Dédicace

C'est avec profonde gratitude et sincères mots que je dédie ce modeste travail :

À l'homme le plus cher à mon coeur, à mon père **Nacer** :

« L'épaule solide, l'oeil attentif compréhensif et la personne la plus digne de mon estime et de mon respect. Aucune dédicace ne saurait exprimer mes sentiments, que dieu te préserve et te procure santé et longue vie. »

À la femme la plus chère à mon coeur, à ma mère **Yamina** :

" Tu m'a donné la vie, la tendresse et le courage pour réussir. Tout ce que je peux t'offrir ne pourra exprimer l'amour et la reconnaissance que je te porte."

En témoignage, je vous offre ce modeste travail pour vous remercier pour vos sacrifices et pour l'affection dont vous avez toujours m'entourée.

À mon cher frère **Younes**.
À ma chérie, ma soeur **Khadija**.
À ma chérie cousine et ma jumelle ***Amira***.
À toute ma famille.

À tous ceux qui sont chères, proches de mon coeur, et tous ceux qui m'aiment et qui auraient voulu partager ma joie...

Rechid Asma.

Remerciement

Avant tout, je tiens à remercier notre grand **Dieu** tout puissant de m'avoir guidé durant toutes mes années d'études et m'avoir donné la force, la volonté et le courage pour terminer ce projet.

Mes remerciements les plus chaleureux s'adressent à mon encadreur :

Dr. Kerdoudi Mohamed Lamine :

Je tiens à vous exprimer toute ma profonde gratitude pour votre accueil bien veillant, pour vos conseils avisés, ainsi que pour vos encouragements et vos commentaires qui ont enrichi et aidé ce travail à réussir.

Je vous en remercie très sincèrement et souhaite vous exprimer ici la plus haute estime que j'ai envers vous.

Je tiens à remercier également les membres du jury d'avoir accepté d'évaluer ce travail.

Merci encore à tous mes enseignants du département d'informatique de Biskra.

Résumé

La réutilisation de logiciel est le processus de création de systèmes logiciels à partir de logiciels existants. Elle présente de nombreux avantages en termes de temps, de coûts et de qualité. La Ligne de produit logiciels(LdP) est l'une de stratégie de la réutilisation logiciel le plus efficace dans les entreprises. Le BUT4Reuse est un Framework générique et extensible pour l'extraction de LdP. Dans le cadre de cette recherche, nous avons formulé la problématique suivante : exploiter les variantes des distributions Eclipse pour l'extraction de LdP et par la suite, la génération d'un produit Eclipse personnalisé minimal. L'objectif de ce projet est de développer un adaptateur pour les distributions d'Eclipse pour dériver un produit Eclipse personnalisé minimal. Pour cet objectif nous allons proposé un processus générique pour l'extraction de LdP qui prendre en considération les traces d'exécution de logiciels sur des cas d'utilisations dans l'objectif de connaître les éléments concrète de logiciels. Par la suite, nous allons appliqué ce processus sur Eclipse. Nous avons développé un adaptateurs pour les distributions d'Eclipse qui prendre en entrée le code source et les traces d'exécution. Enfin, nous avons évalué notre adaptateur sur des variantes d'Eclipse Kepler par le calcul de rappel et de précision et la comparaison des résultats obtenu par notre adaptateur et l'ancienne version d'adaptateur Eclipse et génération d'une version minimal d'Eclipse.

Mots clés :

LdP, Eclipse, SOA, OSGi, Réutilisation logiciel.

Abstract

Software reuse is the process of creating software systems from existing software. It has a lot of advantage in terms of time, costs and quality. The Software Product Line (SPL) is one of the most effective software reuse strategy in business. The BUT4Reuse is a generic and extensible framework for SPL extraction. As part of this research, we formulated the following problematic : exploit the variants of the Eclipse distributions for LdP extraction and subsequently, the generation of a minimal customized Eclipse product. The goal of this project is to develop an adapter for Eclipse packages to derive a minimal custom Eclipse product. For this purpose, we will propose a generic process for SPL extraction that takes into the consideration the software execution traces on use cases in order to know the concrete elements of software. Subsequently, we will apply this process on Eclipse. We had developed an adapter for Eclipse packages that take source code and execution traces as input. Finally, we evaluated our adapter on variants of Eclipse Kepler by the calculation of recall and precision and the comparison of the results obtained by our adapter and the old version of Eclipse adapter and generation of a minimal version of Eclipse.

keywords :

SPL, Eclipse, SOA, OSGi, Software Reuse,

Développement d'adaptateur pour les distributions Eclipse

RECHID Asma

Table des matières

Liste des figures

Liste des tables

Liste des code source

Introduction générale

La réutilisation de logiciel est le processus de création de systèmes logiciels à partir d'un logiciels existants, plutôt que de construire des systèmes logiciels à partir de zéro. Cette vision simple mais puissante a été introduit en 1968. Le passage au développement basé sur la réutilisation répond à la demande de réduction des coûts de production et de maintenance des logiciels, de la livraison plus rapide des systèmes et de l'amélioration de la qualité des logiciels. De plus en plus les entreprises encouragent la réutilisation pour augmenter le rendement de leurs investissements en logiciels. Il existe plusieurs approches de la réutilisation logiciel tel que **Design Patterns**, **Model Driven Engineering**, **Service oriented Systems**, **ligne de produit logiciel**.

La tendance actuelle des entreprises et des organisations est l'informatisation des lignes de produits logiciels(LdP). Une LdP est définie comme un ensemble de systèmes partageant un ensemble de propriétés communes et satisfaisantes des besoins spécifiques pour un domaine particulier [11]. Par exemple, Eclipse fournit des distributions qui ciblent différents profils de développeur tel que les testeurs, les développeurs des logiciels automobiles, Java EE, c/c++... Il existe cependant des cas où un grand nombre de produits similaires et dans le même domaine sont modélisés séparément, sans prise en compte la notion de LdP et de la variabilité dès le départ. L'extraction de LdP manuelle induit un coût élevé et pendre beaucoup de temps.

Dans ce contexte, il existe le Framework BUT4Reuse (Bottom-Up Technologies for Reuse)[37] pour l'extraction automatiquement de LdP à partir de plusieurs produits logiciels similaires existants. L'idée est d'analyser statiquement (l'analyse du code source) ces variantes du logiciel pour construire une ligne de produits en identi-

fiant les éléments communs et les éléments variables et ne prend pas en considération les traces d'exécution de ces artefacts. Cependant les traces d'exécution viennent après l'analyse dynamique qui fournissent des informations concrets et précises sur le comportement des artefacts, où ces derniers sont exécutés plusieurs fois selon un ensemble de scénarios.

BUT4Reuse permet de définir un composant adaptateur spécifique pour chaque type d'artefact, à savoir : code source, image, modèle, son, etc. Cet adaptateur est responsable à la décomposition d'un artefact en des éléments constitutifs et permet de définir comment un ensemble des éléments doivent être regrouper pour construire une entité réutilisable.

Les distributions Eclipse sont un exemple de variants logiciels chacune est dédiée à un type spécifique de développeur. La problématique que nous procurons, fait référence à exploiter ces variantes pour construire une ligne de produit logiciels et par la suite, la génération d'un produit Eclipse personnalisé minimal. Ce dernier est composé d'un nombre réduit de plugins qui assurent uniquement les besoins de l'utilisateur.

Notre objectif consiste à réaliser un adaptateur pour les distributions d'Eclipse en exploitant les traces d'exécution de ces derniers. Pour cela :

1. Nous allons proposer un processus pour l'extraction de LdP à partir d'un ensemble de variantes d'application.
2. Nous allons proposer un méta-modèle pour regrouper les informations des traces d'exécution des distributions d'Eclipse. Les modèles des traces sont créés conformément à ce méta-modèle.
3. Nous allons appliquer notre processus pour la génération d'un produit Eclipse personnalisé minimal. Pour cela, nous allons implémenter un adaptateur pour les distributions d'Eclipse. Il a comme entrée le code source et Byte-code des distributions (variants) Eclipse et les traces d'exécution qui sont générées en exécutant un ensemble de use cases.
4. Pour l'évaluation de notre processus, nous allons mener une expérimentation en utilisant des variantes d'Eclipse Kepler, le calcul de rappel et de décision et la génération d'une version minimal en comparant ces résultats avec celle

de l'ancienne adaptateur.

Ce mémoire est organisé comme suit :

- Le premier chapitre s'intitule : " Ligne de produit logiciels", nous introduisons les stratégies du réutilisation logiciels. Par la suite, nous concentrons sur l'approche de ligne de produit logiciels où on présente les points essentiels relatifs à l'ingénierie des lignes de produits logiciels.
- Le deuxième chapitre s'intitule : " Buttom Up for Reuse", nous présentons le Framework générique et extensible pour l'extraction de LdP BUT4Reuse.
- Le troisième chapitre s'intitule : " Spécification OSGi et Eclipse", nous présentons le Framework OSGi et les concepts relatifs à la spécification OSGi. En outre, nous exposons Eclipse qui utilise l'implémentation de la spécification OSGi Equinox.
- Le quatrième chapitre s'intitule : " Développement d'un adaptateur pour les distributions d'Eclipse ", nous présentons le travail réalisé qu'il s'agit d'un processus générique pour l'extraction de LdP pour n'importe quel type d'artefact application, l'application de ce dernier pour les distributions d'Eclipse, un méta-modèle collecte les informations des traces d'exécution et une implémentation d'adaptateur des distribution d'Eclipse.
- Le cinquième chapitre s'intitule : "Évaluation", nous allons expérimenter et évaluer notre adaptateur sur les distributions d'Eclipse Kepler en calculant le rappel et la décision.
- A la fin, nous terminons notre travail par une conclusion générale.

Chapitre 1

Ligne de Produit Logiciels

1.1 Introduction

Une ligne de produit logiciel(LdP) est un ensemble de produits logiciels qui sont construits d'une manière prescrite à partir d'un ensemble d'artefacts. Ces derniers doivent être liées entre eux de façon explicite et planifié. Le but général de LdP est la minimisation des coûts, ainsi que les délais de développement pour les mettre dans le marché et la construction des produits personnalisés.

Dans ce chapitre, nous allons présenter brièvement les techniques de réutilisation. Par la suite, nous allons focaliser sur les lignes de produits logiciels, en décrivant le processus de l'ingénierie de ligne de produit, ainsi que les types de variabilité gérés dans ce processus. Ce chapitre couvre également les approches de modélisation de variabilité et les approches existantes pour adopter une ligne de produit logiciels.

