À la fin de ce processus, nous avons les dépendances de chaque caractéristique du modèle FM. Dans la suite de cette thèse, nous pouvons à présent exploiter ces dépendances pour implémenter efficacement les différentes opérations sur les modèles de caractéristiques.

4.5 Implémentation des opérations d'analyse des modèles de caractéristiques basées sur les dépendances transitives

En utilisant les dépendances transitives entre caractéristiques, nous pouvons définir quelques opérations de bases sur les modèles de caractéristiques de la manière suivantes:

1. satisfiabilité du modèle de caractéristiques:

Un modèle de caractéristique, FM est satisfiable si au moins un produis est dérivable a partir de la LPL qu'il représente. Dans la mesure où un produit doit nécessairement inclure au moins une caractéristique finale et que celle-ci requiert et est requise par la racine du modèle, alors un modèle

de caractéristique est satisfiable implique que la racine n'est pas dans l'ensemble des caractéristiques morte donc :

Algorithme 4.8. satisfiabilité des modèles de caractéristiques

Paramètres d'entrée: $aFeatureModel, aFMEncoder$: l'encodeur de dépendances correspondant

$estSatisfiable\ (aFeatureModel)$
$debut$
  $si\ (\ aFMEncoder.caracteristiquesMortes[aFMEncoder[racine].indice] = 0)\ alors$
    $Retourner\ vrai$
  $Sinon$
    $Retourner\ Faux$
  $finSi$
fin

2. caractéristiques mortes

Les caractéristiques mortes sont calculées par l'algorithme de calcul des dépendances et ne nécessite donc aucun calcul supplémentaire :

Algorithme 4.9. calcul des caractéristiques mortes

Paramètres d'entrée: $aFeatureModel, aFMEncoder$: l'encodeur de dépendances correspondant

$CaractéristiquesMorte(aFM)$
$Debut$
  $Retourner(ensembleDesCaracteristiques\ (aFMEncoder.caracteristiquesMortes))$
fin

3. validation de produits

Un produit représente par une combinaison c de caractéristique est valide si et seulement si, pour toute caractéristique f :

- si $f \in c$ alors $\begin{cases} inferences(f) \sqsubseteq c\ (1) \text{ et} \\ exclusions(f) \cap c = \emptyset\ (2)\ et \\ descandants(f) \cap c \neq \emptyset\ (3) \end{cases}$
- et si $f \notin c\ alors\ inferencesInverses(f) \cap c = \emptyset (4)$

Puisque $inferences\big(parent(f)\big) \sqsubseteq inferneces(f)$et $exclusions(parent(f)) \subseteq exclusion(f)$ alors il suffit de vérifier les conditions 1 et 2 pour les caractéristiques finales. Aussi, Par ce que $inferencesInverses(f) \sqsubseteq infernecesInverses\big(parent(f)\big)$ alors il suffit de verifier la condition 4 pour les caractéristiques parents si elles sont désélectionnées (i.e. il n'est pas nécessaire de la vérifier pour les sous-caractéristiques d'une caractéristique désélectionnée). La condition 3 en

revanche doit être vérifiée pour toutes les caractéristiques. Cela donne l'algorithme 4.7

Algorithme 4.10. vérification de la validité d'un produit

Paramètres d'entrée: c : une collection de caractéristiques représentant un produit, $aFeatureModel, aFMEncoder$: l'encodeur de dépendances correspondant

$ProduitValide(c)$
$Debut$
  $productBitSet \leftarrow bitSetDe(c)$
  $Pour\ chaque\ caractéristique\ f$
    $Si\ f\ est\ finale\ alors$
      $si\ productBitSet[aFMEncoder[f].indice] = 1\ alors$
        $TousDansProduit(aFMEncoder[f].inferences, productBitSet)$
        $AucunDansProduit(aFMEncoder[f].exclusions, productBitSet)$
      Sinon
        $AucunDansProduit((aFMEncoder[f].inferencesInverses, productBitSet)$
      $finSi$
    $finSi$
  $finPour$
fin

Afin de mettre en œuvre les autres opérations, nous exploitons l'opérateur de sélection transitive décrit par les algorithmes 4.2 et 4.3. L'intuition derrière cet operateur est de propager instantanément la sélection ou la désélection d'une caractéristique donnée en fonction de ses dépendances.

Les deux opérations de sélection et de désélection transitives peuvent être exploitées pour implémenter les opérations sur les FM comme suit:

4. Calculer les produits dérivables: Cette opération génère simplement tous les produits valides grâce à un algorithme de force brute. Cette opération est donc NP complète. Cependant, l'utilisation de l'opérateur de sélection transitive réduit la complexité de l'algorithme car l'espace de recherche est considérablement réduit à chaque fois qu'une décision sur la sélection d'une caractéristique est faite. En effet, plusieurs caractéristiques sont définies simultanément. Par conséquence, a chaque opération de sélection/désélection l'espace de recherche est réduit proportionnellement au nombre

de dépendances de la caractéristique et non pas simplement divisée par 2 comme illustré par le tableau 4.4. En outre, l'operateur de sélection transitive nous permet de garantir que le produit génère est valide. Par conséquent, il n'est pas nécessaire de vérifier sa validité (qui est un facteur supplémentaire pour augmenter la complexité l'algorithme de force brute).

Le processus de recherche des produits dérivables commence par la sélection de la racine du modèle de caractéristique (index = 1) dans un $BitSet$ d'éléments nuls:

$toutLesProduits\ (aFeatureModel)$

$debut$

$produits := selectionnerEtExplorer(aFeaturemodel, 1, nilElemBitSet).$

$Return\ products.$

fin

La méthode $selectionnerEtExplorer$ invoque elle-même la méthode $evaluerSousConfigurations$ qui effectue une recherche exhaustive des solutions dans l'ensemble de l'espace de recherche comme suit:

Algorithme 4.11 exploration des sous-configurations valides

$evaluerSousConfigurations(\ aBitSet\ , index)$

$debut$

"$La\ feature\ a\ l'index\ index\ n'a\ pas\ ete\ positionee\ encore$"

$si\ (aBitSet\ [index] = nil)\ alors$

$deselectinnerEtExplorer(\ index\ , aBitSet.)$

$selectinnerEtExplorer(\ index, aBitSet)$

$sinon$

"$La\ feature\ a\ l'index\ index\ a\ deja\ ete\ positionee$"

$si\ indice < nbreCaracteristiques\ alors$

$evaluerSousConfigurations(aBitSet\ , index + 1)$

$sinon\ retourner\ configuration\ Correspondant(aBitSet)$

$finsi$

fin

Où:

$selectinnerEtExplorer(\ index\ , aBitSet.)$ Est une méthode récursive qui effectue une sélection transitive de la caractéristique à l'index i. Le résultat est un sous-ensemble de produits dérivables qui comprennent la caractéristique à l'index i. La méthode $deselectinnerEtExplorer$ positionne le bit i et tous les bits dépendants dans aBitSet. Si tous les bits sont positionnés dans le BitSet, (nous

avons un produit final), alors le produit est ajouté à l'ensemble de ceux dérivables. Autrement, l'algorithme continue à évaluer les configurations sous l'indice suivant. L'opération de désélection est effectuée en suivant la même logique avec la méthode: de$selectinnerEtExplorer(\ index\ , aBitSet.)$

Le nombre de produit dérivables est simplement obtenu comme suit :

Algorithme 4.12. Calcul du nombre des produits dérivables

$nomberDesProduitsDerivables\ (aFeatureModel)$
$debut$
$Retourner\ nombredElementsDans\ (toutLesProduits\ (aFeatureModel)$
fin

5. Calcul des commonalites:

C'est le rapport des produits incluant une caractéristique donnée au nombre total de produits dérivables:

Algorithme 4.13 Calcul des commonalités d'une caractéristique

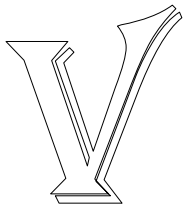
$RatioDesProduitsIncluant\ (aFeature, aFeatureModel)$
$Debut$
$products:=\ selectionnerEtExplorer(aFeaturemodel, indexOf(f), nilElemBitSet)$
$retourner\ nombreDesElementsDans\ (products)$
$/\ nomberDesProduitsDerivables\ (aFeatureModel)$
fin

4.6 Conclusion

Dans ce chapitre nous avons introduit les notions de variabilité liée et de relations transitives entre les caractéristiques. Leur définitions mathématiques ont été données et un algorithme efficace pour le calcul des dépendances transitives a été également présenté. Nous avons ensuite introduit les deux operateurs de sélection transitive qui exploitent les dépendances transitives. L'utilisation des dépendances transitives et des operateurs de sélection transitive nous ont permis d'implémenter efficacement les différents algorithmes de raisonnement sur les modèles de fonctionnalités. Quelques uns de ces algorithmes ont été proposés pour illustrer le propos. Plus particulièrement, les algorithmes de propagation de la sélection, de la validation de produits et de dérivation de produits

qui sont couramment rencontrés dans les techniques d'adaptation à l'exécution ont fait l'objet d'une explication étendue. L'avantage de notre approche par rapport aux techniques de raisonnement existantes, notamment les solveurs logiques, est la complexité réduite des algorithmes de raisonnement qui est elle-même due a la réduction progressive de l'espace des solutions par la propagation des choix des caractéristiques via les dépendances transitives précédemment calculées. Cette propriété sera plus longuement expliquée dans le chapitre suivant qui est consacré a la prise de décision orientée objectifs et basée sur un algorithme génétique exploitant les notions introduites dans ce chapitre.

Chapi



Une approche évolutionnaire pour la prise de décision

5.1 Introduction

Les systèmes logiciels sont de plus en plus mobiles, pervasifs et connectés. Par conséquence, ces systèmes doivent continuellement adapter leur comportement aux changements de l'environnement dans lequel ils évoluent afin d'offrir un service optimal. Récemment, Plusieurs travaux ont monté l'intérêt de l'adoption des lignes de produits dynamiques pour les systèmes adaptables ; L'adaptation est décrite en terme de points de variation, représentés par le modèle de caractéristiques (ou feature model), dont la résolution est reportée au moment de l'exécution. Toutefois, les approches existantes soufrent du manque de mise à l'échelle. En effet, la complexité de la prise de décision (le choix de la configuration cible) est exponentiellement corrélée aux nombre de fonctionnalités des systèmes induisant un important temps de calcul. Dans ce chapitre nous présentons un nouvel algorithme génétique pour la prise de décision dans les systèmes embarqués adaptables basée sur l'exploitation des opérateurs de sélection transitive de caractéristiques introduits plus haut. Ces nouveaux operateurs exploitent la notion de dépendance transitives entre les caractéristiques afin d'accélérer la génération de configurations valides. Afin de montrer l'intérêt de notre proposition nous la comparerons à un autre algorithme génétique pour la sélection de caractéristiques. Nous rapporterons les résultats expérimentaux montrant que notre approche produit des solutions de qualité comparable aux meilleures méthodes connues dans la littérature en un temps sensiblement moins long.

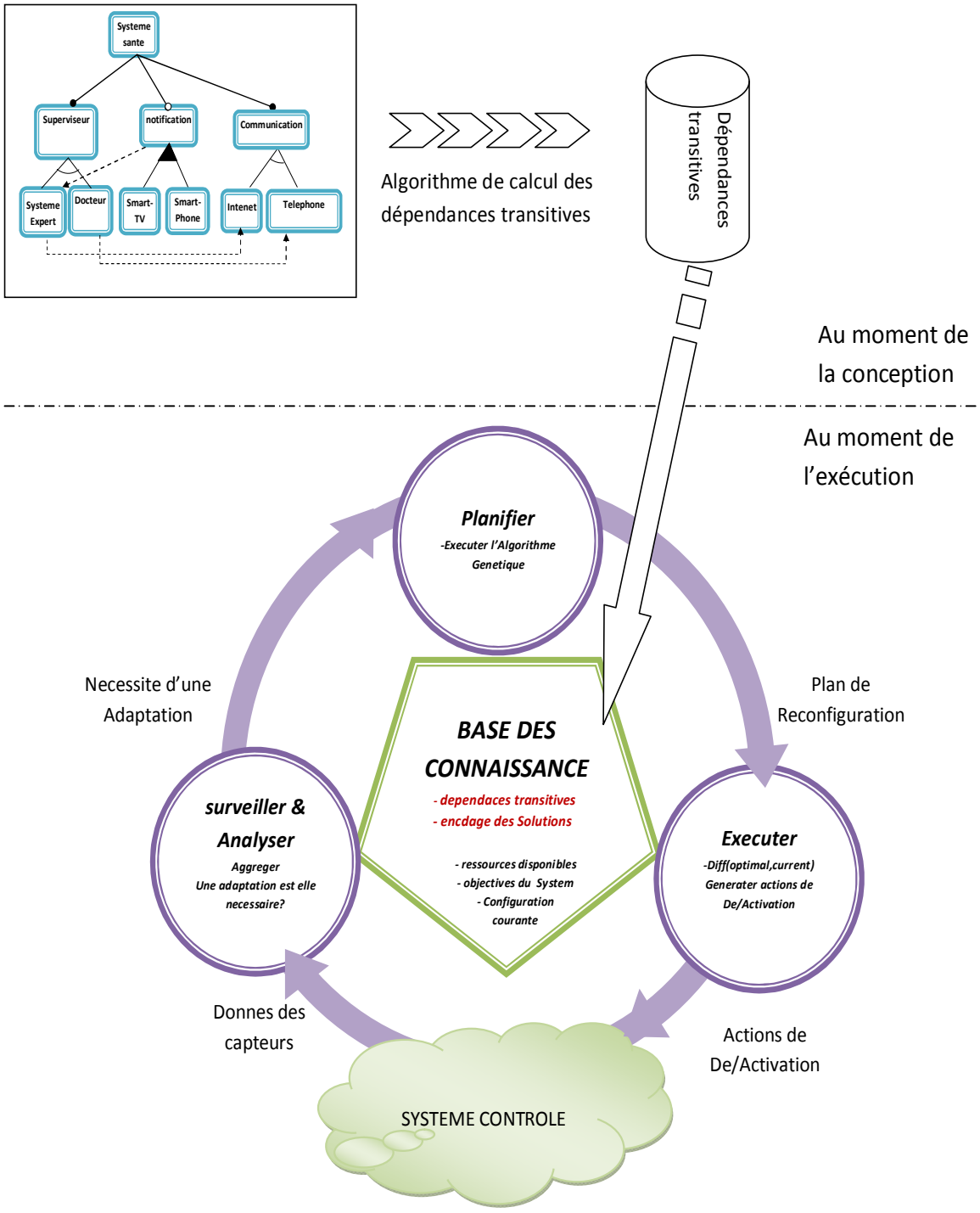


Figure 5.1 : Aperçus de l'approche

5.2 Aperçu de l'approche

La Figure 5.1 est une adaptation de la boucle de contrôle MAPE-K. Deux étapes sont explicitement distinguées. La première étape, à la conception, consiste en la construction de la base

de connaissance pour l'adaptation. Celle-ci comporte principalement les éléments suivants :

- Les dépendances entre caractéristiques: pour chaque caractéristiques f, l'ensemble de ses dépendances exprimé par les ensembles $inferences, exclusions, InferencesInverses$ et $descendants$.
- Les fonctions de fitness : $F(configuration, contexte)$ qui associe une valeur numérique reflétant sa qualité à une configuration donnée, étant donné un contexte d'exécution précis.
- La représentation des solutions : Qui décrit comment une solution potentielle (une combinaison de caractéristiques) est représentée dans l'algorithme génétique.

La seconde étape commence à l'exécution ; l'infrastructure de surveillance exploite les différents capteurs pour collecter les données du contexte d'exécution. Si les changements reportés sont suffisamment signifiants, l'algorithme génétique est exécuté pour générer une nouvelle configuration qui optimise les objectifs du système. Finalement, la nouvelle configuration est exploitée pour générer un plan de reconfiguration qui sera exécuté par l'infrastructure de reconfiguration.

Dans le présent chapitre, nous nous intéressons principalement à l'étape de planification consistant en l'algorithme génétique. Ce dernier est plus amplement expliqué dans la section suivante.

5.3 Un nouvel algorithme génétique pour la prise de décision dans les systèmes logiciels embarqués adaptables

Dans cette section, nous présentons l'algorithme génétique qui implémente la prise de décision dans les systèmes adaptables. Cet algorithme exploite les opérateurs de sélection transitive introduits précédemment pour accélérer le processus de génération de solutions. Mais avant d'aborder cet algorithme plus en détails, appliquons nous à développer une formulation mathématique du problème de prise de décision en fonction d'abord de la combinaison des caractéristiques, puis en fonction de la sélection des caractéristiques et des dépendances entre celle-ci.