1.2 Réutilisation du logiciel

La réutilisation est une stratégie d'ingénierie logicielle. La réutilisation de logiciel est le processus de création d'un nouveau logiciel en réutilisant des morceaux de logiciel déjà existants plutôt qu'en partant de zéro[1]. Le développement basé sur la réutilisation a un avantage en terme de réduction des coûts de développement et de maintenance des logiciels, et en terme de livraison rapide et de l'amélioration de la qualité des logiciels [2]. Depuis l'année 1960 où la réutilisation de logiciel est proposée jusqu'à aujourd'hui plusieurs approches ont été proposées. La Figure 1.1

technique synthétise les techniques les plus populaires qui sont :

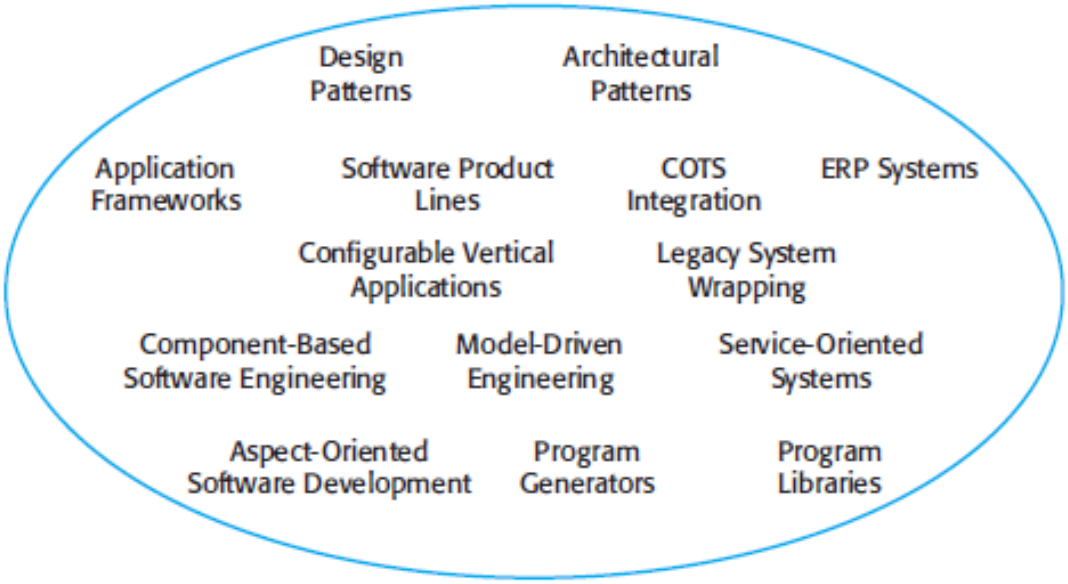


FIGURE 1.1 – Techniques de la réutilisation[1].

— **Design Patterns** : une conception générale qui répond à un problème de conception récurrent dans les systèmes orientés objet. Il décrit le problème, la solution, et leurs applications ses conséquences. Il donne également des astuces de mise en ouvre et des exemples. La solution est un arrangement général des objets et des concepts qui résoudre le problème. La solution est personnalisée et mise en ouvre pour résoudre le problème dans un contexte particulier. Gamma, Helm, Johnson et Vlissides décrivent vingt-trois patterns tel que : Factory, Abstract Factory, Prototype, Composite [3].

— **Architectural Patterns (Style)** : Description abstraite d'une architecture logicielle testée et testée dans différents systèmes logiciels. La description du modèle inclut des informations sur l'utilisation appropriée du modèle et l'organisation des composants de l'architecture[1]. Il existe une variante des patterns architectural tel que Model-view-Controller (MVC est un pattern architectural destiné aux interfaces graphiques)[3], Entity-component-system (ECS est un pattern architectural utilisé dans le développement de jeux) Service-oriented architecture (SOA).

— **Application Frameworks** : Un ensemble de classes concrètes et abstraites réutilisables qui implémentent des fonctionnalités communes à de nombreuses applications dans un domaine. Les classes du framework d'application sont spécialisées et instanciées pour créer une application. Les Applications Frameworks sont utilisées dans le développement de l'interface utilisateur graphique, les applications Web. NET Framework (NET Fra facilite la tâche des développeurs en proposant une approche unifiée à la conception d'applications Windows ou Web) ,Enterprise JavaBeans (EJB) est une architecture de composants logiciels côté serveur pour la plateforme de développement Java EE[1].

— **COST Intégration** : Un produit COTS (Commercial Off-The-Shelf)est généralement un matériel informatique ou un logiciel conçu pour des utilisations spécifiques et mis à la disposition du grand public. Par exemple Microsoft Office ou un logiciel antivirus. Les systèmes COTS intégrés sont des applications qui incluent deux produits COTS ou plus, ou parfois des systèmes d'application hérités. On utilise cette approche lorsqu' il n'existe pas de système COTS qui répond à tous nos besoins, en intégrant un nouveau produit COTS avec les systèmes que vous avez utilisé déjà.

— **Configurable Vertical Application** : Un système générique est conçu pour pouvoir être configuré pour les besoins des clients du système spécifique[1].

— **Legacy Systems Wrapping** : les systèmes existants sont enveloppés(Wrapping) en définissant un ensemble d'interfaces et en fournissant un accès à ces systèmes hérités via ces interfaces[4]. Par exemple la création de services Web à partir de code hérité[5].

— **Enterprise Resource Planning(ERP) Systems** : les systèmes de planification des ressources d'entreprise ou systèmes d'entreprise sont des systèmes logiciels de gestion d'entreprise comprenant des modules des domaines fonctionnels tels que la planification, la fabrication, les ventes, le marketing, la distribution, la comptabilité, les finances, la gestion des ressources humaines et la gestion de projet[6]. Les systèmes ERP sont configurés pour répondre aux besoins de chaque entreprise utilisant le système

— **Component Based Software Engineering** : Les systèmes sont dévelop-

pés en intégrant des composants (collections d'objets) conformes au modèles de composants[7]. Il existe de nombreux modèles de composants par exemple CORBA Component Model(CCM), FRACTAL component model, and Service Component Architecture (SCA).

— **Model Driven Engineering** : l'ingénierie dirigée par les modèles est une approche de développement logiciel qui considère les modèles comme l'élément le plus important dans le processus de développement. Le code est (semi-) automatiquement généré à partir de ces modèles. La maintenance et l'évolution de logiciels à travers l'utilisation des langages de transformation des modèles [8]. Quelques langages de transformations de modèles : Query/View/ Transformation(QVT), Atlas(ATL), Kermeta.

— **Service oriented Systems** : l'architecture orientée services est un style architectural pour les systèmes distribués, il rend la fonctionnalité de l'application à fournir sous forme d'un ensemble de composants appelé services pouvant être invoqués, publiés et découverts via un contrat. Ce dernier fournit une spécification de l'objectif, des fonctionnalités, des contraintes de l'utilisation des services[9].

— **Programs Libraires** : Une bibliothèque de logiciels est un ensemble de données et du code de programmation utilisée pour développer des programmes logiciels et des applications. Par exemple les API Java, Google Maps API.

— **Program Generator** : un générateur de programme qui permet à une personne de créer son propre programme avec moins d'effort et de connaissances en programmation. Avec un générateur de programme, un utilisateur peut uniquement être invité à spécifier les étapes ou les règles requises pour son programme et ne pas avoir besoin d'écrire de code ou très peu de code[10]. Il existe une variété d'outils pour la génération d'interfaces de base de données,d'interfaces graphiques tel que Next-Gen UI(NGUI), Skeleton-Generator (beta).

— **Software Product line (SPL)** : Une ligne de produit logiciels est constituée d'une famille d'applications apparentées appartenant à la même organisation[1]. Dans notre travail, on va concentrer sur l'approche de ligne de produit logiciels.

1.3 Définition de ligne de produit logiciel

"*A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.*"[11]

La ligne de produits logiciel est un ensemble de produits logiciels (images, applications, modèle...) qui partage et gère un ensemble des caractéristiques nommés "Features" (on va utiliser dans le reste de ce mémoire le terme Feature) qui réponde aux besoins spécifiques d'un segment de marché ou une mission particulière, et qui sont développées d'une manière prescrite à partir d'un même ensemble assets de base.

Assets (assets de base) : sont les artefacts et les ressources réutilisables qui constituent la base de la ligne de produits logiciels. Ces assets incluent souvent, l'architecture, les composants logiciels, les modèles de domaine, les exigences, la documentation, les spécifications, les modèles de performance, les calendriers, les plans et les scénarios de test, les plans de travail et les descriptions de processus. L'architecture d'un produit est la clé pour collecter les assets[12].

1.4 Ingénierie de ligne de produit logiciels

L'ingénierie des lignes de produits logiciels est une méthodologie permet le développement de plusieurs produits logiciels qui ont des similarités[13]. Il ne s'agit plus de modéliser et de développer des systèmes individuels. L'ingénierie des lignes de produits logiciels s'intéresse plutôt à modéliser et développer une famille de produits avec un gain en terme de minimisation des coûts, délais de développement et de mise en marché[14].

1.5 Motivation

De nombreuses raisons poussent les entreprises à se pencher dans une approche d'ingénierie de ligne de produits logiciels. Celles-ci vont des aspects plus orientés

processus, tels que le coût et le temps, aux qualités du produit, telles que la fiabilité, qu'aux utilisateurs finaux, tels que la cohérence de l'interface utilisateur. L'évolution vers une ingénierie de ligne de produits logiciels est généralement fortement basée sur des considérations économiques. Elle réduit les coûts, les délais de mise sur le marché et améliore les qualités des produits résultants, telles que leur fiabilité. Les ligne de produits logiciels sont un paradigme mature pour la gestion de la variabilité en génie logiciel[15, 16] . Ils permettent de définir une famille de configurations de produits et de plus tard, générer systématiquement les variantes de produit associées. L'inspiration pour proposer cette pratique d'ingénierie est communément attribuée à l'industrie manufacturière où différents Les composants réutilisables prédéfinis sont généralement combinés pour répondre aux différents besoins des clients.

1.6 Processus générique de l'ingénierie des LDP

L'ingénierie des lignes de produits logiciels distingue deux étapes complémentaires illustrés dans la Figure1.2 : l'ingénierie du domaine et l'ingénierie d'application[16]. L'ingénierie du domaine consiste à développer les assets qui seront réutilisés pour la construction des produits. L'ingénierie d'application consiste à utiliser les assets pour la construction d'un produit ou d'une application particulière.

Un domaine : est un ensemble de connaissances spécialisé, un domaine d'expertise ou un ensemble de fonctionnalités connexes. Par exemple, le domaine des télécommunications est un ensemble de fonctionnalités de télécommunications qui, à son tour, comprend d'autres domaines tels que la commutation, les protocoles, la téléphonie et les réseaux. Une ligne de produits logiciels de télécommunications est un ensemble spécifique de systèmes logiciels fournissant certaines de ces fonctionnalités[12].

Dérivation de produit : est l'activité de réalisation d'un produit singulier par composition, adaptation et spécialisation d'artefacts partagés par la ligne de produits, selon un plan de production prédéfini [30]. La dérivation de produit de la phase de l'ingénierie d'application est soutenue par le développement par la réutilisation des assets construits dans l'étape de l'ingénierie du domaine.

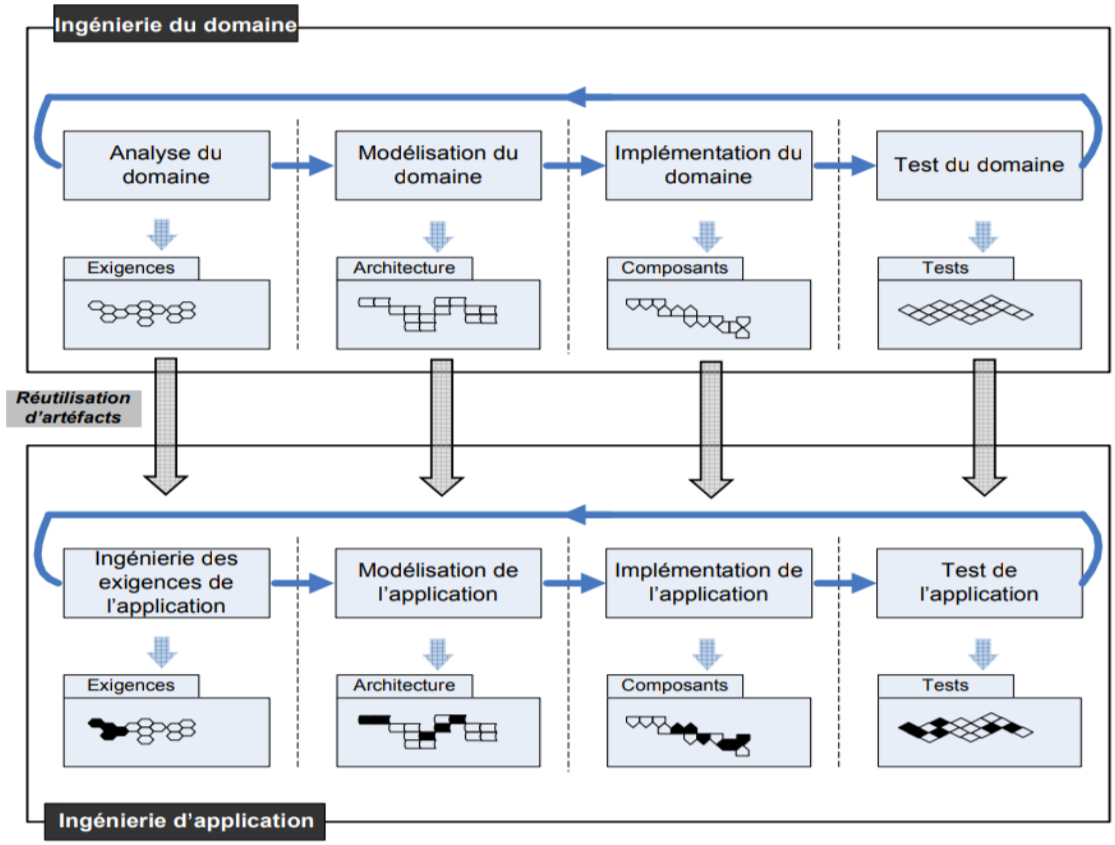


FIGURE 1.2 – Processus générique de ligne de produit logiciels [16].

1.6.1 Ingénierie du domaine (développement pour la réutilisation)

L'étape de l'ingénierie du domaine vise à définir la portée de la ligne de produits logiciels, en définissant et planifiant les points variantes et les points communs entre les membres de la ligne de produit. Par exemple les exigences, les composants et les plans de test, qui seront construits de façon à ce qu'ils soient réutilisés dans les produits de LdP. L'ingénierie du domaine est composée de quatre sous-étapes sont l'analyse, la modélisation, l'implémentation et le test du domaine.

1. **Analyse du domaine** : on analyse les exigences du domaine pour identifier celles qui sont communes à tous les produits de la ligne de produit et celles qui sont spécifiques à des produits particuliers.
2. **Modélisation du domaine** : cette sous-étape définit l'architecture global

de la ligne de produit, Cette architecture donne une vue structurelle sur les différents membres de la ligne de produit logiciels et identifier les parties réutilisables de de cette architecture.

3. **Implémentation du domaine** : dans cette sous-étape les différentes parties réutilisables de l ?architecture sont détaillées et implémentées dans des composants logiciels réutilisables sont les assets de base.
4. **Test du domaine** : cette sous-étape est responsable de la validation et la vérification des assets de base obtenus . Les tests effectués sur les composants se font par rapport à leur conformité aux spécifications de besoins.

1.6.2 Ingénierie d'application (développement par la réutilisation)

L'étape d'ingénierie d'application est responsable à la dérivation d'un produit logiciel (application) par la réutilisation des assets de base construit dans l'ingénierie du domaine. On sélectionne les assets de base qui sont présents dans ce produit spécifique. L'ingénierie d ?application est composée de quatre sous-étapes sont ingénierie des exigences, la modélisation, l'implémentation et le test de l'application, dont chaque sous- étape utilise des artefacts du domaine et produit des artefacts d'application.

1. **Ingénierie des exigences d'application** : cette sous-étape spécifie les exigences pour une application particulière en exploitant les exigences du domaine et peut être d'autre exigences spécifiques liées à un client qui ne sont pas capturés pendant l'analyse de domaine.
2. **Modélisation de l'application** : cette sous-étape permet elle permet d'obtenir l'architecture du produit à partir de l'architecture globale définie pendant la modélisation du domaine. Pour ce faire, les parties requises du modèle sont sélectionnées et incorporées.
3. **Implémentation de l'application** : consiste à la réalisation du produit concret souhaité. Les composants qui correspondent aux parties sélectionnées dans l'étape précédente sont alors assemblés.

4. **Test de l'application** : cette sous-étape permet de vérifier et valider l'application par rapport à sa spécification en exploitant les tests des assets de base sélectionnées pour cette application.

1.7 Variabilité de ligne de produit logiciels

La variabilité est une propriété essentielle dans l'ingénierie de la ligne de produits. La variabilité dans une ligne de produit logiciels est définie dans l'ingénierie du domaine à chaque niveau d'abstraction (exigences, architecture, composants, cas de test, etc) de leur sous-étapes par l'introduction des points de variation. Ces derniers servent à identifier et localiser l'endroit où se produit la variabilité et précise les solutions possibles pour résoudre cette variabilité. La variabilité est exploitée pendant l'ingénierie d'application en liant les variables appropriées à un produit spécifique[17].
Il y' a plusieurs définitions de la variabilité dans lignes de produits logiciels, on sélectionne les définitions les plus pertinentes :
Selon Weiss et Lai ont donné la définition de la variabilité comme :
"...An assumption about how members of a family may differ from each other" [17]
On peut expliquer ce passage comme suit : la variabilité est définie comme une hypothèse sur la façon dont les membres d'une famille peuvent se différencier entre eux.
Et selon Felix Bachmann et Paul Clements ont donné la définition de la variabilité comme : *"variability means the ability of a core asset to adapt to usages in the different product contexts that are within the product line scope"*[18]

Dans cette définition la variabilité est la capacité d'un asset de base à s'adapter aux usages dans les différents contextes de produits dans LdP.

1.7.1 Classification de la variabilité

La variabilité dans la ligne de produits logiciels est classifiée en quatre catégories[16] :

Variabilité dans le temps

La variabilité dans le temps est l'existence de différentes versions d'un artefact qui sont valables aux différents moments. La dimension temporelle de la variabilité couvre le changement d ?un artefact variable dans le temps, elle est synonyme de l ?évolution du logiciel. Par exemple la fondation Eclipse créé chaque année une nouvelle version tel que Kepler, Neon, Mars.

Variabilité dans l'espace

La variabilité dans l'espace est l'existence d'un artefact à des différentes formes à la fois. Par exemple pour le produit Eclipse, il y a Eclipse pour les testeurs, les développeurs en c++...

Variabilité externe

La variabilité externe est la variabilité des artefacts visible par les clients. Les clients demandent à développer des applications personnalisées en fonction de leurs besoins, cela implique que les clients doivent connaître au moins une partie de la variabilité de ligne de produits logiciels.

Variabilité interne

La variabilité interne est celle qu'est invisible pour les clients et sa définition et résolution est une responsabilité du l'organisation.

1.8 Approche de modélisation de variabilité par les features

L'ingénierie de ligne de produits logiciels consiste à produire un ensemble de produits associés qui partagent des points communs et des points variabilités. Le feature model est largement utilisé pour représenter les points communs et les points de différences au niveau d ?exigence dans la ligne de produits logiciels. Le feature model est un modèle d'informations dans lesquels l'ensemble de produits de la LdP

est représenté sous forme des entités correspondant aux assets de base qui construit la LdP[19].

1.8.1 Définition de feature

On va présenter deux définitions celle de Kang et l'autre de Apel.
"*A prominent or distinctive and user-visible aspect, quality, or characteristic of a software system or systems*"[20].
"a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder's requirement, to implement and encapsulate a design decision, and to offer a configuration option"[21]

1.8.2 Feature Model

Le feature model (FM) est introduit dans l'approche FODA (Feature Oriented Domain Analysis). Le FM est une description graphique sous forme d'une hiérarchie des exigences d'une ligne de produits logiciels [10]. Il représente les choix à faire pour déterminer les features des produits envisagés[22, 23]

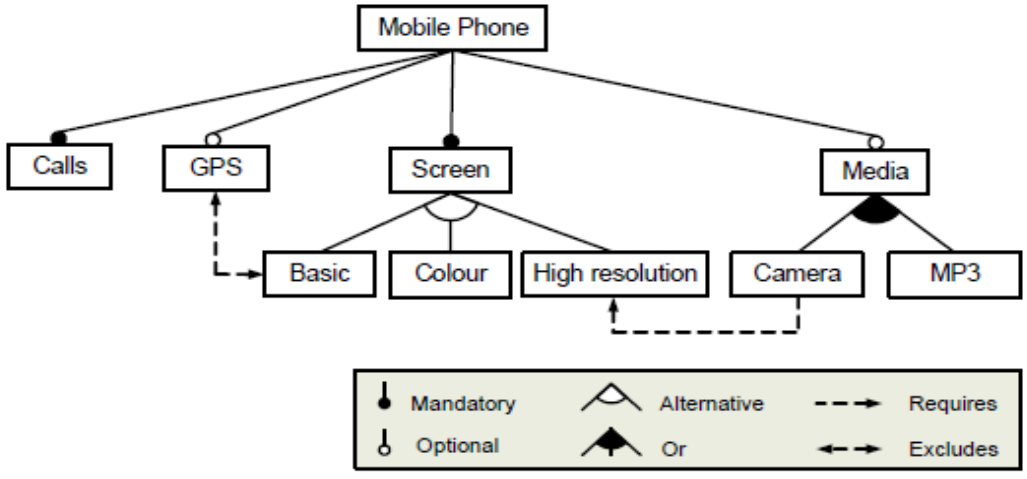


FIGURE 1.3 – Exemple de feature model[19].

La figure 1.3 présente un exemple de feature model liée à un produit Mobile phone qui représente le feature racine de ce FM. Cette feature(feature parent) est

composée par des sous features(features enfants) qui sont elles-mêmes composées par d'autre sous features jusqu'à atteindre les features feuilles.
Des différents symboles graphiques permettent de distinguer les relations entre les features sont :

- **Mondatory** : Une feature enfant a des relations obligatoires avec son parent lorsque l'enfant est inclus dans tous les produits dans lesquels sa caractéristique parente apparaît. Par exemple, chaque téléphone mobile dans notre exemple doit fournir de calls.
- **Optionel** : Une feature enfant a une relation optionnelle avec son parent quand, l'enfant peut être éventuellement inclus dans tous les produits dans lesquels le parent apparaît. Dans l'exemple, le logiciel pour les téléphones mobiles peut éventuellement inclure un support pour GPS est optionnel et n'est pas obligatoire sa dépend le mobile phone en question.
- **Alternative** : Un ensemble de features enfants ont une relation alternative avec leur parent. Une feature enfant peut être sélectionnée lorsque son parent fait partie du produit. Dans cet exemple, un téléphone mobile doit inclure Screen qui peut inclure l'un des supports : basic, colour ou high resolution.
- **relation Or** : Un ensemble de features enfants ont une relation Or avec leurs parents quand un ou plusieurs d'entre eux peuvent être inclus dans les produits dans lesquels sa feature mère apparaît. Dans la figure 1, chaque fois que Media est sélectionné, Camera, MP3 ou les deux peuvent être sélectionnés.