5.3.1 Formulation mathématique

La prise de décision (ou planification) consiste en la recherche des (ou de la) configuration du système qui répond le mieux aux besoins de l'utilisateur ou plus généralement optimise les objectifs du système. Nous pouvons des lors proposer une première formulation du problème de la

prise de décision comme suit :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(x)}\right) \\ x \in \overline{Conf} \end{cases} \quad (1)$$

Ou $\overline{\text{Conf}}$ represente l'ensemble des configurations valides du système et Θ_{context} represente l'ensemble de ses fonctions-objectifs. Il est d'abord important de mentionner au sujet de Θ_{context} qu'il s'agit d'un ensemble (une ou plusieurs) fonctions-objectifs, généralement divergentes voire contradictoires. Par exemple, dans le cas d'un appareil mobil, les fonctions objectifs pourraient correspondre aux facteurs de consommation d'énergie, la surchauffe de l'appareil et le confort d'utilisation : rendu des graphiques, luminosité de l'écran, connectivite …). Par ailleurs, ces objectifs peuvent-ils être ou non fonction du contexte d'exécution. Ainsi, la consommation d'énergie ne dépend elle que des applications en cours d'exécution. Par contre, la surchauffe de l'appareil dépend elle en partie de la température ambiante. De même la luminosité de l'écran est elle fonction de la lumière ambiante et la connectivité de la position géographique,…

Intéressons nous à présent à l'ensemble des configurations valides Conf. Ainsi que nous l'avons expliqué dans le chapitre 3, les lignes de produits dynamiques utilisent des abstractions, tel que les caractéristiques, pour raisonner sur et évoluer les configurations du système a l'exécution. Ainsi, les états du système sont ils exprimées en termes de combinaisons de caractéristiques. Soit $FM =< G, r, E_{MAND}, F_{XOR}, F_{OR}, Impl, Excl >$ le modèle de caractéristique qui décrit la variabilité dynamique du système, alors la formule (1) peut elle être réécrite de la manière suivante :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(c)}\right) \\ c \in [\![FM]\!] \end{cases} \quad (2)$$

En se référant à la définition de l'ensemble $[\![FM]\!]$, nous pouvons réécrire la formule (2) comme suit :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(c)}\right) \\ c \in \mathcal{P}(F) et \\ c\ satisfait\ les\ contraintes\ de\ FM \end{cases} \quad (3)$$

Nous allons à présent exploiter la notion de dépendance liée introduite au chapitre précédent pour reformuler la formule (3) de la manière suivante :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(c)}\right) \\ c \in \mathcal{P}(F) et \\ \forall f \in c, inferences(f) \sqsubseteq c \\ \forall f \in c, exclusions(f) \cap c = \emptyset \\ \forall f \in c, descandantsDirectes\,(f) \cap c \neq \emptyset \\ \forall f \notin c, inferencesInverses(f) \cap c = \emptyset \end{cases} \quad (4)$$

La formule (4) exprime une idée majeure de notre travail qui est la vision de la prise de décision comme un problème d'optimisation sur un sous ensemble de l'espace des combinaisons de caractéristiques qui sont liée par des relations de dépendances transitives. Ainsi que nous l'avons expliqué au chapitre précédent, la variabilité liée et les relations de dépendances transitives permettent de propager instantanément la décision de (de)sélection d'une caractéristiques et de garantir la validité de la combinaison obtenue. Ceci a un impact immédiat sur le temps de génération des solutions pour le problème de recherche et donc un impact certain sur les performances globale de l'algorithme de recherche

Adoptons par ailleurs la notation suivante : x_i^c représente le choix de sélection de la caractéristique f_i dans une configuration c valide de FM ($c \in [\![FM]\!]$). Plus particulièrement :

- $f_i \in c \Leftrightarrow x_i^c = 1$
- $f_i \notin c \Leftrightarrow x_i^c = 0$

Nous pouvons enfin exploiter $FMEncoder$ qui est la représentation binaire des ensembles de dépendances transitives correspondant à FM et calculé par l'algorithme 4.5. Cela donne la nouvelle formulation suivante :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(X)}\right) \\ X \in \{0,1\}^{card(F)}\ et \\ x_i^c = 1 \Longrightarrow \begin{cases} \forall j \in [1, card(F)], si\ FMEncoder[f_i].inferences[j] = 1 \Longrightarrow x_j^c = 1 \\ \forall j \in [1, card(F)], si\ FMEncoder[f_i].exclusions[j] = 1 \Longrightarrow x_j^c = 0 \\ \exists j \in [1, card(F)], FMEncoder[f_i].descendants[j] = 1 \Longrightarrow x_j^c = 1 \end{cases} \\ x_i^c = 0 \Longrightarrow \forall j \in [1, card(F)], si\ FMEncoder[f_i].inferencesInverse[j] = 1 \Longrightarrow x_j^c = 0 \end{cases} \quad (5)$$

Dans la section suivante, nous présentons notre algorithme génétique pour la résolution de la formule (5). Nous montrerons ensuite par une étude comparative l'intérêt de l'utilisation de la formule (5) et des dépendances transitives entre caractéristiques.

5.3.2 L'algorithme génétique basé sur les dépendances transitives

Dans cette sous-section, nous introduisons l'algorithme génétique qui résout l'ensemble des équations de la formule (5) et dont la structure générale se présente comme suit :

Algorithme 5.1. L'algorithme génétique pour la prise de décision

Paramètre d'entrée : $FMEncoder$

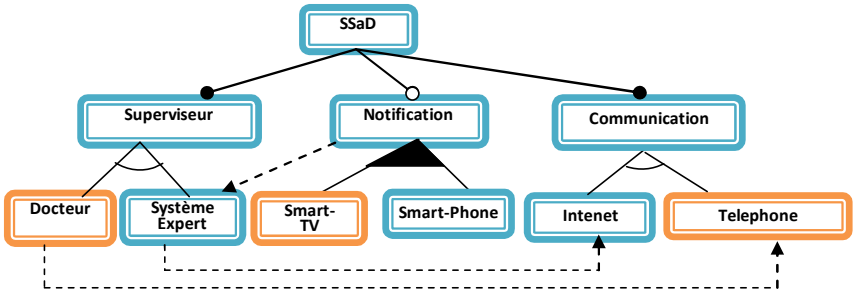
```
debut
    pour(i de 1 a tailleDePopulation)
        individu ← genererIndividuAvecSelectionTransitive()
        ajouteraLaPopulation(individu)
    finPour
    Tantque(nonFini)
        (P1, P2) ← selectionnerParrents
        fils ← Croisement(P1, P2);
        fils ← MutationAvecSelectionTransitive(fils);
        MetreaJourPopulationAvec(fils)
    FinTantque
    Retourner MieulleurIndividu(Population)
Fin
```

La principale nouveauté de l'algorithme 5.1 réside dans les opérations d'initialisation de la population et de mutation des individus. En effet, ces deux opérations sont basées sur les opérateurs de sélection transitive présentées dans le chapitre précédent. Alors que nous décrivons les différentes opérations de notre algorithme dans les paragraphes suivants, l'accent sera surtout porté sur ces deux aspects majeurs.

Encodage des solutions

L'encodage représente le mécanisme par lequel l'algorithme génétique représente les solutions potentielles. Dans notre cas, une solution consiste en une combinaison de caractéristiques à inclure ou exclure dans la solution afin d'optimiser la fitness du système. A cet effet, l'algorithme génétique utilise des chromosomes binaires de taille N (Ou N est le nombre de Fonctionnalités i.e. $N = cardinalite(F)$). Chaque gène i du chromosome indique si la caractéristique correspondante f_i sera inclue dans la solution ($gène[i] = 1$) ou exclue de la solution ($gène[i] = o$) représenté par ce chromosome. La figure 5.2 illustre l'encodage des solutions pour l'exemple d'un système de soins à domicile simplifié (SSaD). Les boites bleues représentent des caractéristiques qui seront inclues dans la configuration alors que les boites oranges représentent celle qui ne le seront pas.

Initialisation



Nom de la caractéristiques	SSaD	Superviseur	Système Expert	Docteur	Notification	SmartTV	Smart Phone	Commun ication	Internet	Téléphone
Indice	1	2	3	4	5	6	7	8	9	10
Le Chromosome	1	1	1	0	1	0	1	1	1	0

Figure 5.2. L'encodage des solutions dans notre algorithme génétique

L'algorithme génétique commence par générer une population initiale d'individus d'une manière pseudo-aléatoirement ; certains gènes sont choisis aléatoirement et sont affectés des valeurs binaires aléatoires. D'autres gènes sont positionnés de manière déterministe en fonction de la valeur des gènes précédents. Ceci est fait grâce aux opérateurs de sélection transitive afin de garantir la validité des solutions générées. La procédure d'initialisation est représentée par l'algorithme 5.2

Algorithme 5.2 : Génération de la population Initiale.

Paramètre d'entrée : $FMEncoder$

```
initialiserPopulation()
debut
    tant que le nombre d'individu < K
        tant qu'il persiste des gènes non positionnées
            Choisir un au hasard, soit le gène d'indice i
            générer une valeur binaire aléatoire, v.
            si v = 1,
                "positionne gène i a 1 et positionner les gènes dépendants en conséquence"
                selectionPseudoAleatoire(FMEncoder[i])
            sinon
                "positionne gène i a 0 et positionner les gènes dépendants en conséquence"
                deselectiontransitive(FMEncoder[i])
        FinTantque
    FinTantque
fin
```

Sélection

La sélection des individus est basée sur leurs fitness. La fitness est une valeur dépendant du

contexte d'exécution et reflétant la satisfaction de l'utilisateur. La valeur de la fitness est calculée sur la base des caractéristiques sélectionnées dans la solution. Une méthode courante consiste à associer une valeur d'utilité (ou coût) a chaque caractéristique. Cette valeur, dépend généralement du contexte (la surchauffe de l'pareil mobil ou sa connectivite) mais peut aussi être une valeur statique (La consommation d'énergie).

Il existe dans la littérature, plusieurs stratégies de sélection. Pour notre algorithme nous avons choisis la sélection par tournoi qui consiste à sélectionner le meilleur individu dans un ensemble de *M* individus aléatoirement pris dans la population. Le choix de la stratégie de sélection peut toutefois être différent. Une étude plus approfondie, permettrais de savoir la meilleure stratégie possible.

Évolution

L'évolution est assurée par les opérateurs de croisement et de mutation. Nous avons utilisé pour l'algorithme proposé le croisement uniforme (Harman et al. 2014). Il est toutefois possible d'utiliser n'importe lequel des autres opérateurs dans la littérature dans la mesure où le croisement n'est pas tenu de générer des individus valides car le nouvel individu sera altéré par l'opérateur de mutation.

Pour la mutation en revanche, Nous avons défini un nouvel opérateur basé sur l'opérateur de sélection transitive de caractéristiques. Le rôle de la mutation est de i) Améliorer les capacités d'exploration de l'algorithme génétique en évitant les optimum locaux ii) garantir que les nouveaux individus sont conforme vis a vis des contraintes du modèle de caractéristique. Ce nouvel opérateur de mutation est inspiré de la mutation uniforme. Son comportement est représenté par l'algorithme 5.3.

Algorithme 5.3 : L'operateur de Mutation des individus

Paramètre d'entrée : $FMEncoder$

```
MuterIndividu ()
debut
   Tant qu'il persiste des gènes non positionnées par l'opérateur de mutation.
      Choisir un gène au hasard soit i l'indice de ce gene.
      Tirer une valeure aleatoire R
      "inverser la valeur de gene i avec une probabilite M_rate"
      Si R > M_rate
         si gene[i] = 1
            deselectionTransitive(FMEncoder[i])
         Sinon
            selectionPseudoAleatoire(FMEncoder[i])
         finSi
```

```
        finSi
    finTantQue
fin
```

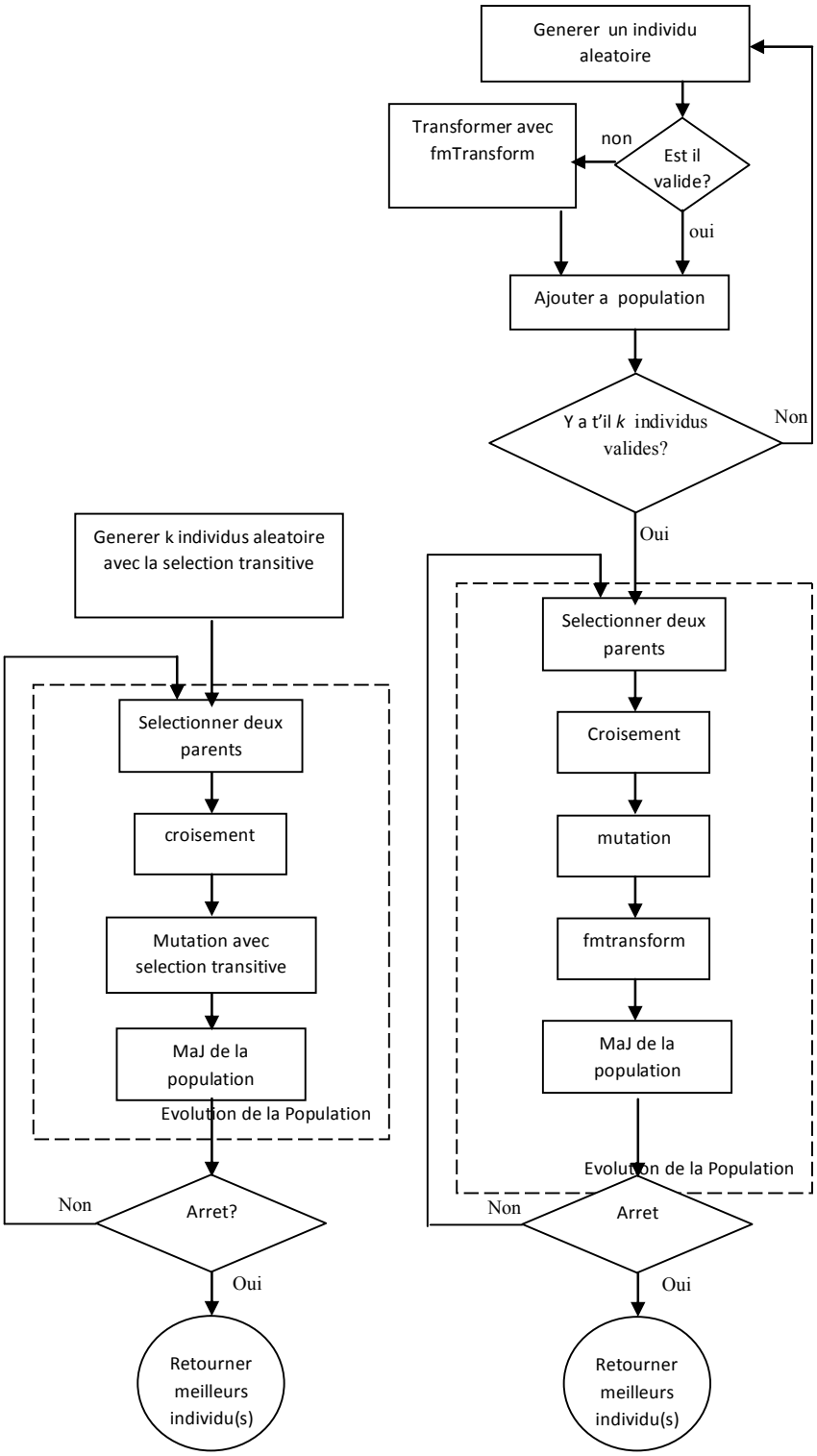
Critère d'arrêt

Le critère d'arrêt peut être un nombre maximum de génération, une durée déterminée de temps ou un indicateur de stagnation de la recherche.

5.3.3 Évaluation de l'algorithme

Afin d'évaluer l'algorithme proposé, nous avons conduit un certain nombre d'expériences sur notre algorithme génétique et comparer ses performances à l'algorithme présenté dans (Guo et al. 2011) et étendu pour l'adaptation à l'exécution dans (Pascual et al. 2015a). Notre objectif est de montrer que notre algorithme génétique peut délivrer des solutions d'une qualité équivalente à celles produites par l'algorithme de (Guo et al. 2011) et (Pascual et al. 2015a) en un temps plus court. Ainsi, nous comparons notre algorithme avec l'algorithme de (Guo et al. 2011) et (Pascual et al. 2015a) selon les trois critères qui sont la valeur de la meilleure *fitness* trouvée, le nombre de combinaisons explorées et le temps d'exécution.

D'abord, comparant les deux algorithmes génétiques sur le plan structurel. La figure5.3 illustre la différence entre la structure de notre algorithme (figure 5.3.a) et celle de la littérature (figure 5.3.b)



(a) notre Algorithme Genetique

(b) Algorithme Genetique de la litterature

Figure 5.3. Structure de notre algorithme génétique vs celle de l'algorithme génétique de la littérature

Ainsi qu'il peut être constaté à partir de la figure 5.3.b, l'algorithme de Guo et al et Pascual et al, est basé sur L'opérateur $fmTransform$ qui transforme un individu aléatoire en un individu valide. Cet opérateur utilise les deux opérations de sélection ($includeFeature$) et désélection ($excludeFeature$) représentés par les algorithmes 5.4 et 5.5.