Notez qu'une feature enfant ne peut apparaître que dans un produit, si sa feature parent inclus. La feature racine est une partie de tous les produits de la famille de produits logiciels.

Outre les relations entre les features, un feature model peut également contenir des contraintes d'arbre croisé entre les features. Ces relations sont de forme :

- **Require** : Si une feature A nécessité une feature B, l'inclusion de A dans un produit implique l'inclusion de B dans ce produit. Les phones mobiles, qui ont Camera doivent inclure un support d'une High resolution Screen.
- **Excludes** : Si une feature A exclu une feature B, les deux features ne peuvent pas faire partie du même produit. Par exemple le GPS et Basic Screen sont

des caractéristiques incompatibles.

1.9 Approches d'adoption d'une ligne de produits logiciels

La stratégie d'adoption de LDP dépend fortement du scénario de la société[24, 25]. Krueger a distingué trois catégories d'adoption de ligne de produit sont : proactive, réactive et extractive[25] :

1.9.1 Approche proactive

Une LDP est conçue en tenant compte des besoins actuel et futur des clients. L'organisation identifie et analyse de manière proactive l'ensemble complet des features envisagées et crée les assets réutilisables correspondants.

1.9.2 Approche réactive

Contrairement à l'approche proactive où l'ensemble du LDP doit être créé avant le début de la production, l'approche réactive vise à créer une LDP opérationnel minimal, puis à l'étendre progressivement pour l'adapter aux nouveaux besoins des clients. En termes d'effort, l'approche réactive nécessite moins d'effort initial que l'approche proactive.

1.9.3 Approche extractive

Les variantes de produits existantes sont utilisées pour identifier les features et les assets réutilisables afin de créer la ligne de produit logiciel. L'exploitation des actifs existants est un domaine de pratique permettant d'établir la capacité de production de LDP et de l'exploiter. La figure1.4 illustre les activités importantes dans l'adoption de LDP extractive[26] :

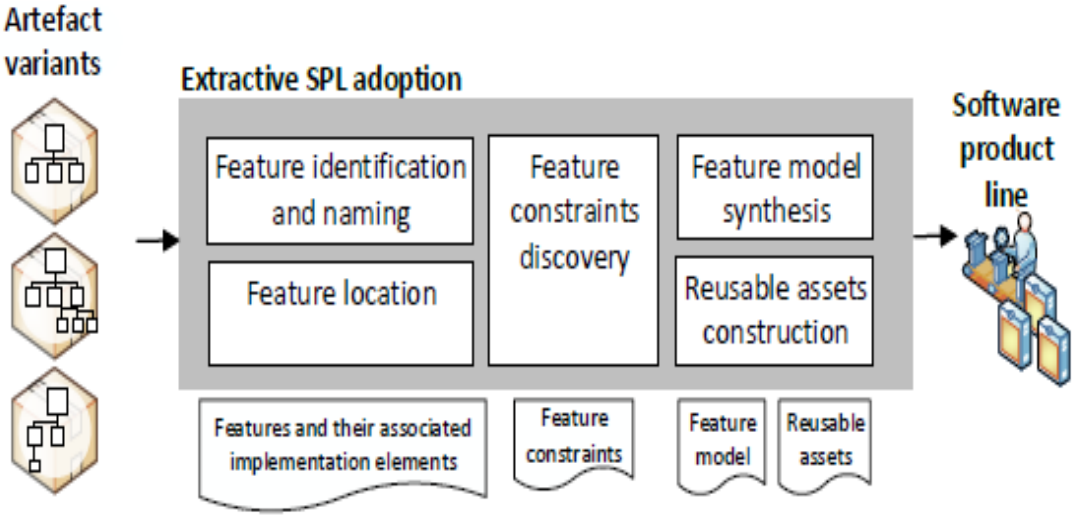


FIGURE 1.4 – Activités importes pour l'adoption de ligne de produit logiciels extractive [26].

— **Feature identification and naming** : cette activité sert à obtenir les features de chaque artefact entré et les éléments d'implémentation de chaque feature, et associé un nom à chaque feature pour utiliser dans le FM. Cette activité est très importante pour accomplir les connaissances nécessaires pour les artefacts en cas les informations pour un produit ne sont pas disponible.

— **Feature location** : cette activité permet d'identifier les éléments de mise en ouvre de chaque feature.

— **Feature constraints discovery** : cette activité est importante pour garantir la validité des configurations. Une configuration est une sélection de features qui répondent à toutes les contraintes qui sont les relations entre les features dans le FM.

— **Feature model synthesis** : cette activité consiste à définir la structure du modèle de caractéristique pour la ligne de produit logiciel, le modèle doit être compréhensible que possible pour les parties prenantes du LDP.

— **Reusble assets construction** : cette activité permet créer des assets réutilisable à partir des éléments d'implémentation associés à chaque feature obtenue lors de l'identification ou de localisation des features

1.10 Conclusion

Dans ce chapitre, nous avons présenté l'approche d'ingénierie de ligne de produit logiciels. C'est une approche qui se base sur la réutilisation systématique des artefacts logiciels existants pour développer des produits logiciels spécifiques à un domaine et des utilisateurs particulières. Nous avons illustré le processus générique de l'ingénierie de ligne de produit logiciels. Ensuite nous avons expliqué le concept variabilité et de modélisation de la variabilité par les features. En outre, nous avons montré les approches d'adaptation de LdP.

Dans le chapitre suivant, nous allons présenter l'état de l'art sur le domaine de ligne de produits logiciels par le Framework proposé pour supporter cette approche.

Chapitre 2

BUT4Reuse

2.1 Introduction

Dans le chapitre précédant, nous allons présenter l'approche extractive des lignes de produits logiciels, dont les variantes de produits existantes sont utilisées pour identifier les features et les assets réutilisables afin de créer la ligne de produit logiciel.

Dans ce chapitre, nous allons présenter le Framework BUT4Reuse (Bottom-Up Technologies for Reuse).

2.2 Framework générique et extensible pour l'adaptation de LdP extractive

Le but de ce Framework est de concevoir un environnement commun pour unifier les activités de processus d'adaptation de LdP extractive (descendante), où il y'a les produits qui sert à définir la ligne de produit par l'amélioration et la proposition des techniques pour l'identification, la localisation, la découverte de contraintes, la synthèse de features model et la construction d'assets réutilisables. Ce chapitre est basé sur le travail [26, 37].

2.2.1 Principes du Framework But4Reuse

Les principes de ce Framework pour l'adaptation de LdP extractive sont mis de façon générique et extensible pour supporter plusieurs type d'artefacts. Cette approche repose sur trois principes suivants :

1. Un artefact logiciel typique peut être décomposé en éléments distinctifs appelés "éléments".
2. calcule de similarité entre chaque paire d'éléments.
3. Construction des assets réutilisable à partir des éléments récupérés depuis les artefacts existants.

Le principe 1 et 2 permettant d'identifier les points communs et variables entre les variante d'un type d'artefact, qui seront exploités dans le processus d'identification et de localisation des features. L'approche LdP extractive par ce Framework mis en oeuvre un composant principale nommé **adaptateur**.

2.2.2 Conception d'un adaptateur pour un type d'artefact

L'adaptateur est responsable de décomposer chaque type d'artefact en éléments constitutifs et définir comment un ensemble des éléments doivent être regrouper pour construire un asset réutilisable. La Figure 2.1 illustre des exemples des adaptateurs traitant différents types d'artefacts.

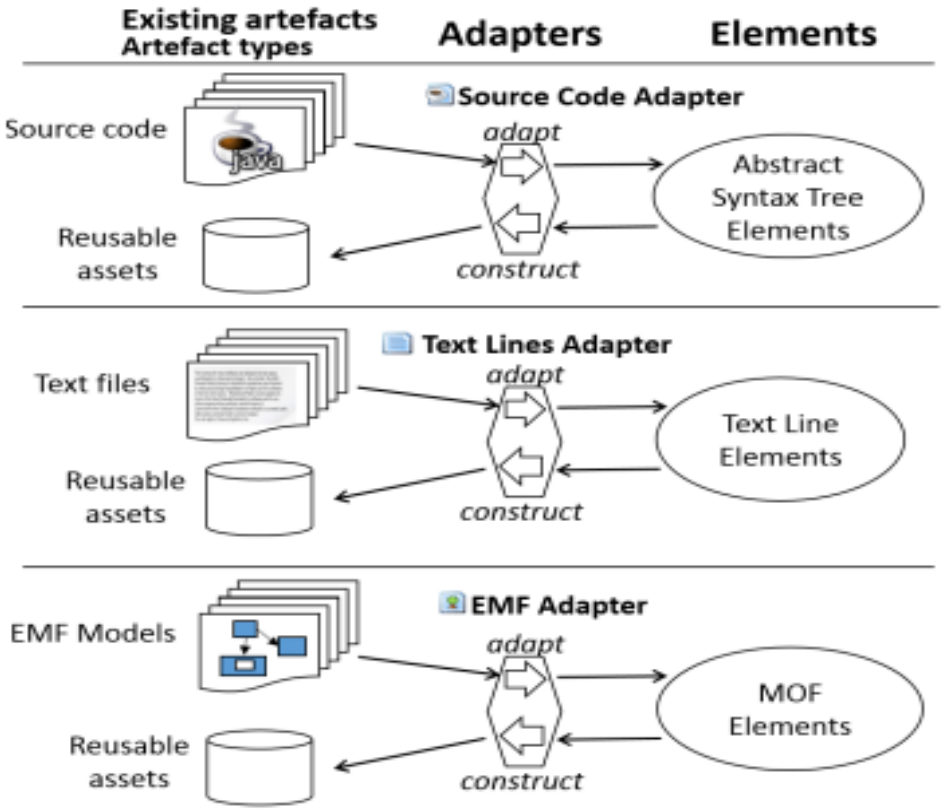


FIGURE 2.1 – Exemples de types d'artefact et création de la représentation d'éléments via les adaptateurs [37].

Selon cette approche, la conception d'un adaptateur pour un type d'artefact donné nécessite les activités suivantes :

— **Activité 1 : identification des éléments :**
l'identification des éléments qui composent un artefact. Cela définira la granularité des éléments dans un type d'artefact donné.

— **Activité 2 : définition des contraintes structurelles :**
identifier les dépendances structurelles des éléments.

— **Activité 3 : définition de la métrique de similarité :**
définition d'une métrique de similarité entre n'importe quelle paire d'éléments. Un élément doit comparer sa similarité avec un autre élément on obtient une valeur allant de zéro (complètement différent) à un (identique). Il n'y a pas de règle générale pour calculer la similarité.

— **Activité 4 : construction d'assets réutilisables :**

définir comment utilisez un ensemble d'éléments pour construire des actifs réutilisables.

Adaptateur d'images

La Figure 2.2 montre l'application des activités de conception d'un adaptateur pour concevoir un adaptateur d'image. Dans l'activité 1, une image est composée de éléments de pixel (PixelElements). L'adaptation d'une image en éléments pixels consiste à charger la matrice de pixels et à ajouter les pixels non transparents au format PixelElement. Dans l'activité 2, chaque élément pixel, aura une dépendance structurelle avec sa position où un seul pixel est autorisé pour chaque position. Cela permet de découvrir automatiquement les contraintes structurelles (par exemple, le chevauchement des pixels n'est pas autorisé, donc deux chemises ne peuvent pas être utilisées dans la même image). Dans l'activité 3, la métrique de similarité consiste à vérifier que les positions des éléments Pixels sont exactement les mêmes, puis à calculer la similarité de la couleur et canal alpha. Dans l'activité 4, la construction d'un asset réutilisable consiste à ajouter les pixels à leurs positions correspondantes dans une image transparente.

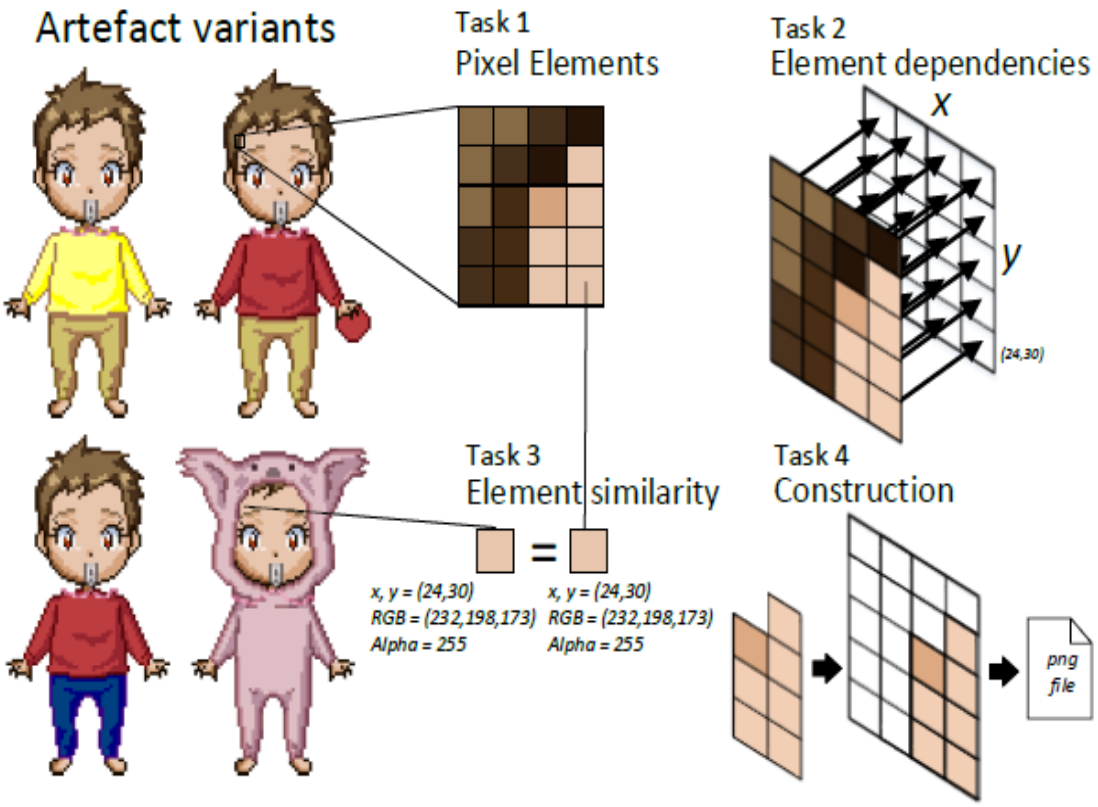


FIGURE 2.2 – Conception de l'adaptateur images. [26]

La Figure 2.3 présente le résultat de l'application de l'adaptateur des images sur des variantes d'images, l'adaptateur prendre comme une entrée les images par la suite, il identifie les blocs (ensembles d'éléments) et les contraintes entre eux, puis en synthèse le modèle de features à partir des blocs et les contraintes identifie, en fin la construction d'un nouvel artefact par la sélection des blocs qui on a besoin en respectant les contraintes entre ces blocs.

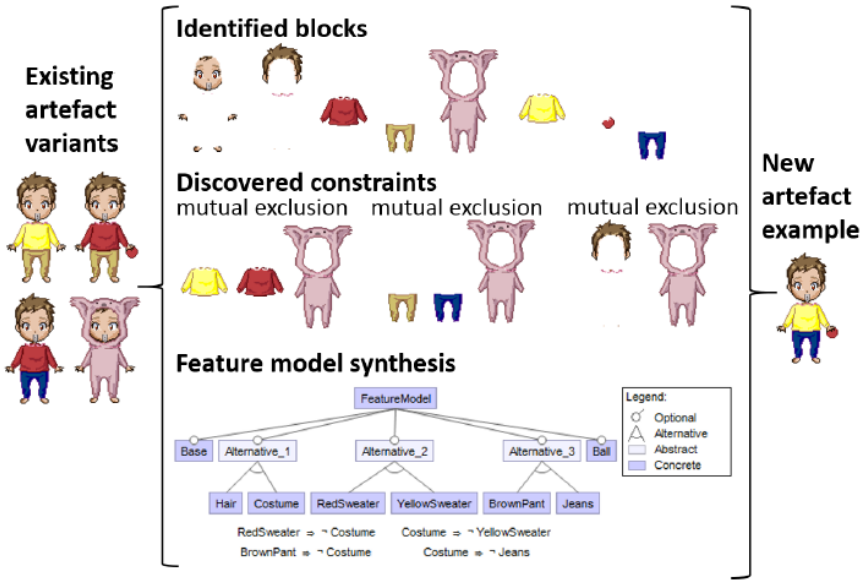


FIGURE 2.3 – Exemple de variantes d'image et résultat de l'application de l'adaptateur d'images[37].

Adaptateur d'Eclipse

L'entrée de l'adaptateur Eclipse est le dossier d'installation d'Eclipse. Les éléments identifiés sont PluginElement pour les plugin et FileElement pour tous le fichier d'installation d'Eclipse. La similarité entre les PluginElement est faite par la comparaison de l'identificateur unique de plug-in (SymbolicName), la similarité entre les FileElement a été implémentée en comparant leur URI relatif résolu par rapport au dossier racine Eclipse. L'identifiant attribué à ce type de dépendance est Require-Bundle, où le plugin définis dans sa fichier manifast ces dépendances statiques se forme de Require-Bundle(plugins), par conséquence, un PluginElement dépendra structurellement des autre PluginElement. La construction des assets est faite en copiant les plugins et les fichiers associés à chaque bloc.

Adaptateur de code source Java et C

L'entrée de l'adaptateur des code sources est un dossier contenant le code source. Les éléments identifiés sont FSTNonTerminalNode représentent les noms du packages et classes et les éléments FSTTerminalNode représentent les attributs et les méthodes[39]. La métrique de similarité utiliser est la comparaison de positions et

de noms feature structure trees (FST). Les Dépendances des nœuds FST et dépendances du code source détectées avec Puck [40].

2.2.3 Activités pour l'adoption de la LdP extractive avec le Framework

Le Framework d'adoption de LdP extractive s'appuie sur une architecture composée de différentes activités comme il est représenté sur la Figure 2.4, dont l'entrées sont les variantes d'artefact image et la sortie c'est la ligne de produit logiciels.

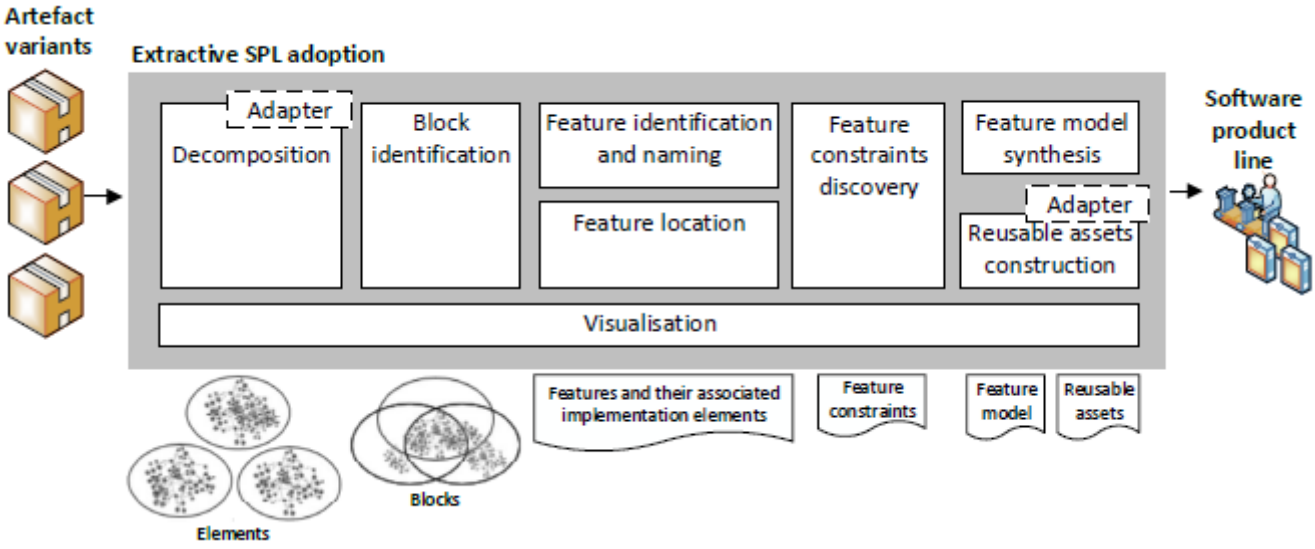


FIGURE 2.4 – Activités pertinentes lors de l'adoption de LdP extractive dans BUT4Reuse.[26]

Les activités d'adaptation de LdP extractive définis par ce Framework sont :

- **Décomposition** : cette activité responsable à la décomposition des variantes artefact à un ensemble d'éléments en utilisant un adaptateur comme montre la Figure 2.4. Cette couche est extensible et prends en charge différents types d'artefacts, L'utilisation de l'adaptateur spécialisé à un type d'artefact donnée fournit un représentation interne commune par les éléments.
- **Identification des blocs** : cette activité serte à identifier les blocs. Un bloc est un ensemble d'éléments obtenu par l'analyse et la comparaison des va-

riantes d'artefact. Les blocs permettent d'augmenter la granularité de l'analyse par les experts du domaine pour ne pas raisonner au niveau d'élément. L'identification des blocs représente une étape initiale avant le raisonnement au niveau des features, les technologies d'identification des blocs nous leur montrerons plus tard.