Algorithme 5.4. La sélection par l'opérateur $fmtransform$

Paramètres d'entrée : $\boldsymbol{IndividuAleatoire}$
$\boldsymbol{includeFeature}(f)$
debut
si f *n'estpas la racine alors*
$\boldsymbol{includeFeature}$ $(parent(f))$.
finsi
si f *est dans un groupe* XOR, *alors*
$\forall e \in groupeXor$ $\boldsymbol{excludeFeature}$ (e).
finsi
si f *possede des sous caracteristiques obligatoires alors*
$\forall\ e \in sousCaracteristiquesObligatoire$ $\boldsymbol{includeFeature}$
finsi
si $\exists\ e, f \Longrightarrow e$ *alors*
$\boldsymbol{includeFeature}$ (e)
$finsi.$
si $\exists\ e, f \Longrightarrow |e$ *alors*
$\boldsymbol{excludeFeature}$ (e)
finsi
si f *possede des* $sousCaracteristiques$ *alors*
$e = choisirSousCaracteristique(f)$
$\boldsymbol{includeFeature}(e)$
$finsi$
fin

Algorithme 5.5. La désélection par l'opérateur $fmtransform$

Paramètres d'entrée : $\boldsymbol{IndividuAleatoire}$
$\boldsymbol{excludeFeature}(f)$
debut
$\forall e \in sousCaracteristiques(f)$ $\boldsymbol{excludeFeature}$ (e).
Si f est une sous caractéristiquesobligatoire de e alors
$\boldsymbol{excludeFeature}$ (e).
finsi
si $\exists e, e \Longrightarrow f$ *alors*
$\boldsymbol{excludeFeature}$ (e).
finsi
fin

Il est claire que les opérations $includeFeature$ et $excludeFeature$ sont lourdement récursives. Au contraire les opérations de sélection et désélection transitive le sont très peu. Notre intuition est donc que notre algorithme sera capable d'évoluer plus rapidement que celui de la littérature. Pour s'en assurer, nous avons mené nos expériences sur des modèles de caractéristiques de tailles différentes. Les modèles de caractéristiques sont générés suivant la méthode présentée dans (Thüm et al. 2009) afin de refléter des modèles de caractéristiques du monde réel. La génération du problème d'optimisation a été faite selon la méthode présentée dans (Pascual et al. 2015a).

Les différents algorithmes qui ont servi aux expériences ont été implémentés en Pharo, un dialecte Smalltalk avec des facilités de prototypage et d'analyse[1] [2]. Les expériences ont été menées sur une machine MacBook Pro avec un processeur Intel Core 2 Duo d'une fréquence de 2.66 GHz et disposant de 8Go de RAM.

5.3.3.1 Génération des modèles de caractéristiques et des problèmes d'optimisation

Nous avons implémenté l'algorithme de Thum et al pour la génération de modèles de Fonctionnalités aléatoires (Thum et al. 2009). Cet algorithme permet de générer des FM semblables à ceux rencontrés dans le monde réel. Leurs caractéristiques sont illustrées par le tableau 5.1 :

[1] Le code est en access libre sur smalltalkhub.com/mc/Alidra/BitSetFMs/main/ et smalltalkhub.com/mc/Alidra/geneticAlgorithmFamily2ProductDerivation/main/

[2] www.Pharo.org

Tableau 5.1 caractéristiques des modèles de caractéristiques pour l'évaluation de l'algorithme génétiques	
Propriété du modèle de caractéristique	**Valeur**
Nombre maximum de sous-caractéristiques	10
Ratio groupes ET	0.25
Ratio des caractéristiques optionnelles dans les groupes ET	0.25
Ratio des groupes OR	0.25
Ratio des groupes XOR	0.25
Ratio des contraintes extra-structurelles sur le nombre de caractéristiques	0.1
Ratio des contraintes implique	0.5
Ratio des contraintes exclut	0.5

Pour la génération des problèmes d'optimisation, nous avons associé des valeurs aléatoires dans l'intervalle *minVal*, *MaxVal* à chacune des caractéristiques du modèle. La fitness d'une solution est alors la somme des valeurs utilités de toutes les caractéristiques sélectionnées dans cette solution.

5.3.3.2 Discussion des résultats

Nous avons configuré les deux algorithmes à comparer avec les paramètres du tableau 5.2 :

Tableau 5.2 les paramètres des algorithmes génétiques

Paramètre	**Valeur**
Taille de la population	200
Probabilité de Mutation (M_{rate})	0.1
Critère d'arrêt	nombre de générations < 200
Taux de croisement	0.5

Les trois critères de comparaison des algorithmes sont les suivants :

- Le ratio des *fitness* de la meilleure solution : fitness de la meilleure solution trouvée par notre algorithme/*fitness* de la meilleure solution trouvée par l'algorithme de (Guo et al. 2011)
- Le temps d'exécution des algorithmes génétiques
- Le ratio des nombre des combinaisons parcourues par les algorithmes génétique : le nombre de configurations explorées par notre algorithme/le nombre de configurations

explorées par l'algorithme de (Guo et al. 2011).

Il est important de noter que notre objectif n'est pas d'évaluer l'intérêt d'utiliser une politique de prise de décision basée sur la fitness. En effet, ce point a été largement démontré par une variété de travaux tel que Madame (Geihs et al. 2009), Music (Rouvoy et al. 2009) (Svein et al. 2009) (Jiang et al. 2010) (Perrouin et al. 2008). et autres approches (Guo et al. 2011) et (Pascual et al. 2015a). Notre objectif n'est pas non plus d'estimer la capacité de l'algorithme génétique proposé à générer des solutions proche de l'optimale. Nous nous contentons à cet effet de comparer la qualité des solutions trouvées par notre algorithme à celles trouvées par l'algorithme de (Guo et al. 2011) et (Pascual et al. 2015a) dont les auteurs ont montré qu'elles étaient supérieures à 87% d'optimalité. Par contre, nous montrons que notre algorithme peut parcourir l'espace de recherche sans altérer les capacités d'exploration ni la qualité des solutions trouvées en un temps considérablement plus court.

5.3.3.3 Etude Comparative

Les tableaux 5.3 et 5.4 représentent les résultats expérimentaux de l'AG de Guo et al, et notre propre AG respectivement. Ces résultats consistent en des temps moyens d'exécution, le nombre de solutions explorées et les meilleures valeurs de fitness. Ces valeurs ne sont pas significatives en elles-mêmes car elles sont appliquées à des problèmes aléatoires. Ainsi, nous synthétisons les résultats tableau 5.3 et 5.4 dans le tableau 5.5 pour comparer les deux algorithmes.

Le tableau 5.5 montre les résultats expérimentaux obtenus à partir de la comparaison des différents essais des deux algorithmes génétiques présentés ci-dessus sur des modèles de caractéristiques de différentes tailles. La première colonne du tableau 5.5 montre la taille (nombre de caractéristiques) des modèles étudiés. La colonne 2 indique le gain du temps d'exécution conséquent à l'utilisation de notre algorithme génétique par rapport à celui de (Guo et al. 11). La colonne 3 indique le rapport entre le nombre de combinaisons explorées des deux algorithmes et de la colonne 4, le rapport de leurs meilleures fitnesses comme expliqué par les formules suivantes:

- $Colonne\ 2\ =$
  $(exécution\ Temps\ de\ Guo\ et\ al\ GA\ -\ Temps\ d'exécution\ de\ notre\ GA)\ /$
  $exécution\ Temps\ de\ Guo\ et\ al\ GA.$
- $Colonne\ 3\ =$
  $(nombre\ de\ solutions\ explorées\ par\ Guo\ et\ al\ GA)\ /$
  $(nombre\ de\ solutions\ explorées\ par\ notre\ GA)$
- $Colonne\ 4\ =$

$$(Moyenne\ des\ valeurs\ de\ fitness\ des\ meilleures\ solutions\ trouvées\ par\ Guo\ et\ al\ ) /$$
$$(Moyenne\ des\ valeurs\ de\ fitness\ des\ meilleures\ solutions\ trouvées\ par\ notre\ GA)$$

Nous avons répété chaque expérience 100 fois et calculé la moyenne de chaque valeur. D'après le tableau 5.5, on peut clairement remarquer que notre algorithme génétique évolue plus rapidement que celle de (Guo et al. 11) sans aucun impact significatif sur les performances par rapport à celui-ci. En effet, les rapports des meilleures valeurs de fitness, ainsi que le rapport entre le nombre de combinaisons étudiées oscillent autour de 1, et la valeur moyenne de variation est inférieure à 0,01.

D'un autre côté, il est évident à partir du tableau 5.5 que l'algorithme génétique exploitant les dépendances entre les caractéristiques converge en moins de temps que celui de (Guo et al. 11). En outre, la différence entre les deux temps d'exécution est proportionnel au nombre de modèles de caractéristiques et notre algorithme peut être jusqu'à 99% plus rapide que celui de la littérature.

Tableau 5.3. *Résultats Expérimentaux de l'AG de (Guo et al., 2011)*

Taille du modèle de caractéristiques	Temps moyen d'exécution (en seconds)	Nombre moyen des combinassions explorées	moyenne des meilleures valeurs fitness
100	8	227,542	79,747
200	32	299,300	83,300
500	165	164,756	176,670
800	634	299,557	231,000
1000	682	300,000	192,667
2000	2056	300,000	224,000

Tableau 5.4 *Résultats Expérimentaux de l'AG proposé*

Taille du modèle de caractéristiques	Temps moyen d'exécution (en seconds)	Nombre moyen des combinassions explorées	moyenne des meilleures valeurs fitness
100	1	214,699	84,472
200	2	299,580	79,901
500	4	159,822	171,058
800	14	299,626	244,129
1000	15	300,000	213,000
2000	26	300,000	265,667

Tableau 5.5*. Comparaison des résultats Expérimentaux*

Taille du modèle de caractéristiques	Gain de temps moyen d'exécution	Ratio des combinassions explorées	Ratio des meilleures valeurs fitness
100	88%	1,059	0,944
200	94%	0,999	1,042
500	98%	1,030	1,032
800	98%	0,999	0,946
1000	98%	1,000	0,904
2000	99%	1,000	0,843

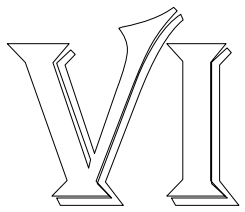
5.4 Conclusion

Dans ce chapitre, nous avons présenté notre approche évolutionnaire pour l'adaptabilité des systèmes logiciels. Nous avons mis l'accent sur la phase de prise de décision dont nous avons précédemment identifié l'importance. Dans notre travail, cette phase est basée sur un algorithme génétique qui exploite les relations de dépendances transitives introduis plus haut dans cette thèse. Afin de montrer l'intérêt de notre approche, nous avons comparé notre algorithme génétique à l'AG existant dans la littérature. Les résultats numériques montrent un gain de temps qui peut être supérieur a 90% dans le cas de système de moyenne taille et un gain de temps qui avoisine les 99% dans le cas de système de grande a très grande taille. A partir de cela nous concluons que notre algorithme est plus adaptée pour la planification de l'adaptation des systèmes embarqués modernes car :

1- Il a une complexité de calcul significativement réduite et qui n'est pas exponentiellement corrélée au nombre de caractéristiques du système.
2- Il exige une puissance de calcul réduite relativement a d'autre techniques similaire notamment celles basées sur les solveurs logiques et particulièrement comparativement a la technique de référence rapportée plus haut.
3- Il fournie une réponse proche de l'optimale dans des délais significativement plus court que la technique de référence.
4- Il permet une meilleure mise à l'échelle que l'approche de référence. Ceci est d'autant plus important que les systèmes embarqués modernes connaissent une évolution rapide de leur taille et de leur complexité.

Dans le prochain chapitre, nous présentons une extension de notre algorithme génétique pour le problème délicat de l'auto-guérison dans les systèmes logiciels embarqués adaptables.

Chapi



Adaptation de l'approche évolutionnaire au problème de l'auto-guérison

6.1 Introduction

Les systèmes logiciels moderne évoluent dans des contextes instables, ouverts et dynamiques où les composants peuvent dysfonctionner ou se déconnecter. Ces systèmes doivent adapter leurs comportements pour réparer (ou au moins atténuer) la perte de composants afin de continuer à fournir leurs services. Cette propriété d'adaptation face à une perte ou un disfonctionnement de composant est qualifiée d'auto-guérison. De nombreuses approches ont été proposées dans la littérature pour réaliser efficacement l'auto-guérison. Cependant, la plupart de ces approches gèrent difficilement les scénarios imprévus et/ou les contraintes de temps de réponse. Pour faire face à ces limitations, nous ajustons l'algorithme génétique introduit dans le chapitre précédent au problème spécifique de l'auto-guérison sur la base d'une reformulation du problème de la prise de décision. En plus, afin de réduire le temps de réponse du processus de prise de décision en cas de panne, la défaillance des composants est anticipée par la simulation de la perte du composant dans le calcul de l'algorithme génétique. Une machine d'état finie est alors construite à partir des prévisions de défaillance et sert à fournir d'une manière quasi-instantanée le plan de reconfiguration le plus approprié par rapport à l'environnement courant et aux exigences changeantes des utilisateurs.

6.1 Aperçu de l'approche

L'auto-guérison est l'un des aspects important du comportement autogéré des systèmes autonome. L'auto-guérison désigne la capacité d'un système autonome à recouvrir ou du moins

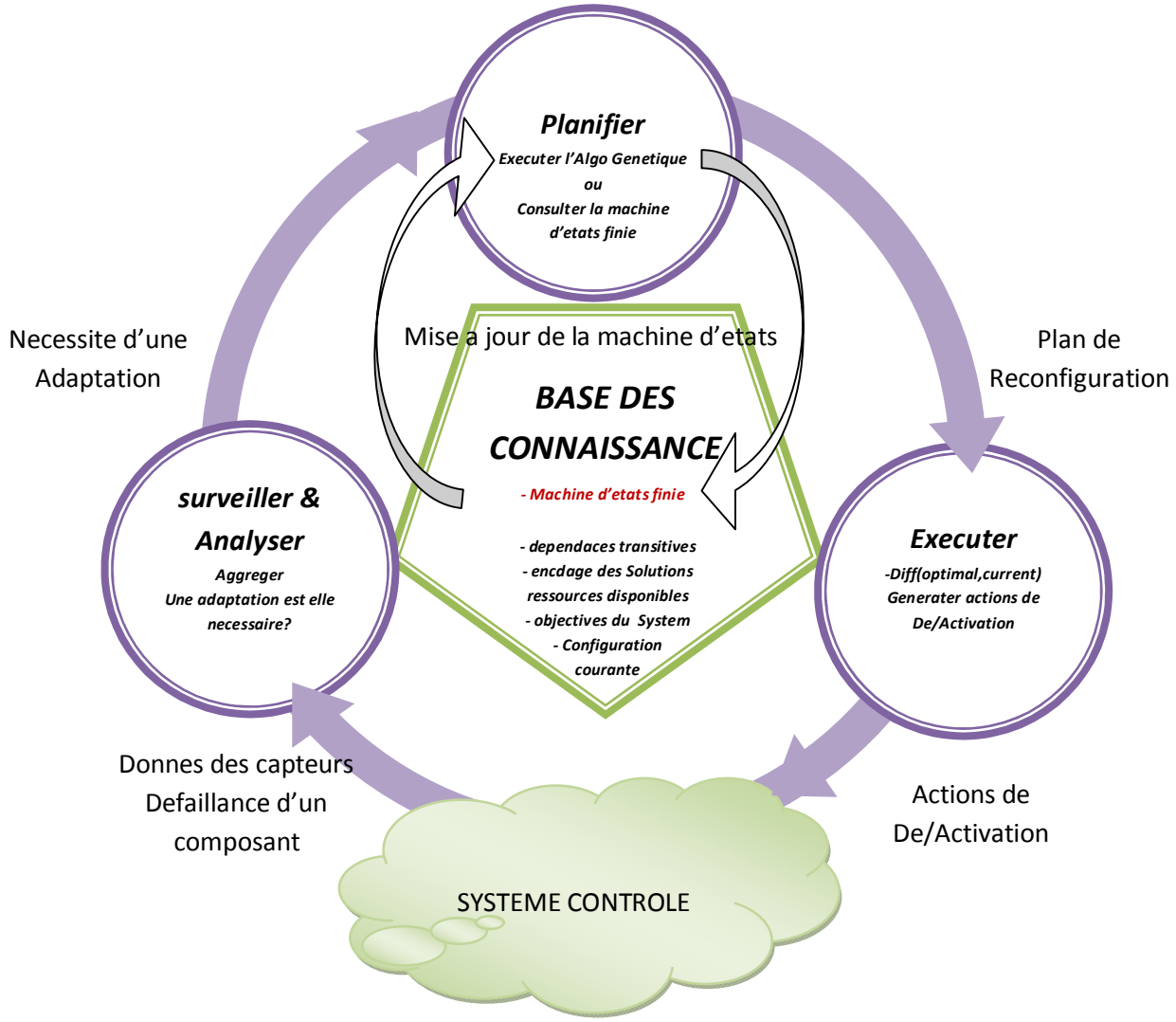


Figure 6.1 Aperçus de l'approche évolutionnaire pour l'auto-guérison

mitiger la perte, la défaillance ou le dysfonctionnement de l'un de ses composants (Cetina et al. 2009). Comme dans le cadre globale de l'autogestion, l'auto-guérison implique trois capacités : la surveillance, la prise de décision et la reconfiguration. Nous nous concentrons dans le présent chapitre sur la prise de décision en supposant l'existence d'une infrastructure de surveillance et de reconfiguration à l'exécution. Pour rappel, la surveillance (ou monitoring) vise à collecter les donnes sur l'environnement et les rapporter au processus de prise de décision. L'infrastructure de reconfiguration exécute les plans de configuration grâce à l'utilisation d'un pilote d'adaptation. Entre les deux, la prise de décision analyse les données de surveillance et établit les plans de reconfiguration en conséquence.