- ☐ **Identification et nommage des features** : cette activité consiste à analyser les blocs pour les mapper aux features par exemple, un bloc peut être fusionné avec un ou plusieurs autres blocs pour représenter une seule feature, ou diviser les éléments d'un bloc afin de l'ajuster aux features. Les noms des blocs peuvent être modifiés par les experts du domaine pour avoir des noms représentatifs plus significatifs.
- ☐ **Localisation des features** : cette activité vise à crée des correspondances entre les features et les blocs identifies par une valeur de confiance entre zéro et un. Par exemple, si la technique de localisation des features se termine qu'une feature donnée se trouve dans un bloc donné avec une confiance égale un, cela signifie que la technique est presque certaine que les éléments de ce bloc implémentent cette feature.
- ☐ **Découverte des contraintes** : cette activité permet de découvrir les contraintes structurelles entre les blocs qui compose les artefacts, dans l'exemple de variantes d'images, les éléments Pixel ne peuvent pas se chevaucher donc la technique utilisé va identité une contrainte d'exclusion mutuelle entre les blocs de même position.
- ☐ **Synthèse de modèles features** : les features et les contraintes identifiées sont utilisées pour créer automatiquement le modèle de features.
- ☐ **Construction d'assets réutilisables** : cette activité permet de crée des assets réutilisables à l'aide de l'adaptateur. Un assets réutilisable construit à partir d'un ensemble d'éléments selon les besoins, ces éléments peuvent correspondre à un bloc, à une feature identifiée. Dans l'exemple de variantes d'image, les assets réutilisables ont été construits correspondent aux images des blocs identifiés, comme illustré à la Figure 2.3.
- ☐ **Visualisation** : l'activité de visualisation est orthogonale à toutes les autres activités car elle est destinée à présenter à l'expert du domaine toute infor-

mation pertinente générée par une autre activité. Les visualisations peuvent être utilisées non seulement pour montrer, mais aussi pour interagir avec les résultats des différentes activités.

2.3 Implémentation de BUT4Reuse

BUT4Reuse est l'implémentation du Framework générique et extensible d'adaptation de LdP extractive présenté précédemment. La généricité du BUT4Reuse est assurer grâce à les adaptateurs, chacun de ce dernier trait un type d'artefact diffèrent. Actuellement, il intègre 15 adaptateurs mais la version publiée implémente 12 adaptateurs seulement sont présenter dans la Figure 2.5. L'extensibilité de BUT4Reuse est présentée par les différentes techniques intégrées dans chaque activité du Framework.

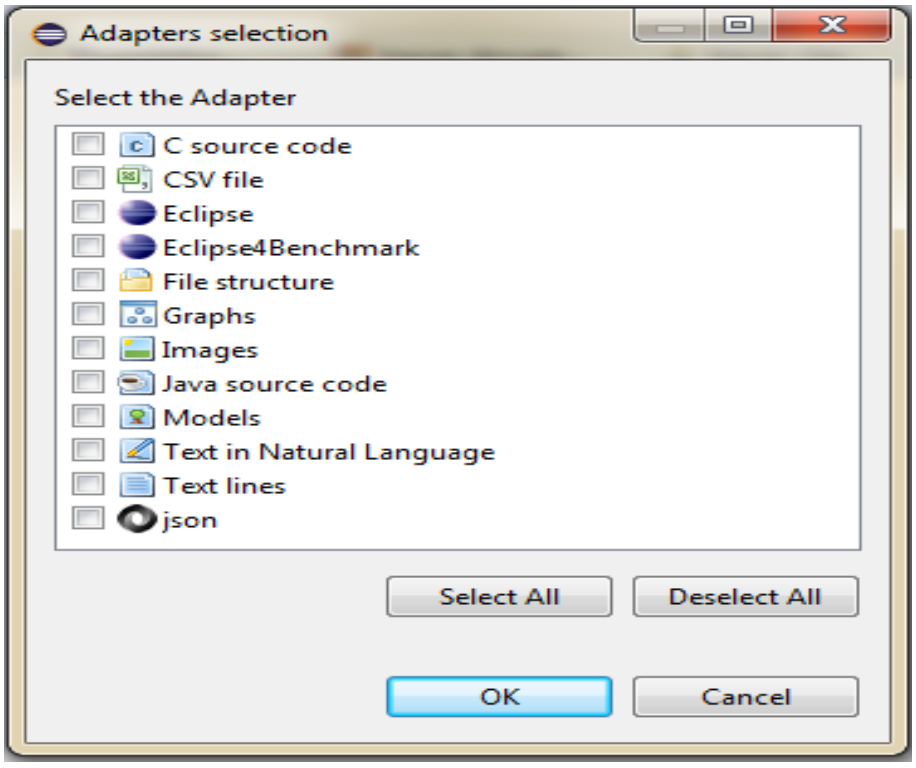


FIGURE 2.5 – La liste des adaptateurs intégré dans BUT4Reuse.

2.3.1 Identification de bloc

Les techniques d'identification des blocs intégrées dans BUT4Reuse partagent une phase d'analyse de similarité en utilisant la fonction de similarité identifié par l'adaptateur pour comparer les éléments, la Figure 2.6 montre les techniques utilisées et le résultat d'identification de blocs présenté dans le même Figure 2.6 avec le fond gris.

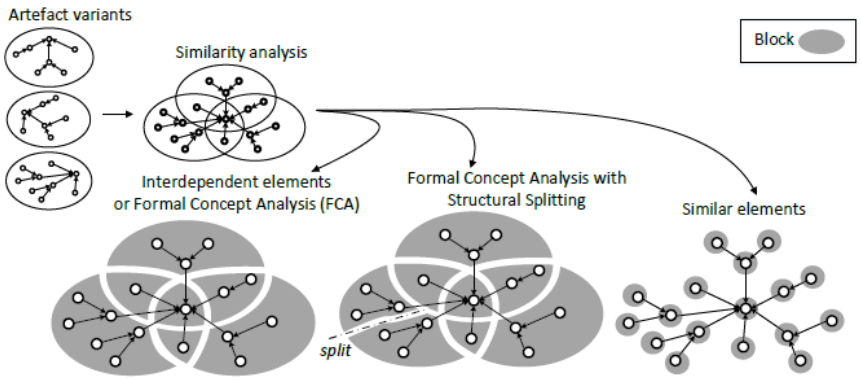


FIGURE 2.6 – Les techniques d'identification de bloc[37].

— **Interdependant elements** : l'algorithme d'éléments interdépendants proposé dans le travail [31] est basée sur un processus formel de définition d'un bloc définis comme suit : étant donné un ensemble A d'artefact que nous voulons comparer deux éléments (e1, e2) des artefacts d'A, e1 et e2 sont interdépendants si et seulement s'ils appartiennent exactement les mêmes artefacts de A. Par conséquent, e1 et e2 sont interdépendants si les deux conditions suivantes sont remplies :

$$\exists a \in A e_1 \in a \wedge e_2 \in a \tag{2.1}$$

$$\forall a \in A e_1 \in a \Longrightarrow e_2 \in a \tag{2.2}$$

L'idée derrière l'algorithme est séparer les différentes intersections entre les variantes d'artefact comme illustrer à gauche de la Figure 2.6, en obtenant un bloc pour chaque intersection des variantes d'artefact. Cette technique a utilisé dans l'exemple illustre les variantes d'image.

— **Formal Concept Analysis (FCA)** : Analyse de concept formel regroupe des éléments partageant des attributs communs. Dans BUT4Reuse, les en-

tités ou les objets du contexte formel sont les variantes d'artefacts et les attributs sont la présence ou l'absence de chacun d'éléments. Avec cette entrée, FCA découvre un ensemble de concepts contenant au moins un élément sont considérés comme un bloc. Au niveau technique, Le BUT4Reuse implémenté FCA pour l'identification de blocs à l'aide de Galatea. Les résultats expérimentaux confirment que les mêmes blocs sont obtenus en utilisant la technique des éléments interdépendants présentée précédemment et l'analyse de concept formel.

— F**CA with structural slpitting** : analyse formelle de concept avec fractionnement structurel permet d'étendre le FCA en divisant davantage le bloc obtenu sur la base des informations concernant les dépendances structurelles des éléments. Concrètement, il détecte des groupes d'éléments dans un bloc qui ne dépendent pas d'autres éléments du bloc. Dans la Figure 2.6 montre comment le bloc en bas à gauche est divisé en deux blocs parce que pas dépendance total entre les éléments qui compose le bloc, il y'a deux sous ensemble indépendante.

— **Similar elements** : l'algorithme d'éléments similaires fournit la granularité la plus fine dont chaque élément après la phase d'analyse de similarité correspond à un bloc comme montre la figure 6 chaque élément est associé à un bloc.

2.3.2 Identification des features

L'identification des features est un processus manuel uniquement actuellement, soutenu par les visualisations. Les experts du domaine analysent les éléments des blocs identifiés pour les mapper aux features. Dans le cas des blocs définit par l'algorithme d'éléments interdépendant chaque bloc directement associé à une feature comme monter dans d'images.

2.3.3 Localisation des features

La localisation des features consiste à mapper les blocs aux features. BUT4Reuse introduit la notion de confiance de feature localisée avec une valeur de zéro à un

expliquer précédemment. Le BUT4Reuse intégré les techniques de localisation des features suivantes :

— **Feature-Specific** : l'idée de cette technique est pour une feature donnée, les blocs sont ceux qui sont toujours présent lorsque la feature est présente, par la définition d'un pourcentage. Un pourcentage de 100% pour une paire donnée de bloc et feature signifie que le bloc apparaît toujours lorsque la feature est présente dans les artefacts implémente cette feature.

— **Strict Feature-Specific (SFS)** : SFS est plus restrictif que la technique Feature-Specific en suivant deux hypothèses : une feature est localisée dans un bloc lorsque le bloc apparaît toujours dans les artefacts implémentent cette feature et le bloc n'apparaît jamais dans les artefacts qui n'implémentent pas cette feature.

Le but4Reuse intégré autre technique tel : Latent Semantic Indexing (LSI), SFS and Shared term et SFS and Term frequency sont détailler dans le travail [37].

2.3.4 Découverte des contraintes

BUT4Reuse implémente plusieurs techniques de découverte des contraintes parmi les features ou les blocs qui peuvent être appliquées simultanément.

— **Découverte des contraintes structurelles** : elle consiste à analyser les dépendances structurelles entre des paires de features ou de blocs. Plus précisément, ils sont identifient les contraintes structurelles tel que :

$$requises(A \Longrightarrow B) \tag{2.3}$$

$$exclusionmutuelle](A \wedge B) \tag{2.4}$$

en analysant les dépendances structurelles existent entre les éléments de blocs ou de features.

— **Découverte des contraintes basées sur FCA** : cette technique a été présentée et évaluée dans le Framework de recherche d'adoption de LDP extractive [34, 35, 36]. Lors de l'application de FCA, les concepts sont reliés par des réseaux de concepts qui représentent des relations potentielles entre les blocs.

2.3.5 Synthèse de modèles features

La synthèse de modèles features est couverte par BUT4Reuse avec deux implémentations sont :

— **Diagramme de features à plat** : cette technique crée un modèle de features sans aucune information hiérarchique entre les features. Il est basé sur la création d'une feature racine abstraite dans laquelle toutes les features sont ajoutées en tant que sous- features, les contraintes sont ajoutées en tant que contraintes d'arbre croisé.