Afin d'unir l'effort pour la réalisation de l'adaptation et de l'auto-guérison, nous proposons d'étendre l'algorithme génétique présenté dans le chapitre précédent afin de i) trouver un plan de reconfiguration qui maintient un niveau de qualité de service proche de l'optimal même en cas de la défaillance d'un composant du système. ii) fournir le plan de fonctionnement dans les délais impartis.

À cette fin, la stratégie suivante sera adoptée:

- Exécution de l'algorithme génétique en anticipant la défaillance de certains composants actifs présentement ou à l' avenir.
- Construire une machine d'états finie qui spécifie les configurations futures en fonction de la défaillance de composants
- Maintenir à jour la machine d'état à chaque reconfiguration du système ou changement des paramètres d'exécution
- En cas de défaillance d'un composant, trouver la reconfiguration directement à partir de la machine d'états finie.

Le schéma de notre approche est représenté par la figure 6.1 et sera expliqué tout au long de ce chapitre. Par ailleurs, nous considérons, l'exemple du système de soin a domicile (SSaD) présenté au chapitre 5. Nous avons légèrement modifié la spécification de ce système pour une meilleure illustration du propos. Le modèle de caractéristique correspondant est représenté par la figure 6.2.

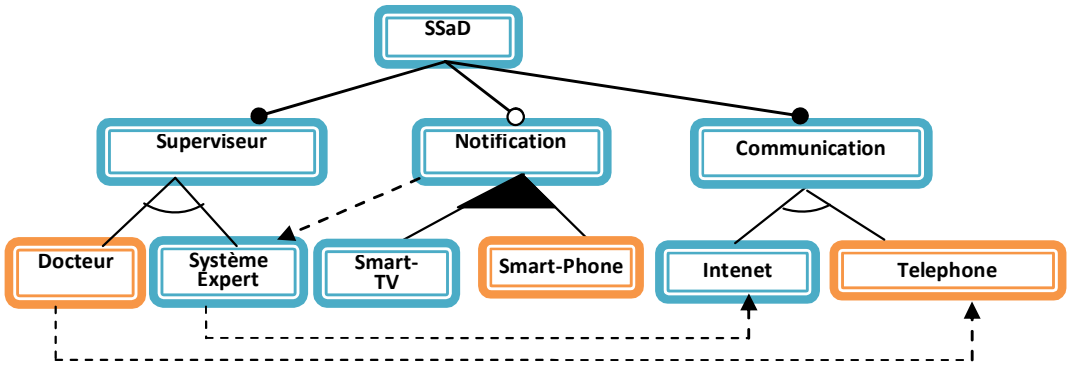


Figure 6.2 l'exemple modifié du système d'aide a la sante a domicile (SSaD)

6.2 Une approche évolutionnaire pour l'auto-guérison

Dans le chapitre précédent nous avons proposé un algorithme génétique qui explore l'espace des configurations du modèle de caractéristiques afin de sélectionner la meilleure configuration

compte tenu du contexte d'exécution et des attentes spécifiques des utilisateurs. Dans les prochains paragraphes nous adaptons l'approche proposée au contexte particulier de l'auto-guérison.

6.2.1 Reformulation du problème de la prise de décision pour l'auto-guérison

Nous reprenons la formulation que nous avons propose dans le chapitre 5 pour le problème de la prise de décision dans les systèmes adaptables en rajoutant l'information sur la disponibilité des composants du système. Ainsi la formule (5) qui été initialement :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(X)}\right) \\ X \in \{0,1\}^{card(F)}\ et \\ x_i^c = 1 \Longrightarrow \begin{cases} \forall j \in [1, card(F)], si\ FMEncoder[f_i].inferences[j] = 1 \Longrightarrow x_j^c = 1 \\ \forall j \in [1, card(F)], si\ FMEncoder[f_i].exclusions[j] = 1 \Longrightarrow x_j^c = 0 \\ \exists j \in [1, card(F)], FMEncoder[f_i].descendants[j] = 1 \Longrightarrow x_j^c = 1 \end{cases} \\ x_i^c = 0 \Longrightarrow \forall j \in [1, card(F)], si\ FMEncoder[f_i].inferencesInverse[j] = 1 \Longrightarrow x_j^c = 0 \end{cases} \quad (5)$$

Devient-elle :

$$\begin{cases} optimiser\left(\overrightarrow{\Theta_{context}(X)}\right) \\ X \in \{0,1\}^{card(F)}\ et \\ si\ f_i \in Defaillantes \Longrightarrow x_i = 0\ et \\ x_i = 1 \Longrightarrow \begin{cases} \forall j \in [1, card(F)], si\ FMEncoder[f_i].inferences[j] = 1 \Longrightarrow x_j^c = 1 \\ \forall j \in [1, card(F)], si\ FMEncoder[f_i].exclusions[j] = 1 \Longrightarrow x_j^c = 0 \\ \exists j \in [1, card(F)], FMEncoder[f_i].descendants[j] = 1 \Longrightarrow x_j^c = 1 \end{cases} \\ x_i = 0 \Longrightarrow \forall j \in [1, card(F)], si\ FMEncoder[f_i].inferencesInverse[j] = 1 \Longrightarrow x_j^c = 0 \end{cases} \quad (6)$$

Où $Defaillantes$ représente l'ensemble des caractéristiques correspondant aux composants défaillants détectés par l'infrastructure de monitoring (nous dirons pour simplifier : les caractéristiques défaillantes).

La formule 6 exprime explicitement le besoin d'écarter de l'espace de recherche, les configurations qui incluent des caractéristiques correspondantes aux composants défaillants. Ainsi, ses caractéristiques sont elle explicitement désélectionnées de toutes les solutions envisageables. Les dépendances transitives sont là encore d'un intérêt majeur dans la mesure où elles permettent de propager la désélection des caractéristiques défaillantes et ainsi non seulement garantir que les solutions trouvées ne requièrent pas des composants défaillants, mais aussi réduisent significativement l'espace de recherche des solutions. Dans la section suivante, nous exploitons cette nouvelle formulation de la prise de décision par la transformation de notre algorithme génétique pour la prise de décision dans les systèmes adaptables.

6.2.2 Ajustement de l'algorithme génétique basé sur les dépendances transitives pour

le problème de l'auto-guérison

Nous maintenons pour notre algorithme génétique la même structure d'ensemble explicitée par l'algorithme 5.1 et reprise par l'algorithme 6.1. Un point de différence important est toutefois le passage en paramètres d'entrée de l'ensemble des caractéristiques défaillantes qui doivent être écartées des solutions générées par l'algorithme génétique.

Algorithme 6.1. L'algorithme génétique pour la prise de décision pour l'adaptabilité et l'auto-guérison

Paramètre d'entrée : $FMEncoder, \boldsymbol{Defaillantes}$

$debut$
  $pour(i\ de\ 1\ a\ tailleDePopulation)$
    $individu \leftarrow genererIndividuAvecSelectionTransitive()$
    $ajouteraLaPopulation(individu)$
  $finPour$
  $Tantque(nonFini)$
    $(P1, P2) \leftarrow selectionnerParrents$
    $fils \leftarrow Croisement(P1, P2);$
    $fils \leftarrow MutationAvecSelectionTransitive(fils);$
    $MetreaJourPopulationAvec(fils)$
  $FinTantque$
  $Retourner\ MieulleurIndividu(Population)$
Fin

De même, nous préservons l'encodage des solutions présenté à la section 5.5. Ainsi les chromosomes, représentent les caractéristiques actives/inactives dans le modèle de caractéristiques. Autrement dit, pour un modèle de caractéristiques $FM =< G, r, E_{MAND}, F_{XOR}, F_{OR}, Impl, Excl >$, un chromosome binaire de taille $N = card(F)$ est créé. Par ailleurs, chaque gène du chromosome indique si la caractéristique correspondante est sélectionnée ou non.

6.2.2.1 Initialisation de la population

Cette opération consiste à générer K individus aléatoires qui constituerons la population initiale de l'algorithme génétique et évoluerons progressivement vers les solutions optimales. La procédure de génération des individus aléatoires est basée sur les operateurs de sélection transitive afin de garantir leur validité. De plus, les caractéristiques défaillantes sont écartées de l'espace de recherche par le positionnement des gènes correspondants à zéro (ainsi que les gènes dépendants grâce à la désélection transitive) dés le début de la procédure de génération. Cette opération est illustrée par l'algorithme 6.2

Algorithme 5.2 : Génération de la population Initiale.

Paramètres d'entrée : $FMEncoder, Defaillantes$
$initialiserPopulation()$
$debut$
$tant\ que\ le\ nombre\ d'individu\ <\ K$
$Pour\ chaque\ caractéristique\ f_i \in Defaillantes$
$deselectiontransitive(FMEncoder[i])$
$finPour$
$tant\ qu'il\ persiste\ des\ gènes\ non\ positionnées$
$Choisir\ un\ au\ hasard, soit\ le\ gène\ d'indice\ i$
$générer\ une\ valeur\ binaire\ aléatoire, v.$
$si\ v = 1,$
$"positionne\ gène\ i\ a\ 1\ et\ positionner\ les\ gènes\ dépendants\ en\ conséquence"$
$selectionPseudoAleatoire(FMEncoder[i])$
$sinon$
$"positionne\ gène\ i\ a\ 0\ et\ positionner\ les\ gènes\ dépendants\ en\ conséquence"$
$deselectiontransitive(FMEncoder[i])$
$FinTantque$
$FinTantque$
fin

6.2.2.2 Evolution

Ainsi que nous l'avons expliqué dans le chapitre précédent, l'évolution est assurée par les opérateurs de croisement et de mutation. Alors que l'opérateur de croisement n'est soumis a aucune contrainte particulière dans la mesure où le croisement n'est pas tenu de générer des individus valides car le nouvel individu sera altéré par l'opérateur de mutation (Il est donc possible d'utiliser n'importe lequel des opérateurs dans la littérature), la mutation en revanche doit être redéfini pour respecter les contraintes du modèle de caractéristiques et la disponibilité des ressources conséquentes au défaillance de composants, Nous avons donc modifier l'opérateur de mutation basé sur l'opérateur de sélection transitive de caractéristiques introduit plus haut pour la prise en compte de ce dernier paramètre. Le code de ce nouvel opérateur est représenté par l'algorithme 6.3

Algorithme 6.3 : L'operateur de Mutation des individus

Paramètres d'entrée : $FMEncoder, Defaillantes$
$MuterIndividu\ ()$
$debut$
$Pour\ chaque\ caractéristique\ f_i \in Defaillantes$
$deselectiontransitive(FMEncoder[i])$
$finPour$

$Tant\ qu'il\ persiste\ des\ gènes\ non\ positionnées\ par\ l'opérateur\ de\ mutation.$

```
        Choisir un gène au hasard soit i l'indice de ce gene.
        si f_i ∈ Defaillantes alors
            deselectiontransitive(FMEncoder[i])
        sinon
            Tirer une valeure aleatoire R
            "inverses la valeur de de gene i avec une probabilite M_rate"
            Si R > M_rate
                si gene[i] = 1
                    deselectionTransitive(FMEncoder[i])
                Sinon
                    selectionPseudoAleatoire(FMEncoder[i])
                finSi
            finSi
        finSi
    finTantQue
fin
```

Ainsi, notre nouvel algorithme génétique est capable de traiter naturellement le cas de l'auto-guérison dans les systèmes adaptables. Dans la section suivante, nous allons chercher à améliorer les performances de l'infrastructure décisionnelle par l'exploitation de la structure d'une machine d'état afin d'anticiper la défection des composants du système en pré-calculant les configurations alternatives pour y remédier.

6.3 Anticipation de la défaillance des composants dans le calcul des configurations alternatives

Bien que l'utilisation combinée d'un algorithme évolutionnaire et de la notion de variabilité liée dont découlent les relations de dépendances transitives apporte un bénéfice certain en termes de complexité de calcul et de temps de réponse du processus de prise de décision, certain systèmes embarqués, peuvent imposer des contraintes encore plus strictes sur le temps de réponse. Ainsi par exemple les systèmes critiques, temps réel ou les systèmes évoluant dans des environnements particulièrement instables. Dans ce cas, la réponse a la défaillance d'un composant doit être quasi instantanée. C'est dans ce sens que nous proposons d'anticiper la défaillance des composants en calculant des configurations alternatives qui y remédient. A cette fin, nous exploitons une machine d'états finie qui présente l'avantage d'offrir une structure simple et des algorithmes de parcours efficaces pour trouver rapidement la reconfiguration qui correspond a l'échec d'un ou de plusieurs composants.

Ainsi, afin de construire la machine d'état finie, nous allons exécuter l'algorithme génétique avec comme paramètre l'ensemble $defaillantes$ qui contiendra non seulement les caractéristiques que la plateforme de monitoring a signalé comme défaillantes mais également la ou les caractéristiques dont nous voulons anticiper la panne. Aussi, le processus de recherche de solutions est il orienté vers des zones où des composants spécifiques sont explicitement désactivés simulant une défaillance. Par conséquence, nous obtenons une hiérarchie d'alternatives de plan de reconfigurations qui peuvent être activé dans le cas ou un ou de plusieurs composants du système tombent en panne.

Traitement de l'incertitude des Données de surveillance

Il convient également de noter que ce processus peut être particulièrement intéressant pour remédier à l'incertitude liée aux informations d'échec transmise par l'infrastructure de monitoring. En effet, il arrive souvent que l'infrastructure de monitoring transmette des données incomplètes voir contradictoire sur l'état de fonctionnement des composant du système (Ailisto et al. 2002). En général, la procédure d'analyse synthétise ces données pour en extraire des informations pertinentes et sures. Cependant, l'analyse des données de monitoring ne permet pas toujours d'identifier avec certitude **le** composant qui est responsable du dysfonctionnement du système. Au lieu de ca, un ensemble de composants sont identifier et/ou des probabilités de défaillance sont associées a ces composants.

Dans ce sens, notre algorithme génétique permettrait de lever l'ambigüité sur la source du dysfonctionnement ou au moins, prévoir plusieurs plans d'adaptation pour répondre aux différents scenarios de défaillances rapportées par l'infrastructure de monitoring.

Un point également important est l'identification des composants dont il est pertinent de prévoir la défaillance. En effet, la prédication de la défaillance de la totalité des composants est infaisable en pratique à cause du cout induit par le calcul des configurations qui en découlent. Par vois de conséquence, il est important de déterminer les composants dont il serait judicieux de prévoir la défaillance. Par exemple, il peut être intéressant de restreindre cet ensemble aux seuls composants dont la structure de surveillance estime une haute probabilité d'échec. Une autre possibilité consisterait à utiliser un algorithme d'apprentissage qui serai en mesure d'extrapoler des scenarios de défaillance passés pour prévoir la défaillance des composants à l'avenir et qu'il serai par conséquence plus approprié d'en anticiper l'échec. Dans tout les cas, cet aspect de la complémentarité entre l'algorithme de prise de décision et l'infrastructure de monitoring et d'analyse mérite clairement d'être étudier plus en détail à l' avenir mais il dépasse le cadre de cette thèse.

6.3.1 Une machine d'états finie pour la planification efficace de l'auto-guérison

La machine d'états finie $MEF_{FM,predire} =< S, T, racine >$ sur un modèle de caractéristiques $FM =< G, r, E_{MAND}, F_{XOR}, F_{OR}, Impl, Excl >$ est définie par un ensemble d'états S, un ensemble T de transitions entre ces états et un état racine $racine \in S$ tel que :

- Un état $state \in S$ représente une configuration valide de caractéristiques actives de FM. i.e. $S \subset \mathcal{P}(F)$
- Une transition est un triplet $< state, f, state' >$, où $state\ et\ state' \in S$, $f \in predire$ et $predire \subseteq F$ un sous ensemble des caractéristiques de FM dont nous souhaitons prédire la défaillance. i.e. $T \subset S \times predire \times S$. Une transition associe une configuration cible $states'$ à l'événement de défaillance d'une caractéristique f à partir d'une configuration source $state$.
- $racine$ est la racine de la machine d'état c'est-à-dire une combinaison valide des caractéristiques de FM. i.e. $racine \in S$. Plus spécifiquement, $racine$ représente la configuration courante du système à partir de la quelle les prédictions de défaillances sont faites.

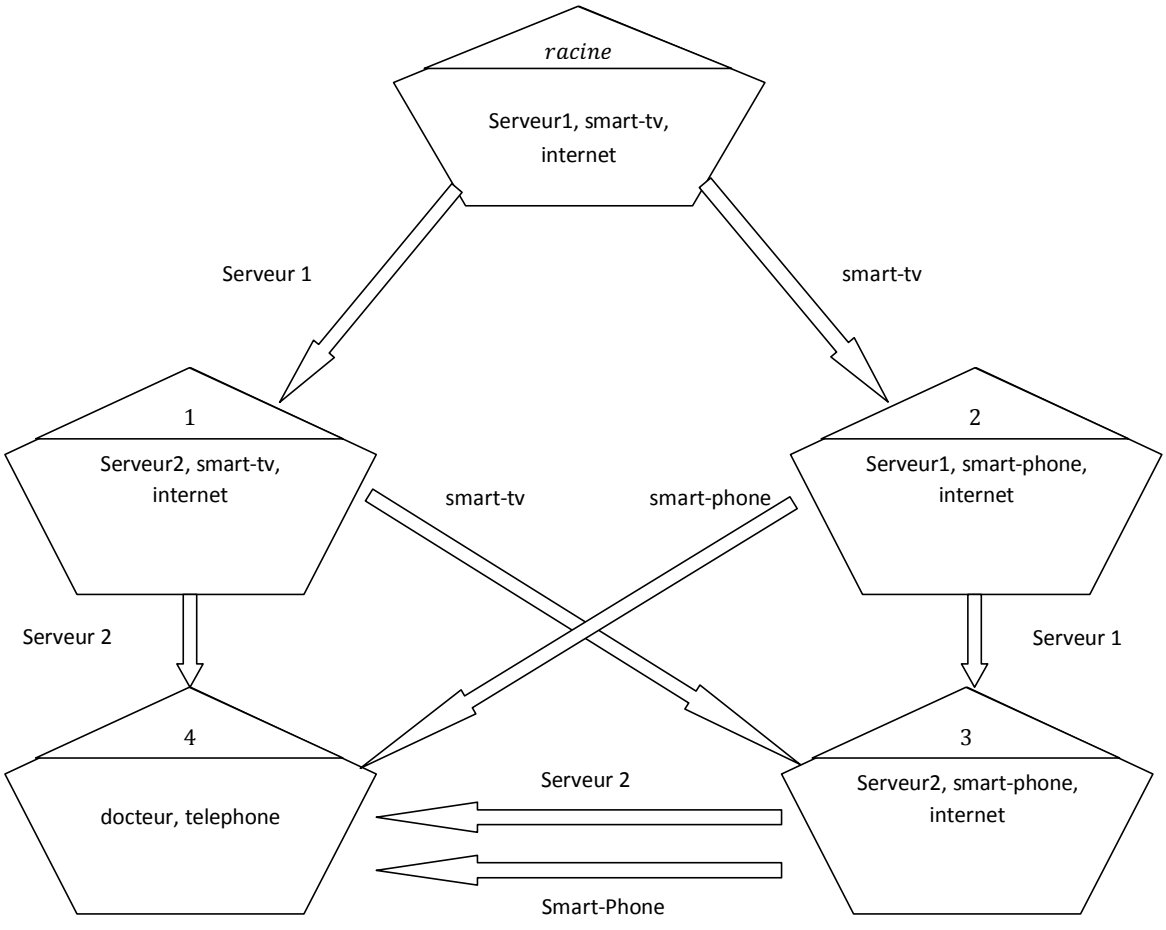


Figure 6.3 un fragment de la machine d'état de l'exemple du Système de Santé à Domicile

La machine d'état finie $MEF_{FM,predire}$ représente les évolutions possibles d'un système modélisée par le modèle de caractéristiques FM à partir de l'état actuel (tel que représente par la configuration $racine$) en cas de dysfonctionement d'un composant liée à une caractéristique f dans l'ensemble $predire$.

La figure 6.3 représente un fragment de la machine d'états finie de l'exemple du Système de Santé à Domicile introduit plus haut. Pour une meilleure lisibilité nous avons choisis de mentionner les caractéristiques finales uniquement. Cette machine d'états finie a été calculée pour l'état racine représenté dans la figure 6.2 (pour rappel les composants actifs sont représentés par des boites bleu).Donc $racine =$

$\{SSaD, Superviseur, SystemeExpert, Notification, Smart_{TV}, communication, internet\}$

Et l'ensemble des caractéristiques dont la machine anticipe la défaillance est $predire = \{serveur1, serveur2, smart_{TV}, Smart_{Phone}\}$

Ainsi, la Machine d'états finie décrit elle l'évolution du système en cas de pannes des composants $serveur1, serveur2, smart_{TV}, Smart_{Phone}$. L'exemple donné par la figure 6.2 est donné à titre illustratif et n'as aucune valeur de vérité en soi. Il illustre toutefois l'intérêt de la combinaison du calcul évolutionnaire et de la machine d'états finie de deux manières :

1. Les états cibles de l'évolution ne représentent pas seulement le remplacement d'un composant par un autre composant alternatif (ex. $serveur1$ et $serveur2$). Mais la recherche dans l'ensemble des configurations valides d'une configuration alternative (proche de l') optimale. Cela peut se traduire par la reconfiguration aux niveaux supérieurs de la caractéristique (Ainsi par exemple, dans le passage de la configuration 1 à la configuration4, les caractéristiques $serveur1$ et $serveur2$ sont remplacées par la caractéristique $docteur$ qui est leurs alternative a un niveau hiérarchique supérieur qui est le niveau de $superviseur$). Plus encore, le remplacement de la caractéristique défaillante peut avoir une incidence sur la sélection d'autres caractéristiques et doit donc être répercuter sur elles (par exemple le remplacement des serveurs du système expert par un docteur humain nécessite l'activation de la communication téléphonique à la place d'internet dans le passage de la configuration 1 à la configuration 4)

2. L'objectif de l'évolution est de trouver une configuration (proche de l') optimale en cas de l'échec d'un composant. Cela peut se traduire par la reconfiguration en profondeur des caractéristiques du système. Ainsi par exemple, la défaillance des composants de notification $Smart_{Phone}$ et $smart_{TV}$ (les états 2 et 3 vers l'état 4) n'entraine elle pas seulement la désactivation de la caractéristique $Notification$ mais implique-t-elle le

remplacement de $systèmeExpert$ par $docteur$. Cela ne peut pas s'expliqué par les contraintes du modèle de caractéristiques : il n'y a pas de lien de causalité entre notification et docteur mais la reconfiguration est motivée par l'optimisation du service fourni au malade (Ceci doit évidement être explicitement exprimé par une fonction objectif adéquate)

Dans la section suivante nous présentons notre algorithme pour la dérivation de la machine d'état finie à partir du calcul évolutionnaire sur le modèle de caractéristique du système et un ensemble de caractéristiques potentiellement défaillantes.

6.3.2 Construction de la machine d'états finie pour la planification de l'auto-guérison

Ainsi que nous l'avons expliqué précédemment, la machine d'état finie pour la planification de l'auto-guérison sert à anticiper la défaillance d'un sous ensemble de caractéristiques afin d'y remédier efficacement tout en maintenant un niveau de service proche de l'optimal. Par conséquence, cette machine d'états finie doit être périodiquement maintenue à jour en cours d'exécution afin de tenir compte de l'évolution constante du système lui-même ainsi que de son environnement. Par conséquence, l'algorithme de génération de la machine d'états finie doit être exécuté de manière régulière ou a chaque modification significative de l'environnement d'exécution.

Il convient a ce propos de noter la fréquence d'exécution de l'algorithme de construction de la machine d'états est un paramètre important pour l'efficacité de l'approche. En effet, si cette fréquence est trop faible, la machine d'états finie risque de produire des configurations qui bien que valides, ne sont pas proche du comportement optimal attendu par ce que leurs fitness a été calculée a partir de paramètres désuets. Au contraire, si la fréquence de construction de cette machine est trop élevée, cela risque de générer une surcharge calculatoire trop importante et nuire ainsi a l'efficacité globale de l'approche.

L'algorithme 5.4 représente le code de génération de la machine d'états finie d'un système logiciel modélisée par un modèle de caractéristiques $FM =< G, r, E_{MAND}, F_{XOR}, F_{OR}, Impl, Excl >$, et représente par l'ensemble des dépendances transitives dans $FMEncoder$. $prevoir$ désigne l'ensemble des caractéristiques dont l'algorithme prévoit la défaillance, $current$ est une configuration de caractéristiques qui represente l'etat actuel du système et $defaillantes$ est l'ensemble des caractéristiques dont l'infrastructure de surveillance a releve le dysfonctionement.

Algorithme 6.4. L'algorithme de génération de la machine d'états finie pour la planification de l'auto guérison

Paramètres d'entré : $FMEncoder, defaillantes, prevoir, current$

Initialement

$racine \leftarrow current$

$S \leftarrow \{racine\}$

$T \leftarrow \emptyset$

$debut$

$Pour\ chaque\ element\ f\ \in prevoir\ faire$

$si\ f \in state\ alors$

$defaillantes' \leftarrow defaillantes \cup \{f\}$

$state' \leftarrow AlgorithmeGenetique(FMEncoder, defaillantes')$

$si\ state \notin S\ alors$

$S \leftarrow S \cup \{state'\}$

$finsi$

$T \leftarrow T \cup (state, f, state')$

$recommencer\ l'algorithme\ avec\ les\ parametres$

$: FMEncoder, defaillantes', prevoir - \{f\}, state'$

$finsi$

$finPour$

$retourner\ MEF_{FM,predire} =< S, T, racine >$

fin

L'algorithme 6.4 commence par initialiser la racine de la machine à l'état actuel du système, l'ensemble des états à l'état initial et l'ensemble des transitions à l'ensemble vide. Ensuite l'algorithme calcul récursivement les états et les transitions de la machine en parcourant l'ensemble des caractéristiques dont il est demander de prévoir la défaillance. Cela se fait par la simulation de l'échec de chacune de ces caractéristiques en les ajoutant à l'ensemble $defaillantes$ lors de l'exécution de l'algorithme génétique pour la prise de décision dans le cadre de l'auto-guérison (algorithme 6.1). L'orque cet algorithme retourne une configuration proche de l'optimale qui exclue la caractéristique potentiellement défaillante, cette configuration est ajouter à l'ensemble des états de la machine et une transition entre $current$ et ce nouvel état est ajoutée à l'ensemble des transitions.

Une piste intéressante pour améliorer l'efficacité de l'algorithme à trouver des alternatives optimales qui remédient au dysfonctionnement des composants serait d'utiliser la technique bien connue de *seeding*. Cette technique consiste à démarrer l'algorithme génétique non pas avec une

population aléatoire mais avec une population de solutions existantes. Dans le cas de l'algorithme 6.4, cette population peut consister simplement en des variantes aléatoire (une sorte de mutants) de la solution courante (dans la mesure où on considère que l'état courant est un état proche de l'optimal). Une autre alternative peut consister à faire le seed avec la population des solutions de l'algorithme génétique pour l'adaptabilité et l'auto-guérison (l'algorithme 6.1) qui s'exécute parallèlement à l'algorithme de génération de la machine d'états finie. La figure 6.4 donne un aperçu de cette approche. Toutefois, sa mise en œuvre n'est pas encore effective mais constitue une piste privilégiée pour les travaux futurs.

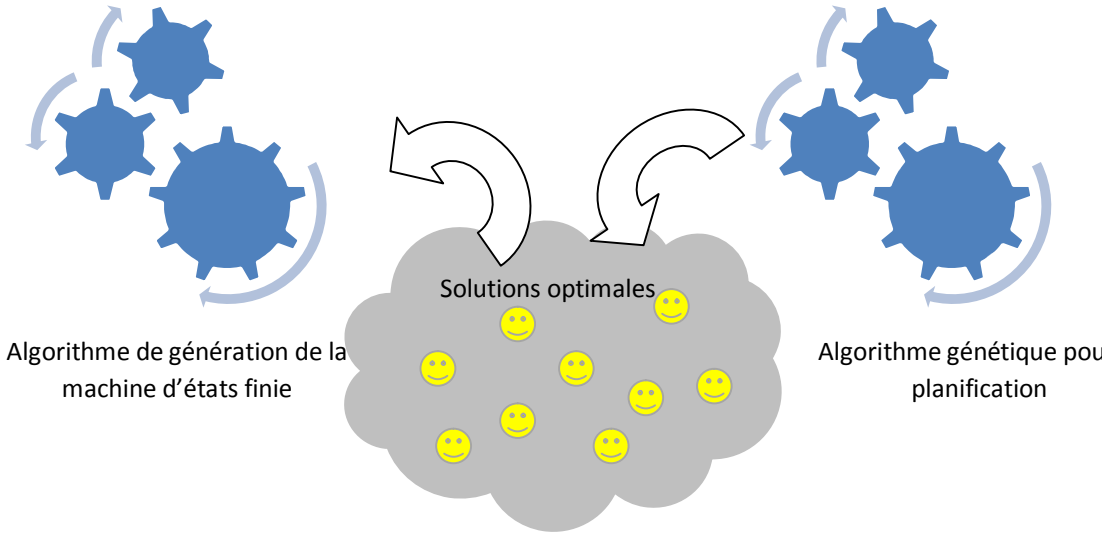


Figure 6.4 Aperçu de la technique de *seeding* entre l'algorithme génétique pour la planification et l'algorithme de génération de la machine d'états finie

6.3.3 L'algorithme de planification de l'auto-guérison

L'algorithme 6.5 illustre le code de l'algorithme de planification du recouvrement de la défaillance d'un composant f. cet algorithme découle naturellement de la structure et de la sémantique de la machine d'états finie décrite dans les sections précédentes.

Algorithme 6.5 Algorithme pour la planification du recouvrement de défaillance

Paramètres d'entrée : $MEF < racine, S, T >$,$defaillantes$, f

$debut$

Si $\exists state \in S\ tel\ que\ (racine, f, state) \in T$ alors

"$franchir\ la\ transition\ (racine, f, state)$"

$defaillante \leftarrow defaillantes \cap state$

$Si\ défaillantes\ = \emptyset\ alors$

$Retourner\ state$

$Sinon$

$f \leftarrow un\ element\ de\ defaillantes$

$Executer\ l'algorithme\ avec\ les\ parametre\ MEF, défaillantes - \{f\}, f$

$sinon$

$Executer\ l'algorithme\ genetique\ pour\ la\ planification$

$finsi$

fin

L'algorithme de 6.5 prend en paramètre d'entrée la machine d'états finie MEF, l'ensemble des caractéristiques défaillantes $défaillantes$, et la caractéristique f qui est celle qui a declencher le processus d'auto-guerison (c'est-à-dire une caractéristique active dans la configuration actuelle du système et dont l'infrastrucutre de surveillance vient de detecter l'échec).

L'algorithme commence par vérifier si la défaillance de f a été anticipée (ce qui revient a cherché une transition a partir de racine étiquetée par f), si ce n'est pas le cas, alors il est nécessaire de déclencher l'algorithme 6.1 pour la planification de l'adaptation et l'auto-guérison ce qui risque d'entrainer un certain délai de réponse. Si au contraire, la défaillance de f a été anticipée, alors il convient de considérer l'état cible de la transition correspondante. Si celui-là ne contient que des caractéristiques non défaillantes, alors il s'agit de l'état cible recherchée et l'algorithme se termine en le retournant. Sinon il est nécessaire de continuer le parcoure de la machine d'état avec l'une des caractéristiques défaillantes dans l'état cible de la transition étiquetée par f.

Prenons a titre d'exemple, le traitement de la défaillance des caractéristiques $serveur1$ et $smart_{TV}$ dans l'exemple illustrée par la figure 6.3. A partir de $racine$, l'algorithme est invoqué avec les paramètres $défaillantes = \{serveur1, smart\_tv\}$ et $f = serveur1$. Dans la mesure ou la transition $(racine, serveur1, cible)$ existe (ou $cible = \{Serveur2, smart_{TV}, internet\}$), et que $cible \cap defaillantes = \{smart_{tv}\} \neq \emptyset$ alors l'algorithme est à nouveau exécuté avec les paramètres $défaillantes = \{smart_{tv}\}$ et $f = smart\_tv$.

Cette fois, la transition $(1, smart_{tv}, cible)$ est trouvée (où $cible = \{Serveur2, smart_{phone}, internet\}$). Puisque $cible \cap defaillantes = \emptyset$, alors la configuration $cible = \{Serveur2, smart - phone, internet\}$ est retournée est elle constitue l'état cible recherché qui recouvre la perte des caractéristiques $serveur1$ et $smart_{TV}$.