— **Alternatives avant Hiérarchie** : cette technique consiste à calculer d'abord les constructions alternatives à partir des contraintes d'exclusion mutuelles, puis à créer la hiérarchie à l'aide de le nécessite des contraintes. Les contraintes qui n'étaient pas incluses dans la hiérarchie sont ajoutées en tant que contraintes d'arbre croisé. Le résultat de cette technique dans le cas des exclusions mutuelles identifiées dans l'exemple d'images illustre dans la Figure 2.3. Les modèles de features synthétisés dans BUT4Reuse sont exportés vers FeatureIDE [41].

2.3.6 Construction des assets réutilisable

La construction des actifs réutilisable est la responsabilité de l'adaptateur qui utilisera les éléments associés aux features pour créer les actifs réutilisables. Une fois que les features sont localisées, BUT4Reuse a une fonctionnalité permettre de générer à nouveau les produits initiaux pour vérifier cette propriété.

2.3.7 Visualisation

BUT4Reuse fournit un ensemble de visualisations sont :

— **Visualisation de barres** : les barres permettent d'afficher plusieurs types d'informations. Concrètement, ça peut afficher comment les éléments sont répartis sur les artefacts, comment les blocs sont répartis sur le artefacts, comment les entités s'étendent dans les blocs et comment les blocs les cartographient. La Figure 2.7 présente un exemple de visualisation de barres de distributeurs automatique montrant comment les blocs sont répartis sur les

sites artefacts, sur le côté gauche, il y a une barre pour chacune des variantes, et sur le côté droit, il y a la liste des blocs identifiés. La hauteur de chaque barre est proportionnelle au nombre de éléments dans l'artefact et chaque bande est colorée en fonction du bloc associé.

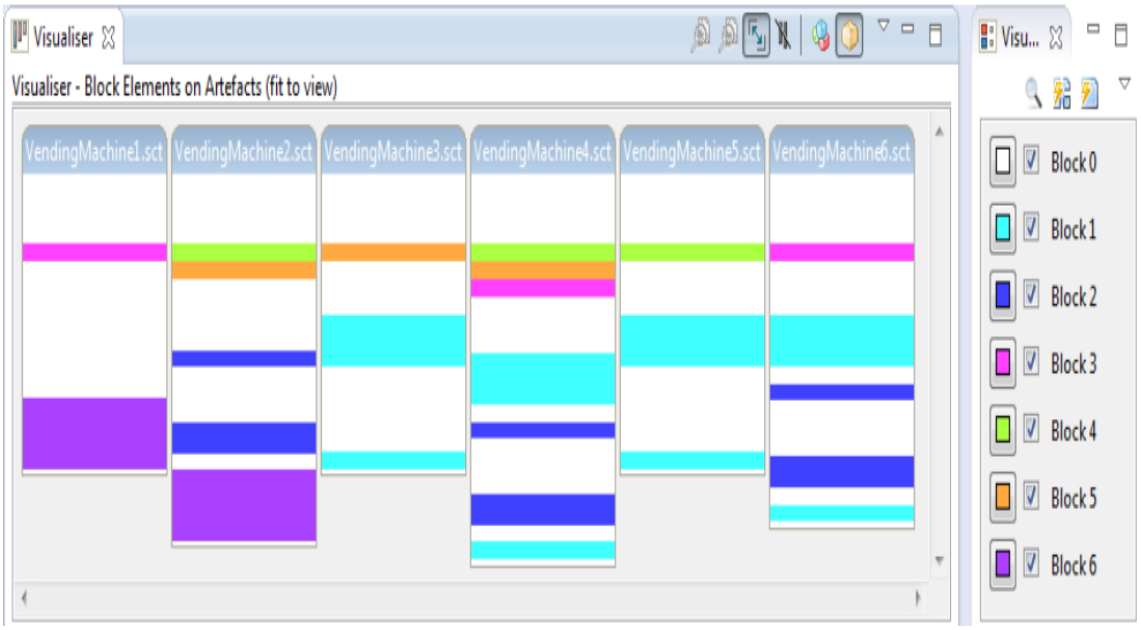


FIGURE 2.7 – Visualisation montrant les blocs (couleurs) sur les artefacts (barres) distributeurs automatique[37].

— **Carte thermique de la localisation de feature (feature location heat-map)** : la carte thermique permet de visualiser les relations entre les features et les blocs afin de faciliter leur localisation, les valeurs le plus importantes dans le matrice sont représentées par des carrés noirs et les valeurs le plus petites par des carrés plus clairs. La Figure 2.8 présente une matrice mettant en relation les features des artefacts distributeur automatique à des blocs identifiés sur cet ensemble d'artefacts. Les valeurs de matrice qui définissent leur couleur est basée sur le calcul de la technique de localisation de feature feature-specific. Par exemple, pour la feature Coffee, les blocs 0 et 4 ont 100% car ces blocs sont présents dans tous les artefacts où nous avons café. La carte thermique est enrichie de repères de localisation rouges pour afficher les résultats de la technique de feature sélectionnée qui peuvent être autres que

la feature spécifique. Dans ce cas, les marques rouges correspondent à SFS, ce qui signifie que, dans le cas du Coffee, le bloc 0 apparaît dans d'autres variantes sans café et le bloc 3 n'apparaît dans aucune variante n'a pas de Coffee. Par conséquent, la marque rouge apparaît uniquement pour le bloc 3.

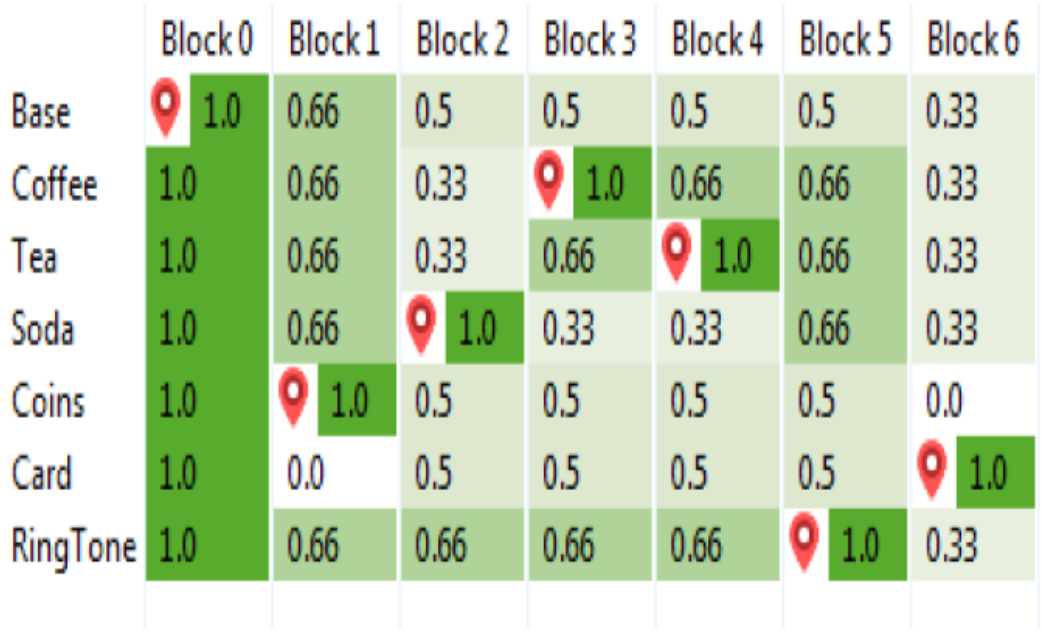


FIGURE 2.8 – Relation entre les blocs et les features concernant leur présence dans les variantes d'artefacts affichées en utilisant une visualisation de carte thermique [37].

— **Représentations graphiques** : BUT4Reuse fournit trois types de visualisations basées sur des graphiques sont :

1. Un graphe où les noeuds sont tous les éléments et les arêtes sont les relations de dépendance entre eux.

2. Un graphe où les noeuds sont les blocs identifiés et les arêtes sont les contraintes identifiées entre eux.

3. Un graphe où les noeuds sont les features que l'on veut localiser et les arêtes sont les contraintes identifiées entre elles. Dans tous ces graphes, les noms des nœuds et des arêtes sont différents pour faciliter la manipulation des graphes.

— **VariClouds** : Une visualisation exploite les nuages de mots, un paradigme

de visualisation largement utilisé pour les données textuelles [33], en prend en charge la dénomination des features lors de l'identification des features pour facilite la première rencontre entre les experts du domaine et les variantes afin de mieux comprendre la sémantique cachée dans les variantes ainsi que leur variabilité. La Figure 2.9 montre cette visualisation pour les distributeurs automatiques. Les noms des blocs peuvent être renommés manuellement par les experts de domaine.

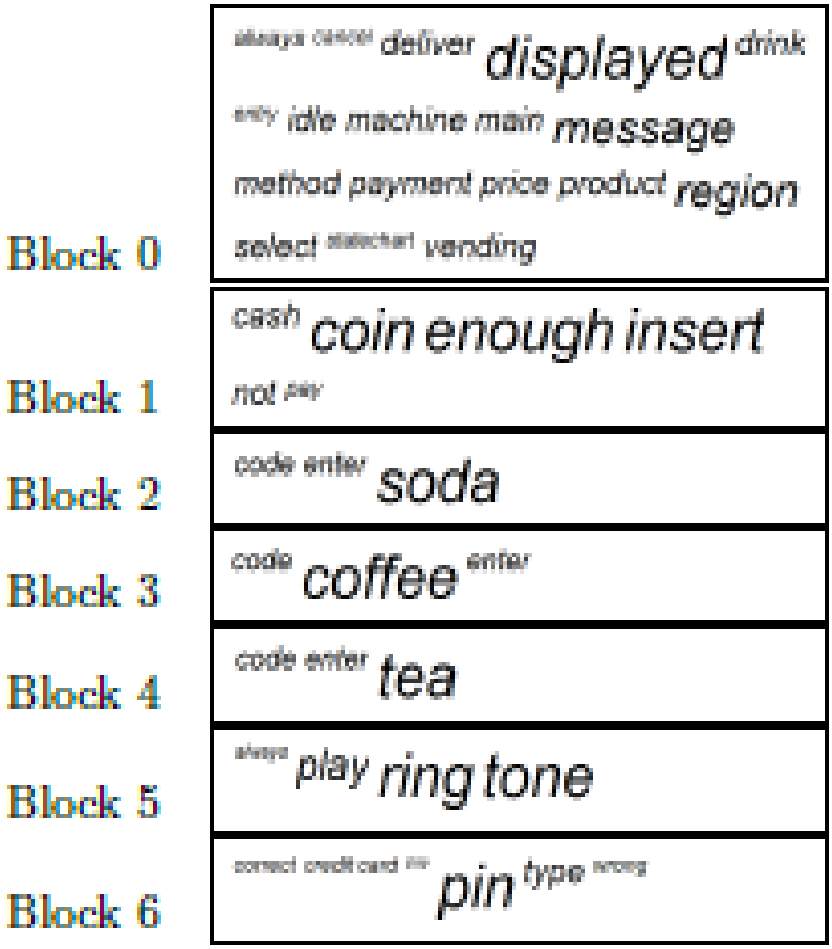


FIGURE 2.9 – Visualisations dans le nuage de mots des blocs identifiés dans variantes distributeurs automatique[26].