Au contraire, si la défaillance de la caractéristique $internet$ est détectée, l'algorithme n'est pas capable de trouver une transition étiquetée par $internet$ a partir de $racine$ (en effet, la défaillance

de $internet$ n'as pas été anticipée par l'algorithme de génération de la machine d'états finie). Dans ce cas il est nécessaire de calculer un état cible en invoquant l'algorithme 6.1 pour la planification de l'adaptabilité et l'auto-guérison.

Il convient de noter enfin, que l'algorithme 6.5 n'est invoqué que dans le cas de la défaillance d'une caractéristique active. Ainsi, si $serveur2$ dysfonctionne, cela n'entraine pas une nouvelle planification. Par contre, sa défaillance est prise en compte par le processus de planification une fois celui-ci déclenchée de la manière que nous avons expliqué plus haut.

6.4 Résumé du chapitre

Dans ce chapitre, nous avons proposé une approche évolutionnaire au problème de la planification de l'auto-guérison. A cette fin, nous avons étendu l'algorithme génétique pour la planification de l'adaptation afin de prendre en considération la perte ou le dysfonctionnement de composants. Plus encore, nous avons exploité une machine d'états finie dans le but de d'anticiper la défaillance de composants clefs afin de réduire le temps nécessaire au recouvrement de leurs pertes. L'algorithme de construction de la machine d'états finie a été proposé et illustré sur l'exemple du système de Sante à Domicile.

Nous avons par ailleurs relevé deux paramètres décisifs de l'approche qui sont la fréquence de mise à jour de la machine d'états finie et le choix des composants dont il convient de prévoir la défaillance. Une étude approfondie de ces deux paramètres nous semble pertinente et fera l'objet de travaux futurs.

Enfin, Nous avons présenté l'algorithme de recouvrement de pannes à la base de la machine d'états finie et de l'algorithme générique pour la planification. Cet algorithme offre l'avantage de produire des plans d'auto-guérison qui maintiennent des niveaux de service proche de l'optimal même dans le cas de la défaillance de plusieurs composants ou en présence d'incertitude vis-à-vis des données de surveillance de l'environnement. Idéalement, ces plans sont génères dans des laps de temps de temps très courts ce qui permet a notre approche d'être mise en œuvre dans des systèmes temps réels particulièrement sensible a la qualité de service.

Il convient toutefois de signaler que l'approche proposée dans ce chapitre fait l'hypothèse de la continuée des fonctions objectifs du système. Plus précisément, si nous supposons que la défaillance d'un composant n'opère pas de rupture dans les fonctions objectifs, alors les plans de recouvrement aurons la même valeur de fitness (ou en tout cas une valeur proche) avant et après la survenue de l'erreur. Si au contraire, les valeurs de fitness des plans après la survenue de l'erreur sont catégoriquement différentes de celles calculées avant sa survenue, alors les plans prévue par la

machine d'états finie ne sont plus optimaux, néanmoins ils restent valides et ne risquent pas d'entrainer le dysfonctionnement ou la défaillance du système entier.

Chapi VII

Conclusion et perspectives

7.1 Contributions

Les travaux présentés dans cette thèse s'articulent autour de la prise de décision dans les systèmes logiciels embarques adaptables. Ils visent à fournir un ensemble cohérent de techniques et d'algorithmes pour la prise de décision basés sur l'approche des lignes de produits dynamiques adaptés aux contraints imposées par les systèmes logiciels embarqués modernes.

Tout d'abord, nous avons introduit les relations de dépendances transitives entre les caractéristiques. L'utilisation des relations de dépendances transitives permet une implémentation simple mais efficace des algorithmes de raisonnent sur les modèles de caractéristiques. Par ailleurs, la représentation binaire de ces relations permet d'améliorer significativement les performances de ces algorithmes. Cela offre la possibilité de modéliser et de raisonner sur de très grands systèmes à un coût relativement acceptable. Nous avons ensuite présenté les deux operateurs de sélection transitive qui exploitent ces relations pour réduire considérablement la complexité algorithmique des algorithmes d'analyse. L'algorithme de calcul des relations de dépendance transitives ainsi que quelques algorithmes de raisonnement sur le modèle de caractéristiques ont également été introduits. Nous avons par ailleurs abordé l'encodage de ces relations sous forme ensembliste binaire afin d'accélérer les différents calculs.

Notre approche évolutionnaire constitue un support efficace à la prise de décision dans les systèmes adaptables. Nous avons notamment introduit notre algorithme génétique qui exploite les operateurs de sélection transitive. Les différents operateurs génétiques ont été revus avec plus ou moins de détails. Nous avons par ailleurs rapporté les résultats de la comparaison numérique que nous avons effectuée avec un algorithme de la littérature et qui montrent la supériorité de notre algorithme.

Enfin, nous avons présenté une adaptation de notre approche évolutionnaire au problème de l'auto-guérison. Cette adaptation comporte notamment l'utilisation d'une machine d'états finie qui est mise à jour périodiquement par un algorithme génétique modifié afin de fournir un plan proche de l'optimum dans un délai de temps acceptable.

Le concept de caractéristique (feature) pouvant s'appliquer dans le contexte des lignes de produits dynamiques à n'importe quelle unité de conception logicielle (modèle, composant ou bout

de code), notre approche peut facilement être intégrée à des plateformes ou des middlewares LDPL existants.

7.2 Perspectives

Les perspectives de ce travail de recherche sont multiples. En effet, l'introduction de la notion de relation de dépendance transitive ouvre des voix intéressantes aussi bien dans la discipline des lignes de produits classiques que dans celle des lignes de produits dynamiques. A court terme, nous envisageons de terminer l'implémentation de la plateforme de raisonnement sur les modèles de caractéristiques de grande et très grande taille en tirant profil des performances améliorées des algorithmes d'analyse basés sur les relations de dépendances transitives. Dans ce sens, une étude comparative poussée avec d'autres techniques de raisonnent sur les modèles de caractéristiques, notamment celles basées sur les solveurs SAT, CSP et BDD, est envisagée. Dans le même ordre d'idée, nous projetons d'intégrer la notion de dépendance transitive aux outils d'analyses basés sur les solveurs afin d'en améliorer les performances en réduisant sensiblement l'espace des solutions valides que le solveur doit explorer.

Par ailleurs, la généralisation aux modèles de caractéristiques étendue, des relations de dépendances transitives présentées ici, permettrait de traiter une famille de systèmes que les modèles de caractéristiques classiques ne permettent pas (ou pas assez) de modéliser. Notamment, les modèles de caractéristiques classiques ne permettent pas d'exprimer des relations de dépendances évoluées (ils utilisent exclusivement les relations d'implications et d'exclusion binaires) alors que les modèles étendus permettent d'exprimer des relations plus complexes sous la forme de formules logiques impliquant un nombre quelconque de caractéristiques (Benavides et al. 2005). Ainsi, la prise en compte de ce type de contraintes peut se faire par l'extension de la définition de la relation de dépendance non déterministe : la (de)sélection d'une caractéristique détermine la sélection d'un sous groupe de caractéristiques dans un groupe (c'est par exemple le cas de la relation de la caractéristique parent avec ses sous-caractéristiques).

D'un autre cote, les performances accrues dues aussi bien aux relations de dépendances transitives qu'a l'algorithme génétiques nous encouragent à explorer des directions de recherches tout à fait nouvelles. Ainsi, l'application des techniques de lignes de produits dynamiques à cette nouvelle tendance du génie logiciel que sont les systèmes cyber-physiques peut enfin être envisagée. En effet, face à la taille particulièrement grande des systèmes cyber-physiques un algorithme de prise de décision qui permet une très bonne mise à l'échelle est tout a fait impératif. Nous pensons qu'une implémentation appropriée des algorithmes présentés dans cette thèse

(notamment, une représentation efficace des $Bitset$ sous une forme binaire, au lieu d'un tableau de bits) permettrait de fournir des solutions optimales dans des délais de l'ordre des microsecondes même en la présence de plusieurs milliers de caractéristiques.

Une amélioration possible de l'algorithme génétique serait une meilleure prise en charge des attributs associés aux caractéristiques. En effet, l'algorithme d'encodage des solutions proposée ici préconise de traiter les attributs comme un groupe exclusif de sous caractéristiques. Cette solution déjà testée dans le cadre d'approches similaires à la notre (Guo et al, 2010) (Pascual et al, 2015) présente deux principaux inconvénients : i) elle suppose que les attributs ont (ou peuvent avoir) des valeurs discrètes en nombre fini, ii) elle peut amener à une explosion de l'espace de recherche des configurations. Afin d'y remédier, nous envisageons l'implémentation d'un algorithme génétique qui exploite un double chromosome pour la représentation des solutions. Cette représentation divise l'espace de recherche en deux espaces distincts (l'espace des combinaisons de caractéristiques et celui des combinaisons d'attributs).

Enfin, l'algorithme de planification de l'auto-guérison ouvre également des pistes intéressantes de recherche. Ainsi, la mise en œuvre de l'approche sur un système réel ne permettra pas seulement de valider l'idée sous-jacente mais sera aussi l'occasion d'étudier les paramètres déterminants de son efficacité à savoir la fréquence de mise à jour de la machine d'états finie et le choix de l'ensemble des composants dont il convient de prévoir la défaillance. Par ailleurs, l'utilisation d'un *seed* commun entre l'algorithme génétique pour la planification et l'algorithme de génération de la machine d'états finie permettrait un gain significatif et par conséquence un impact important sur l'efficacité générale de l'approche proposée dans cette thèse.

Références Bibliographiques

Acher, M., Collet, P., Lahire, P., and France, R. (2011). Decomposing Feature Models: Language, Environment, and Applications. In Automated Software Engineering (ASE'11), short paper: demonstration track, IEEE/ACM (Chapter 8)

M. Acher, Managing Multiple feature models, Foundations language and applications PhD Thesis, University of Nice Antipolis. 2011.

Ailisto, H., Alahuhta, P., Haataja, V., Kylloenen, V. and Lindholm, M. (2002) 'Structuring context aware applications: Five-layer model and example case', Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing, Goteborg, Sweden, available online: http:// www.comp.lancs.ac.uk/computing/users/dixa/conf/ubicomp2 002-models/pdf/Ailisto-Ubicomp%20Workshop8.pdf.

Ajay Gupta. Management of conflicting obligations in self-protecting policy- based systems. In ICAC '05: Proceedings of the Second International Confer- ence on Automatic Computing, pages 274–285, Washington, DC, USA, 2005. IEEE Computer Society.

S. Agarwala, Yuan Chen, D. Milojicic, and K. Schwan. Qmon: Qos- and utility-aware monitoring in enterprise systems. In ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing, pages 124–133, Washington, DC, USA, 2006. IEEE Computer Society.

D. Agrawal, S. Calo, J. Giles, Kang-Won Lee, and D. Verma. Policy man- agement for networked systems and applications. In Integrated Network Man- agement, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on, pages 455–468, May 2005.

J. Aldrich, C. Chambers, and D. Notkin. ArchJava: connecting software architecture to implementation. In Proc. of ICSE 2002, Orlando, USA, pages 187–197. ACM, 2002.

Almeida, A., Dantas, F., Cavalcante, E., Batista, T. (2014) A Branch-and-Bound Algorithm for Autonomic Adaptation of Multi-Cloud Applications. IEEE. 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014). Chicago, United States. pp.315-323,15

Alves, V., Schneider, D., Becker, M., Bencomo, N. and Grace P. (2009) Comparative Study of Variability Management in Software Product Lines and Runtime Adaptable Systems. In Proc. Workshop on Variability Modeling of Software-intensive Systems (VaMoS), pages 9–17. University of Duisburg-Essen.

Anton A. Bougaev. Pattern recognition based tools enabling autonomic com- puting. pages 313–314, 2005. □

Apel, S., Janda, F., Trujillo, S., and Kästner, C. (2009). Model superimposition in software product lines. In Paige, R. F., editor, ICMT, volume 5563 of Lecture Notes in Computer Science, pages 4–19. Springer. 19

Apel, S. and Kästner, C. (2009). An overview of feature-oriented software development.Journal of Object Technology (JOT), 8(5):49–84. 13, 29

Asikainen, T., Männistö, T., and Soininen, T. (2004). Using a configurator for modelling and configuring software product lines based on feature models. In Proceedings of the Workshop on Software Variability Management for Product Derivation, Software Product Line Conference (LPLC3). 18

Asikainen, T., Mannisto, T., and Soininen, T. (2006). A unified conceptual foundation for feature modelling. In Proc. of SPLC'2006, pages 31–40. IEEE. 18

Asikainen, T., Männistö, T., and Soininen, T. (2007). Kumbang: A domain ontology for modelling variability in software product families. Advanced Engineering Informatics, 21(1):23–40. 18

Austin, T., Blaauw, D., Mahlke, S., Mudge, T., Chakrabarti, C., Wolf, W.: Mobile supercomputers. Computer 37(5), 81–83 (2004). doi:http://dx.doi.org/10.1109/MC.2004.1297253

Bachmann, F. and Bass, L. (2001). Managing variability in software architectures. SIGSOFTSoftw. Eng. Notes, 26:126–132. 17, 29

Bass, L., Clements, P., , and Kazman, R. (1998). Software Architecture in Practice. Addison-Wesley. 9, 10, 11

V. Batra, J. Bhattacharya, H. Chauhan, A. Gupta, M. Mohania, and U. Sharma. Policy driven data administration. Policies for Distributed Sys- tems and Networks, IEEE International Workshop on, 0:0220, 2002.

Batory, D., Sarvela, J. N., and Rauschmayer, A. (2004). Scaling step-wise refinement. IEEE Trans. Softw. Eng., 30(6):355–371. 19, 32

Batory, D. S. (2005). Feature models, grammars, and propositional formulas. In 9th International Software Product Line Conference (SPLC'05), volume 3714 of LNCS, pages 7–20. 25, 27, 28, 29, 32, 66, 128

Baldauf, M., Dustdar S. and Rosenberg F. (2007) A survey on context-aware systems, in Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, pp.263–277.

D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mo- hindra, D. G. Shea, and M. Vanover. Autonomic personal computing. IBM Syst. J., 42(1):165–176, 2003.

Benavides, D., Segura, S., Ruiz-Cortes A. (2010) Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (6) 615–636.

Benavides, D., Trinidad, P., Ruiz-Cortes A. (2005) Automated reasoning on feature models, in: Advanced Information Systems Engineering, Springer, pp. 381–390.

Bencomo, N., Blair, G,. Flores, C,. and Sawyer, P. (2008a) Reflective component-based technologies to support dynamic variability. In 2nd International Workshop on Variability Modeling on Software-intensive Systems (VaMoS'08), Essen, Germany.

D. Benavides, S. Segura, A. Ruiz-Cortes, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (6) (2010) 615–636.

Bencomo, N., Grace, P., Flores, C., Hughes, D., and Blair G. (2008) Genie: Supporting the model driven development of reflective, component-based adaptive systems. In ICSE 2008 - Formal Research Demonstrations Track.

Bencomo, N., Lee, J., and Hallsteinsen S. (2010) How dynamic is your Dynamic Software Product Line? In: Proceedings of the 14th International Software Product Line Conference. Lancaster University, Lancaster, pp. 61-68. ISBN 9781862202470

Bencomo, N., Sawyer, P., Blair, G. S. and Grace P. (2008). Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In Proc. Int'l. Software Product Line Conf. (SPLC), pages 23–32. IEEE CS.

Beuche, D., Papajewski, H., and Schrader-Preikschat, W. (2004). Variability management with feature models. Science of Computer Programming, 53(3):333–352. 11, 18

Beuche, D. (2008). Modeling and building software product lines with Pure::Variants. In 2008 12th International Software Product Line Conference, pages 358–358, Limerick, Ireland

BigLever – Gears (2006). http://www.biglever.com/solution/product.html. 32

Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., III, W. N. M., and Diao, Y. Able: A toolkit for building multiagent autonomic systems. IBM Systems Journal, 41:250–371, 2002. □

Bjerregaard, T., Mahadevan, S.: A survey of research and practices of network-on-chip. ACM Comput. Surv. 38(1)

(2006). doi:http://doi.acm.org/10.1145/1132952.1132953.

Borkar, S., Chien, A.A.: The future of microprocessors. Commun. ACM 54(5), 67–77 (2011). doi:10.1145/1941487.1941507. http://doi.acm.org/10.1145/1941487.1941507

Bradley Schmerl and David Garlan. Exploiting architectural design knowledge to support self-repairing systems. In SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering, pages 241–248, New York, NY, USA, 2002. ACM.

Brooks, Jr., F. P. (1987). No silver bullet essence and accidents of software engineering.Computer, 20:10–19. 14

Brown, P.J. (1996) 'The stick-e document: a framework for creating context-aware applications', Proceedings of the Electronic Publishing, Palo Alto, pp.259–272.

Brataas, G., Hallsteinsen, S.O., Rouvoy, R., Eliassen F. (2007) Scalability of decision models for dynamic product lines in: SPLC, (2), pp. 23–32.

Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H. M., Kienle, H. M., Litoiu, M., Müller, H. A., Pezze M. and Shaw M. (2009) Engineering Self-Adaptive Systems through Feedback Loops.vol. 5525, pp. 48–70. Springer, Heidelberg

Burger, D., Goodman, J.R.: Billion-transistor architectures: there and back again. Computer 37(3), 22–28 (2004). doi:http://dx.doi.org/10.1109/MC.2004.1273999

Burns, J., Gaudiot, J.L.: Smt layout overhead and scalability. IEEE Trans. Parallel Distrib. Syst. 13(2), 142–155 (2002). doi:http://dx.doi.org/10.1109/71.983942

Cetina, C., Fons, J., Pelechano V. (2008) Applying software product lines to build autonomic pervasive systems, in: Software Product Line Conference. SPLC'08. 12th International, IEEE, 2008, pp. 117–126.

C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes", IEEE Computer, 2009,

Chen, H., Finin, T. and Joshi, A. (2003) 'An ontology for context-aware pervasive computing environments', The Knowledge Engineering Review, Cambridge University Press, Vol. 18, pp.197–207.

Chen, H. (2004) An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD Thesis, University of Maryland, Baltimore County.

Chen, H., Finin, T. and Joshi, A. (2004) 'An ontology for context-aware pervasive computing environments', Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, Vol. 18, No. 3, pp.197–207.

Chen, L., Babar, M. A., and Ali, N. (2009). Variability management in software product lines: a systematic review. In Muthig, D. and McGregor, J. D., editors, SPLC, volume 446 of ACM International Conference Proceeding Series, pages 81–90. ACM. 18

Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., and Raskin, J.-F. (2010b). Model checking lots of systems: efficient verification of temporal properties in software product lines. In Kramer, J., Bishop, J., Devanbu, P. T., and Uchitel, S., editors, ICSE (1), pages 335–344. ACM. 19, 204

Classen, A., Heymans, P., Schobbens, P.-Y., and Legay, A. (2011). Symbolic model checking of software product lines. In Taylor, R. N., Gall, H., and Medvidovic, N., editors, ICSE'11,pages 321–330. ACM. 19, 130, 204

Clements, P. C. and Krueger, C. (2002). Point – counterpoint: Being proactive pays off – eliminating the adoption. IEEE Software, 19(4):28–31. 11

Clements, P. and Northrop, L. M. (2001). Software Product Lines : Practices and Patterns.Addison-Wesley Professional. 9, 10, 12

Conte, G., Tommesani, S., Zanichelli, F.: The long and winding road to high-performance image processing with

mmx/sse. In: CAMP '00: Proceedings of the Fifth IEEE International Workshop on Computer Architectures for Machine Perception (CAMP'00), p. 302. IEEE Computer Society, Washington, DC (2000)

Czarnecki, K., Bednasch, T., Unger, P., and Eisenecker, U. (2002). Generative Programming for Embedded Software: An Industrial Experience Report, pages 156–172. LNCS. 28, 201

Czarnecki, K., Helsen, S., and Eisenecker, U. (2004). Staged configuration using feature models. In Software Product Lines, volume 3154/2004 of Lecture Notes in Computer Science, pages 266–283. Springer Berlin / Heidelberg. 32

Czarnecki, K., Helsen, S., and Eisenecker, U. (2005b). Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practice, 10(2):143–169. 18, 28, 31, 43, 44, 155, 200, 201

Czarnecki, K., Kim, C. H. P., and Kalleberg, K. T. (2006). Feature models are views on ontologies. In SPLC '06: Proceedings of the 10th International on Software Product Line Conference, pages 41–51,Washington, DC, USA. IEEE Computer Society. 23, 28, 201, 203

Czarnecki, K. and Antkiewicz, M. (2005). Mapping features to models: A template approach based on superimposed variants. In GPCE'05, volume 3676 of LNCS, pages 422– 437. 18, 19, 30, 128, 203

Czarnecki, K. and Eisenecker, U. (2000). Generative Programming: Methods, Tools, and Applications. Addison-Wesley. 12, 14, 16, 21, 29, 32

Czarnecki, K. and Pietroszek, K. (2006). Verifying feature-based model templates against well-formedness ocl constraints. In GPCE'06, pages 211–220. ACM. 19, 204

Czarnecki, K. and Wasowski, A. (2007). Feature diagrams and logics: There and back again. In SPLC'07, pages 23–34. 25, 26, 28, 66, 67, 93, 128, 130, 133, 136

De Jong K. (1975) An analysis of the behavior of a class of genetic adaptive systems", Doctoral Dissertation. Ann Arbor: The University of Michigan.

Deelstra, S., Sinnema, M., and Bosch, J. (2004). Experiences in Software Product Families:Problems and Issues During Product Derivation, pages 165–182. LNCS. xii, 12, 13

Deelstra, S., Sinnema, M., and Bosch, J. (2005). Product derivation in software product families: a case study. Journal of Systems and Software, 74(2):173–194. 12, 84

Yixin Diao, Joseph L. Hellerstein, Sujay Parekh, Rean Griffith, Gail Kaiser, and Dan Phung. Self-managing systems: A control theory foundation. In ECBS '05: Proceedings of the 12th IEEE International Conference and Work- shops on Engineering of Computer-Based Systems, pages 441–448, Washing- ton, DC, USA, 2005. IEEE Computer Society.

Dinkelaker, T., Mitschke, R., Fetzer, K., Mezini M. (2010) A dynamic software product line approach using aspect models at runtime, in: 5th Domain-Specific Aspect Languages Workshop.

Divo C.E.A. (2011) Automated Reasoning on Feature Models via Constraint Programming, master thesis.

Djebbi, O., Salinesi, C., Diaz D. (2007) Deriving product line requirements: the RED-PL guidance approach, in: Software Engineering Conference,. APSEC 2007. 14th Asia-Pacific, pp. 494–501.

Dowling. J, Cunningham, R. Curran, E and Cahill. V. Build- ing autonomic systems using collaborative reinforcement learning. volume 21, pages 231–238, New York, NY, USA, 2006. Cambridge University Press.

El-Mihoub, T., Hopgood, A., Nolle, L., Battersby, A. Hybrid Genetic Algorithms: A Review in Engineering Letters, 13:2, EL_13_2_11. August 2006

Emil C. Lupu and Morris Sloman. Conflicts in policy-based distributed sys- tems management. IEEE Trans. Softw. Eng., 25(6):852–869, 1999.

Ensan, F., Bagheri, E., Gasevic D. (2012) Evolutionary Search-Based Test Generation for Software Product Line Feature Models. CAiSE 2012: 613-628

Erwig, M. (2010). A language for software variation research. In Proceedings of the ninth international conference on Generative programming and component engineering, GPCE '10, pages 3–12, New York, NY, USA. ACM. 17

Erwig, M. and Walkingshaw, E. (2011). The choice calculus: A representation for software variation. ACM Transactions on Software Engineering and Methodology (TOSEM). to appear. 17

C. Escoffier and R. S. Hall. Dynamically adaptable applications with iPOJO service components. In Proc. of SC 2007, Braga, Portugal, volume 4829 of LNCS, pages 113–128. Springer, 2007.

Espinoza, F., Persson, P., Sandin, A., Nystrom, H., Cac-ciatore, E. and Bylund, M. (2001) 'GeoNotes: social and navigational aspects of location-based information systems', Proceedings of the 3rd International Conference on Ubiquitous Computing, Atlanta, Georgia, USA, pp.2–17.

FaMa (2008). http://www.isa.us.es/fama/. 32

Fenson. E and Howard.R. Reinforcement learning for autonomic network repair. In ICAC '04: Proceedings of the First International Conference on Autonomic Computing, pages 284–285, Washington, DC, USA, 2004. IEEE Computer Society. □

Finkelstein, L., Gabriolovic, E., Matias, Y., Rivilin, E., Solan, Z., Wolfman, G. and Ruppin, E. (2001) 'Placing search in context: the concept revisited', Proceedings of the 10th International World Wide Web Conference (WWW 10), Hong Kong.

Flynn, M.J., Hung, P.:Microprocessor design issues: Thoughts on the road ahead. IEEE Micro. 25(3), 16–31 (2005). doi:http://dx.doi.org/10.1109/MM.2005.56

Fujimura, A.: All lithography roads ahead lead to more e-beam innovation. In: Future Fab. Int. (37), http://www.future-fab.com (2011)

Gaia Project (2005) http://gaia.cs.uiuc.edu/.

A. Ganek and R. J. Friedrich. The road ahead achieving wide-scale deployment of autonomic technologies. In Chairing the Town hall meeting at the 3rd IEEE International Conference on Autonomic Computing, 2006.

D. Garlan, B. Schmerl, and J Chang. Using gauges for architecture-based mon- itoring and adaptation. In In Working Conference on Complex and Dynamic Systems Architecture , Brisbane, Australia., 2001.

Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjorven, E., Hallsteinsen, S., Horn, G., Khan, M. U., Mamelli, A., Papadopoulos, G. A., Paspallis, N., Reichle, R. and E. Stav. (2009) A comprehensive solution for application-level adaptation. Softw. Pract. Exper., 39(4):385–422,.

Génova, G., Valiente, M. C., and Marrero, M. (2009). On the difference between analysis and design, and why it is relevant for the interpretation of models in model driven engineering. Journal of Object Technology, 8(1):107–127. 14

Gerald Tesauro and Jeffrey O. Kephart. Utility functions in autonomic sys- tems. In ICAC '04: Proceedings of the First International Conference on Auto- nomic Computing, pages 70-77, Washington, DC, USA, 2004. IEEE Computer Society.

Glass, R. L. (2001). Frequently forgotten fundamental facts about software engineering. IEEE Software, 18:112,110–111. 10

Gomaa, H. and Hussein, M. (2003) Dynamic software reconfiguration in software product families. In PFE, pages 435–444.

Griss, M. L., Favaro, J., and d' Alessandro, M. (1998). Integrating feature modeling with the rseb. In ICSR '98: Proceedings of the 5th International Conference on Software Reuse, page 76, Washington, DC, USA. IEEE. 18, 32

H. Guo. A bayesian approach for autonomic algorithm selection. In Proceedings of the IJCAI workshop on AI and autonomic computing: developing a research agenda for selfmanaging computer systems, 2003.

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.

Halmans, G. and Pohl, K. (2003). Communicating the variability of a software-product family to customers. Software and System Modeling, 2(1):15–36. 17, 30

S. Hallsteinsen, M. Hinchey, Sooyong Park, and K. Schmid. Dynamic software product lines. Computer, 41(4):93–95, April 2008.

Harman, M., Jia, Y., Krinke, J., Langdon, B., Petke, J., Zhang, Y. (2014) Search based software engineering for software product line engineering: a survey and directions for future work (keynote pa- per), in: 18th International Software Product Line Conference (SPLC 14), Florence, Italy, pp. 5–18.

Harter, A., Hopper, A., Steggles, P., Ward, A. and Webster, P. (2002) 'The anatomy of a context-aware application', Wirless Networks, Vol. 8, Nos. 2–3, pp.187–197.

Heidenreich, F., Kopcsek, J., and Wende, C. (2008). FeatureMapper: Mapping Features to Models. In Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08), pages 943–944, New York, NY, USA. ACM. 18, 19, 204

Heidenreich, F., Sanchez, P., Santos, J., Zschaler, S., Alferez, M., Araujo, J., Fuentes, L., amd Ana Moreira, U. K., and Rashid, A. (2010). Relating feature models to other models of a software product line: A comparative study of featuremapper and vml*. Transactions on Aspect-Oriented Software Development VII, Special Issue on A Common Case Study for Aspect-Oriented Modeling, 6210:69–114. 18, 19, 30, 128

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G. and Altmann, J. (2002) 'Context-awareness on mobile devices – the hydrogen approach', Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pp.292–302.

Holland J. H. (1992) Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA.

Hong, J.I. and Landay, J.A. (2001) 'An infrastructure for context-aware computing', Human-Computer Interaction, Vol. 16, pp.287–303.

J. Hong, E. Suh, and S. Kim. Context-aware systems: A literature review and classification. Expert Syst. Appl., 36(4):8509–8522, 2009.

P. Horn. Autonomic computing: Ibm's perspective on the state of information technology, 2001

M. C. Huebscher and J. A. McCann. A survey of autonomic computing— degrees, models, and applications. ACM Computing Surveys, 40(3):1–28, Au- gust 2008.

Hull, R., Neaves, P. and Bedford-Roberts, J. (1997) 'Towards situated computing', Proceedings of the First International Symposium on Wearable Computers (ISWC '97), p.146.

IBM. An architectural blueprint for autonomic computing. Technical report, IBM., 2003.

Isci, C., Buyuktosunoglu, A., Cher, C., Bose, P., Martonosi, M.: An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. In: Proceedings of the 39th annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39, pp. 347–358. IEEE Computer Society, Washington, DC (2006). doi:10.1109/MICRO.2006.8

ITRS: ITRS 2011 Roadmap. Tech. rep., International Technology Roadmap for Semiconductors (2011)

Jacobson, I., Griss, M., and Jonsson, P. (1997). Software reuse: architecture, process and organization for business success. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. 10, 17

Janota, M. (2010). SAT Solving in Interactive Configuration. PhD thesis, Department of Computer Science at University College Dublin. 28, 136, 161, 162, 201

Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. Computer, 36(1):41–50, 2003.

Jiang, S., Svein O., Hallsteinsen, O., Barone, P., Mamelli, A., Mehlhase, S. and Scholz, U. (2010) Hosting and using services with qos guarantee in self-adaptive service systems. In DAIS, pages 15–28.

John, I. and Eisenbarth, M. (2009). A decade of scoping: a survey. In Proc. of SPLC'2009, volume 446 of ICPS, pages 31–40. ACM. 17

K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A feature–oriented reuse method with domain–specific reference architectures," Annals of Software Engineering, vol. 5, pp. 143–168, 1998.

Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). FORM: a feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, 5(1):143–168. 18, 29, 30, 32, 43, 44, 166, 200

Kästner , C. (2010). Virtual Separation of Concerns: Toward Preprocessors 2.0. PhD thesis, University of Magdeburg. Logos Verlag Berlin, isbn 978-3-8325-2527-9. 19, 42, 204

Kästner, C., Thüm, T., Saake, G., Feigenspan, J., Leich, T.,Wielgorz, F., and Apel, S. (2009a).Featureide: A tool framework for feature-oriented software development. In 31st International Conference on Software Engineering (ICSE'09), Tool demonstration, pages 611–614.19, 32, 108, 128

Kästner, C., Thüm, T., Saake, G., Feigenspan, J., Leich, T.,Wielgorz, F., and Apel, S. (2009b). Featureide: A tool framework for feature-oriented software development. In 31st International Conference on Software Engineering (ICSE'09), Tool demonstration, pages 611–614.

Karata¸ A. S., Oguztüzün, S, H. and Dogru. A. (2010) "Global Constraints on Feature Models". Proceedings of Principles and Practice of Constraint Programming - 16th International Conference (CP-2010), Scotland. Springer, vol. 6308, pp. 537-551. ISBN 9783642153952.

Karata¸ A. S., Oguztüzün, S, H. and Dogru. A. (2010) "Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains". Proceedings of Software Product Lines: Going Beyond - 14th International Conference, (SPLC-2010), South Korea. Springer, vol. 6287, pp. 286-299. ISBN 9783642155789.

J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. In POLICY '04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, page 3, Washington, DC, USA, 2004. IEEE Computer Society.

Kim, N.S., Austin, T., Blaauw, D., Mudge, T., Flautner, K., Hu, J.S., Irwin, M.J., Kandemir, M., Narayanan, V.: Leakage current: Moore's law meets static power. Computer 36(12), 68–75 (2003). doi:http://dx.doi.org/10.1109/MC.2003.1250885

J. E. Kim, O. Rogalla, S. Kramer, and A. Hamann. Extracting, specifying and predicting software system properties in component based real-time embedded software development. In Proc. of ICSE 2009, Vancouver, Canada, pages 28–38. IEEE, 2009.

Koza, J. R. (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). The MIT Press.

Koufaty, D., Marr, D.T.: Hyperthreading technology in the netburst microarchitecture. IEEE Micro. 23(2), 56–65 (2003)

Kramer, D. (2014) Unified GUI adaptation in Dynamic Software Product Lines. Doctoral thesis, University of West London.

Krueger, C. W. (1992). Software reuse. ACM Comput. Surv., 24:131–183. 10

Krueger, C. W. (2001). Easing the transition to software mass customization. In Software Product-Family Engineering,

4th International Workshop (PFE 2001), pages 282–293. 11

Krueger, C. W. (2006). New methods in software product line development. In 10th Int. Software Product Line Conf., pages 95–102, Los Alamitos, CA, USA. IEEE Computer Society.11

Lauenroth, K., Pohl, K., and Toehning, S. (2009). Model checking of domain artifacts in product line engineering. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, ASE '09, pages 269–280, Washington, DC, USA. IEEE Computer Society. 19

J. A. Mccann and J.S. Crane. Kendra: Internet distribution delivery system. In Society for Computer Simulation International, editor, In Proceedings of SCS Euromedia, pages 134–140. IEEE, 1998.

Li, J., Liu, X., Wang, Y., Guo J. (2012) Formalizing feature selection problem in software product lines using 0-1 programming, Practical Applications of Intelligent Systems 459–465.

D. Liu and H. Mei, "Mapping requirements to soft- ware architecture by feature-orientation," pp. 69–76, 2003.

Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description lan- guage. pages 291–298, 1999. □

Loukil, S., Kallel, S., Jmaiel M. (2014) Middleware for Dynamically Adapatative Systems, in the proceeding of ARCS 2014 conference. Springer. pages 72-84. Mannion, M. (2002). Using first-order logic for product line model validation. In SPLC 2: Proceedings of the Second International Conference on Software Product Lines, pages 176–187, London, UK. Springer-Verlag. 27

Hanan Lutfiyya, Gary Molenkamp, Michael Katchabaw, and Michael A. Bauer. Issues in managing soft qos requirements in distributed systems using a policy-based framework. In POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, pages 185–201, London, UK, 2001. Springer-Verlag.

Lymberopoulos. L, Lupu. E, and Sloman. M. An adaptive policy- based framework for network services management. J. Netw. Syst. Manage., 11(3):277–303, 2003. □

J. A. Mccann and J.S. Crane. Kendra: Internet distribution delivery system. In Society for Computer Simulation International, editor, In Proceedings of SCS Euromedia, pages 134–140. IEEE, 1998.

Maccari, A. and Heie, A. (2005). Managing infinite variability in mobile terminal software: Research articles. Softw. Pract. Exper., 35(6):513–537. 10

Marcilio Mendonca, AndrzejWa¸sowski, K. C. (2009). Sat-based analysis of feature models is easy. In SPLC'09, pages 231–241. IEEE. 28, 134, 135, 136

Mendonca, M. and Cowan, D. (2010). Decision-making coordination and efficient reasoning techniques for feature-based configuration. Science of Computer Programming, 75(5):311 – 332. Coordination Models, Languages and Applications (SAC'08). 28, 101, 161, 201

McIlroy, M. D. (1968). Mass-produced software components. Proc. NATO Conf. on Software Engineering, Garmisch, Germany. 1, 10

P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B.H.C. Cheng. Composing adaptive software. Computer, 37(7):56–64, July 2004.

Mendonca, M., Branco, M., and Cowan, D. (2009a). S.p.l.o.t.: software product lines online tools. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pages 761–762, New York, NY, USA. ACM. 32, 108, 128

Mendonca, M., Branco, M., and Cowan, D. (2009b). S.p.l.o.t.: software product lines online tools. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pages 761–762, New York, NY, USA. ACM. 32

Metzger, A., Pohl, K., Heymans, P., Schobbens, P.-Y., and Saval, G. (2007). Disambiguating the documentation of

variability in software product lines: A separation of concerns, formalization and automated analysis. In 15th IEEE International Conference on Requirements Engineering (RE '07), pages 243–253. xiv, 17, 18, 30, 44, 72, 96, 97, 99, 101, 166, 182,190, 200

Mezini, M. and Ostermann, K. (2004). Variability management with feature-oriented programming and aspects. SIGSOFT Softw. Eng. Notes, 29(6):127–136. 19

Morin, Brice and Barais, Olivier and Jézéquel, Jean-Marc and Ramos, Rodrigo (2007). Towards a generic aspect-oriented modeling framework. In Models and Aspects workshop, at ECOOP 2007. 19

Morin, B., Vanwormhoudt, G., Lahire, P., Gaignard, A., Barais, O., and Jézéquel, J.-M. (2008). Managing variability complexity in aspect-oriented modeling. Model Driven Engineering Languages and Systems, pages 797–812. 19

Morin, B., Fleurey, F., Bencomo, N., Jezequel, J.M., Solberg, A., Dehlen, V. and Blair G. (2008) An aspect-oriented and model-driven approach for managing dynamic variability. In MODELS'08 Conference, France.

Morin, B., Barais, O., Nain, G. and Jezequel J.M. (2009) Taming dynamically adaptive systems using models and aspects. In International COnference in Software Engineering (ICSE).

Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. Pon- der: A language for specifying security and management policies for dis- tributed systems. Technical report, 2000.

Nguyen T. Littman, M. and H. Hirsh. A model of cost-sensitive fault media- tion. In Proceedings of the IJCAI workshop on AI and autonomic computing: developing a research agenda for self-managing computer systems, 2003.

Nielsen M. J. Salahshour A. Manoel, E. and S. Sampath. Problem Determi- nation Using Self-Managing Autonomic Technology. IBM Redbooks, 2005.

Northrop, L. M. (2002). Sei's software product line tenets. IEEE Softw., 19:32–40. 11, 12

Parnas, D. L. (1976). On the design and development of program families. IEEE Trans. Softw. Eng., 2(1):1–9. 10

Pascual G., Roberto, E., Herrejon, L., Pinto, M., Fuentes, L., Egyed A. (2015a) Applying Multiobjective Evolutionary Algorithms to Dynamic Software Product Lines for Reconfiguring Mobile Applications, The Journal of Systems & Software, doi: 10.1016/j.jss.2014.12.041

Pascual, G., Pinto, M., Fuentes L., (2015b) Self-adaptation of mobile systems driven by the Common Variability Language, Future Generation Computer Systems, Volume 47, Pages 127-144, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2014.08.015.

Paulson.L.D. Computer system, heal thyself. Computer, 35(8):20–22, 2002.

G, Perrouin, F, Chauvel, J, DeAntoni, and JM, Jézéquel (2008) Modeling the variability space of self-adaptive applications. In 2nd International Workshop on Dynamic Software Product Lines (DSPL 2008), pages 15–22.

Perrouin, G., Klein, J., Guelfi, N., and Jézéquel, J.-M. (2008). Reconciling automation and flexibility in product derivation. In SPLC'08, pages 339–348. IEEE. 19

Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. IEEE Intelligent Systems, 14(3):54–62, 1999. □

Pine, B. J. (1999). Mass Customization: The New Frontier in Business Competition. Harvard Business School Press. 9

Pohl, K., Böckle, G., and van der Linden, F. J. (2005). Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA. 9, 10, 12, 17, 18, 30, 44, 71, 96, 205

A. Ponnappan, L. Yang, R. Pillai, and P. Braun. A policy based qos manage- ment system for the intserv/diffserv based internet. In POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02), page 159, Washington, DC, USA, 2002. IEEE Computer Society.

Pradhan, D.K.: Fault-Tolerant Computer System Design. Prentice Hall, Upper Saddle River (1996)

Prakash, T.K., Peng, L.: Performance characterization of spec cpu2006 benchmarks on intel core 2 duo processor. ISAST Trans. Comput. Softw. Eng. 2(1), 36–41 (2008)

pure::variants (2006). http://www.pure-systems.com/pure_variants.49.0.html.

Rabiser, R., Grunbacher, P., and Dhungana, D. (2007). Supporting product derivation by adapting and augmenting variability models. In SPLC '07: Proceedings of the 11th International Software Product Line Conference, pages 141–150. IEEE. 18

Riebisch, M. (2003). Towards a more precise definition of feature models. In Riebisch, M., Coplien, J. O., and Streitferdt, D., editors, Modelling Variability for Object-Oriented Product Lines, pages 64–76. BookOnDemand Publ. Co, Norderstedt. 18

Roblee C. and Cybenko G. Implementing large-scale autonomic server monitoring using process query systems. In ICAC '05: Proceedings of the Second International Conference on Automatic Computing, pages 123–133, Washington, DC, USA, 2005. IEEE Computer Society.

Rosenmuller, M., Siegmund, N., Pukall, M., Apel S. (2011) Tailoring dynamic software product lines, in: Proceedings of the 10th ACM international conference on Generative programming and component engineering, ACM, pp. 3–12.

Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S. Lorenzo, J., Mamelli, A., Scholz, U. (2009) MUSIC: Middleware support for self-adaptation in ubiquitous and service-oriented environments, Software Engineering for Self-Adaptive Systems 164–182.

Russell, S.J., Norvig P. (2010) Artificial Intelligence: A Modern Approach. Prentice Hall. p. Chapter 6. ISBN 9780136042594.

Saleem N. Bhatti and Graham Knight. Enabling qos adaptation decisions for internet applications. Comput. Netw., 31(7):669–692, 1999. □

Sanchez, P., Loughran, N., Fuentes, L., and Garcia, A. (2008). Engineering languages for specifying Product-Derivation processes in software product lines. In 1st International Conference on Software Language Engineering (SLE'08), LNCS, pages 188–207. Springer. 30

Saval, G., Puissant, J. P., Heymans, P., and Mens, T. (2009). Some challenges of featurebased merging of class diagrams. In [Benavides et al. 2009], pages 127–136. 19, 204

Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., and Bontemps, Y. (2007). Generic semantics of feature diagrams. Computer Networks, 51(2):456–479. 18, 23, 28, 53, 67, 71, 77, 101, 161

Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., and Bontemps, Y. (2007). Generic semantics of feature diagrams. Computer Networks, 51(2):456–479. 18, 23, 28, 53, 67, 71, 77, 101, 161

SEI (2011). A framework for software product line practice, version 5.0 http://www.sei. cmu.edu/productlines/frame_report/index.html. 12

Sharma. V, Thomas. A, Abdelzaher. T, Skadron. K, and Lu. Z. Power-aware qos management in web servers. In RTSS '03: Proceed- ings of the 24th IEEE International Real-Time Systems Symposium, page 63, Washington, DC, USA, 2003. IEEE Computer Society. □

Shen, L., Peng, X., Liu, J., Zhao, W. (2011) Towards feature-oriented variability reconfiguration in dynamic software product lines, Top Productivity through Software Reuse 52–68.

Shi, R., Guo, J., Wang Y. (2010) A preliminary experimental study on optimal feature selection for product derivation using knap-sack approximation, in: Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on, Vol. 1, IEEE, pp. 665–669.

P. Shivam, S. Babu, and J. S. Chase. Learning application models for utility resource planning. In ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing, pages 255–264, Washington, DC, USA, 2006. IEEE Computer Society. □

Sima, D.: Decisive aspects in the evolution of microprocessors. Proc. IEEE 92(12), 1896–1926 (2004)

Sinnema, M., Deelstra, S., and Hoekstra, P. (2006). The covamof derivation process. In Morisio, M., editor, ICSR, volume 4039 of Lecture Notes in Computer Science, pages 101– 114. Springer. 18

Sinnema, M. and Deelstra, S. (2008). Industrial validation of covamof. Journal of Systems and Software, 81(4):584–600. 18

Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Ap- proach. Pearson Education, 2003.

R. S. Sutton and A. G Barto. Reinforcement learning: An Introduction. MIT Press, 1998.

Svahnberg, M., van Gurp, J., and Bosch, J. (2005). A taxonomy of variability realization techniques: Research articles. Software Practice and Experience, 35(8):705–754. 17, 18, 30, 168, 180

Svendsen, A., Zhang, X., Lind-Tviberg, R., Fleurey, F., Haugen, Ø., Møller-Pedersen, B., and Olsen, G. K. (2010). Developing a software product line for train control: A case study of cvl. pages 106–120. 19, 204, 205

Soltani, S., Asadi, M., Gasevic, D., Hatala, M. and E. Bagheri, (2012) Automated planning for feature model configuration based on functional and non-functional requirements, in: Proceedings of the 16th International Software Product Line Conference-Volume 1, ACM, pp56–65.

R. Sterritt, B. Smyth, and M. Bradley. Pact: personal autonomic computing tools. pages 519–527, 2005.

J. W. Strickland, V. W. Freeh, Xiaosong Ma, and S. S. Vazhkudai. Governor: Autonomic throttling for aggressive idle resource scavenging. In Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on, pages 64–75, 2005.

Svein, O., Hallsteinsen, O., Jiang, S. and Sanders R. (2009) Dynamic software product lines in service oriented computing. In 3rd Int. Work. on Dynamic Software Product Lines (DSPL).

Szyperski, C., Gruntz, D., and Murer, S. (2002). Component Software - Beyond Object-Oriented Programming. Addison-Wesley. 10

Takayuki Osogami, Mor Harchol-Balter, and Alan Scheller-Wolf. Analysis of cycle stealing with switching times and thresholds. Perform. Eval., 61(4):347– 369, 2005.

Thompson, S.E., Chau, R.S., Ghani, T.,Mistry, K., Tyagi, S., Bohr,M.T.: In search of "forever," continued transistor scaling one new material at a time. IEEE Trans. Semicond. Manuf. 18(1), 26–36 (2005). doi:10.1109/TSM.2004.841816. http://dx.doi.org/10.1109/TSM.2004.841816

G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. Autonomic Computing, 2006. ICAC '06. IEEE International Conference on, pages 65–73, June 2006.

Thompson, S., Parthasarathy, S.: Moore's law: The future of si microelectronics. Mater. Today 9(6), 20–25 (2006)

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

Trinidad, P., Benavides, D., Durán, A., Ruiz-Cortés, A., and Toro, M. (2008a). Automated error analysis for the agilization of feature modeling. J. Syst. Softw., 81(6):883–896. 27, 28

Trinidad, P., Benavides, D., Ruiz-Cortés, A., Segura, S., and Jimenez, A. (2008b). Fama framework. In Proceedings of the 2008 12th International Software Product Line Conference, pages 359–,Washington, DC, USA. IEEE Computer Society. 32

Thum, T., Batory, D. S., Kastner C. (2009) Reasoning about edits to feature models. In: Proceedings of ICSE'09, Vancouver, Canada, pp. 254-264.

Tun, T. T., Boucher, Q., Classen, A., Hubaux, A., and Heymans, P. (2009). Relating requirements and feature configurations: A systematic approach. In SPLC'09, pages 201–210. IEEE. 28, 30, 31, 44, 166, 200

Van der Linden, F. (2002). Software Product Families in Europe: The Esaps & Cafe Projects. IEEE Software, 19:41–49. 12

Voelter, M. and Groher, I. (2007). Product line implementation using aspect-oriented and model-driven software development. In SPLC '07: Proceedings of the 11th International Software Product Line Conference, pages 233–242, Washington, DC, USA. IEEE Computer Society. 19, 30

Wall, D.W.: Limits of instruction-level parallelism. In: ALPLOS-IV: Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 176–188. ACM, New York (1991). doi:http://doi.acm.org/10.1145/ 106972.106991

Weiss, D. M. and Lai, C. T. R. (1999). Software product-line engineering: a family-based software development process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 12, 17

White, J., Schmidt, D., Wuchner, E., Nechypurenko A. (2007) Automating product line variant selection for mobile devices, in: Software Product Line Conference, 2007. SPLC 2007. 11th International, IEEE, pp. 129–140.

White, M., Chen, Y.: Scaled cmos technology reliability users guide. Tech. rep., Jet Propulsion Laboratory, National Aeronautics and Space Administration (2008)

White, J., Dougherty, B., Schmidt D. C. (2009) Selecting highly optimal architectural feature sets with filtered Cartesian flattening, Journal of Systems and Software 82 (8) 1268 – 1284.

Whiteson. S and Stone. P. Evolutionary function approximation for reinforcement learning. J. Mach. Learn. Res., 7:877–917, 2006.

Xu. J and. Fortes. J.A.B. Towards autonomic virtual applications in the in-vigo system. In ICAC '05: Proceedings of the Second International Conference on Automatic Computing, pages 15–26, Washington, DC, USA, 2005. IEEE Computer Society.

Yamaoka M. Matsuda S. Broda K. Kamoda, H. and M. Sloman. Policy conflict analysis using free variable tableaux for access control in web services envi- ronments. In Proceedings of the Policy Management for the Web Workshop at the 14th International World Wide Web Conference (WWW), 2005.

Yang, S., et al: 28nm metal-gate high-k cmos soc technology for high-performance mobile applications. In: Custom Integrated Circuits Conference (CICC), 2011 IEEE, pp. 1–5 (2011). doi:10.1109/CICC.2011.6055355

H. Yin, H. Qin, J. Carlson, and H. Hansson. Mode switch handling for the ProCom component model. In Proc. of CBSE 2013, Vancouver, Canada, pages 13–22. ACM, 2013.

Ji Zhang and Betty Cheng. Model-based development of dynamically adaptive software. In ICSE '06: Proceedings of the 28th international conference on Software engineering, pages 371–380, New York, NY, USA, 2006. ACM.

Ziadi, T. and Jézéquel, J.-M. (2006). Product Line Engineering with the UML: Deriving Products, chapter 15, pages 557–586. Number 978-3-540-33252-7 in Software Product Lines: Reasearch Issues in Engineering and Management. Springer Verlag. 18, 30