Le BUT4Reuse fournit deux autre visualisation Pruned Concept Hierarchy et Feature Relations Graphs (FRoGs) sont détailler dans le travail[26].

2.4 Conclusion

Dans ce chapitre, nous avons présenté le Framework BUT4Reuse, qui représente l'implémentation d'une approche générique et extensible pour l'adaptation de LdP extractive. La généricité est assurée grâce à le composant adaptateur et extensibilité par les différentes techniques et algorithmes concrets intégrer dans chaque activité du Framework d'adoption de LdP extractive.

Chapitre 3

Spécification OSGi et Eclipse

3.1 Introduction

Dans ce chapitre nous allons présenter la spécification OSGi, leur modèle de composants,ainsi que le framework OSGi. Par la suite, nous allons présenter Eclipse et Equinox comme une implémentation les plus populaires de OSGi.

3.2 Spécification OSGi

Open Service Gateway initiative(OSGi) est une spécification a été introduit en 1999 et soutenue par une quinzaine de sociétés parmi lesquelles apparaissent Erikson, SUN et IBM[42]. La spécification OSGi décrit un système de manière modulaire par un ensemble de composants et services écrite en langage Java, avec une gestion de dépendance entre ces composants[43].
Le domaine de déploiement du OSGi est très vaste, et couvre de multiples sections. Nous trouvons Nokia et Motorola qui mènent une technologie basée sur le standard OSGi pour les smart phone. De même, l'industrie automobile a adopté dans la partie essentielle de la spécification Global System for Telematics GST permettant aux véhicules de communiquer entre eux et avec le monde extérieur. L'implémentation de la spécification OSGi Equinox est l'environnement d'exécution d'Eclipse[44]. Apache Felix est le conteneur OSGi open source de Apache Software Foundation [45].

3.2.1 Framework OSGi

Le Framework OSGi forme le noyau de la spécification OSGi, leurs fonctionnalités sont divisés en des couches qui sont : (voir la Figure3.1)

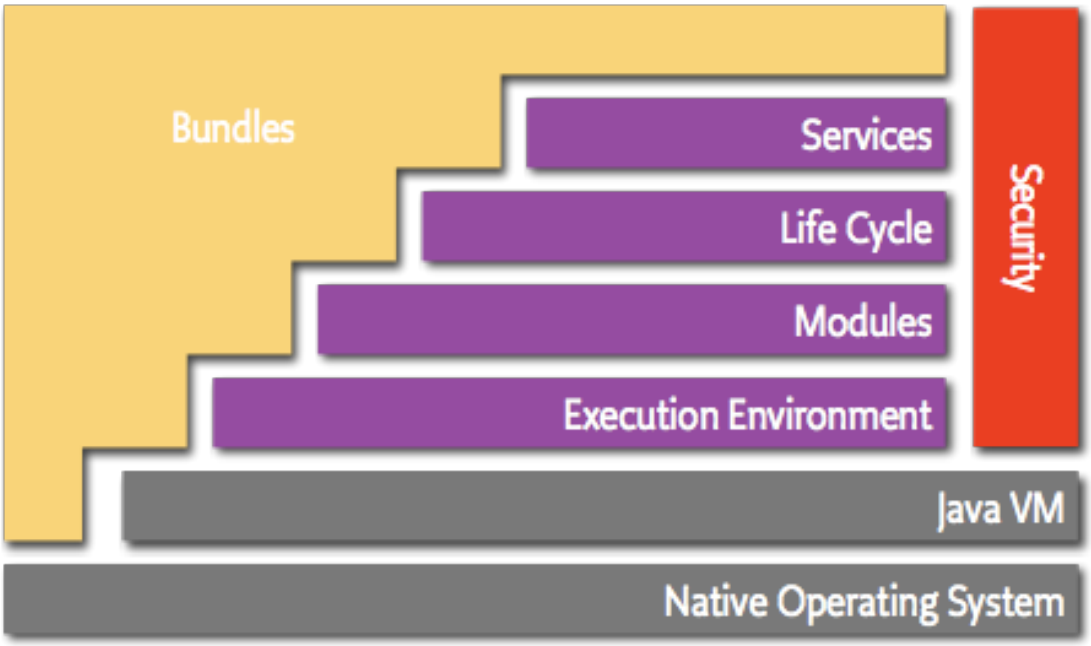


FIGURE 3.1 – Les couches du Framework OSGi[42].

— **Bundles** : Les bundles sont les composants qui composent les applications.
— **Services** : La couche services connecte les bundles de manière dynamique via Service Reristry qui est un API gère les services fournis par des bundles (enregistrement des services, maintien des références, ...).
— **Life Cycle** : La couche de cycle de vie fournit une API grée le cycle de vie d'un bundle. Cette API fournit un modèle d'exécution pour les bundles. Il définit comment les bundles sont démarrés et arrêtés, ainsi que leur mode d'installation, de mise à jour et désinstallé.
— **Modules** : Les modules sont les parties qui permettant la mise en oeuvre des bundles dans le Framework. Ils sont responsables de la gestion des dépendances entre les bundles et du classe loader.
— **Security** : La couche de sécurité gère les aspects de sécurité. cette couche fournit authentification du code, et signe les fichiers JAR numériquement[47].

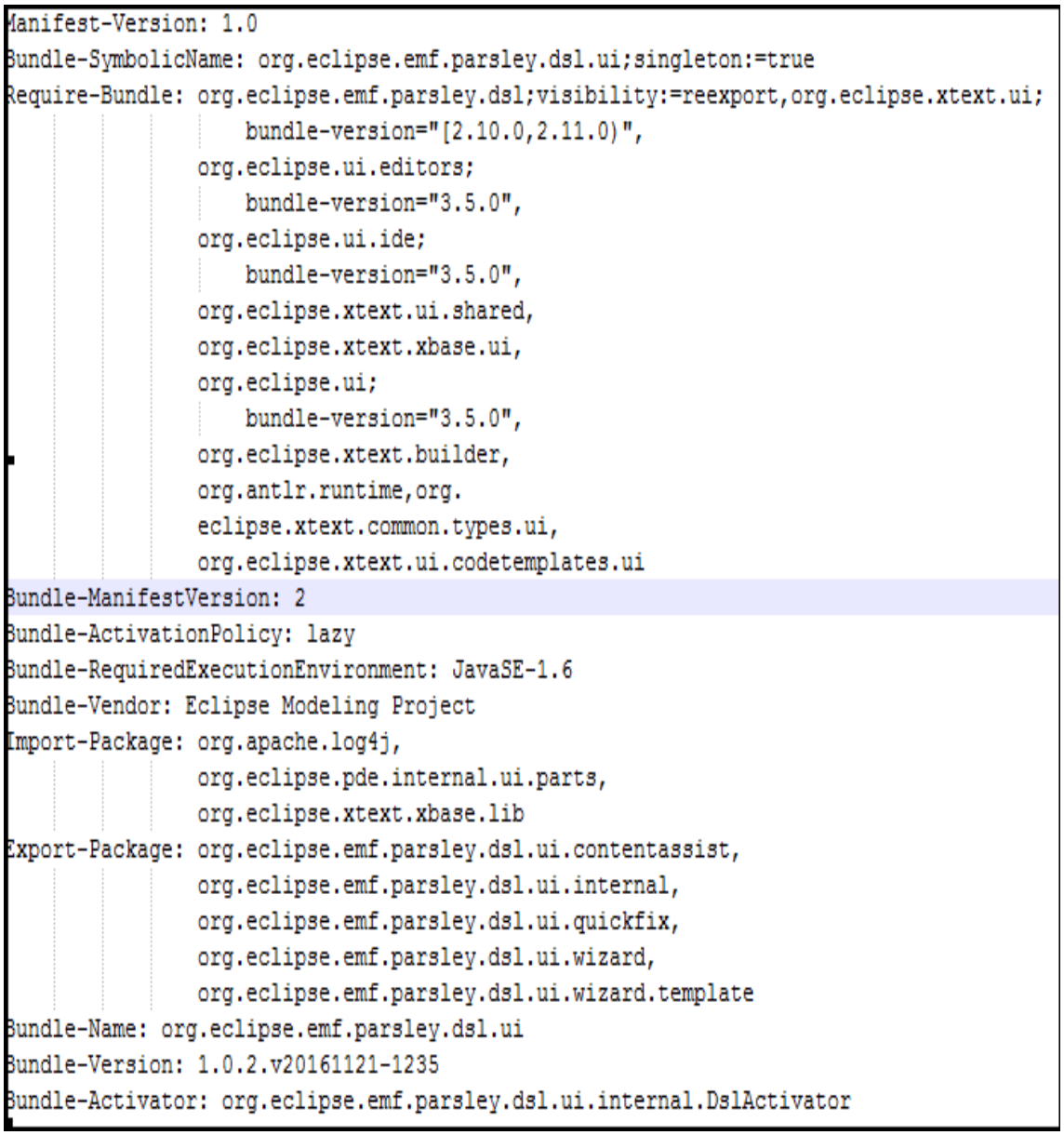
— **Execution Environment** : Définit les méthodes et les classes disponibles sur une plate-forme spécifique.

3.2.2 Composant OSGi

Un composant OSGi appelé bundle ou un Plugin dans la terminologie Eclipse. Il est déployé en tant que fichier Java ARchive (JAR) contenant le code binaire des classes, des ressources comme des fichiers de la configuration ou des images, et le fichier manifeste qui décrit comment le Framework va utiliser le bundle[49, 47]. Il constitue de méta-données tel que le nom du classe d'activation du bundle et d'autre information comme le nom, le nom symbolique, les packages importes et les packages exportés.

Méta-données d'un bundle

Un bundle contient des informations descriptives sur lui-même dans le fichier manifeste contenu sous le nom META-INF/MANIFEST.MF. La Figure 3.2 illustre un extrait du fichier MINIFEST.MT du bundle org.eclipse.emf.parsley.dsl.ui.



```
Manifest-Version: 1.0
Bundle-SymbolicName: org.eclipse.emf.parsley.dsl.ui;singleton:=true
Require-Bundle: org.eclipse.emf.parsley.dsl;visibility:=reexport,org.eclipse.xtext.ui;
                    bundle-version="[2.10.0,2.11.0)",
                org.eclipse.ui.editors;
                    bundle-version="3.5.0",
                org.eclipse.ui.ide;
                    bundle-version="3.5.0",
                org.eclipse.xtext.ui.shared,
                org.eclipse.xtext.xbase.ui,
                org.eclipse.ui;
                    bundle-version="3.5.0",
                org.eclipse.xtext.builder,
                org.antlr.runtime,org.
                eclipse.xtext.common.types.ui,
                org.eclipse.xtext.ui.codetemplates.ui
Bundle-ManifestVersion: 2
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-Vendor: Eclipse Modeling Project
Import-Package: org.apache.log4j,
                org.eclipse.pde.internal.ui.parts,
                org.eclipse.xtext.xbase.lib
Export-Package: org.eclipse.emf.parsley.dsl.ui.contentassist,
                org.eclipse.emf.parsley.dsl.ui.internal,
                org.eclipse.emf.parsley.dsl.ui.quickfix,
                org.eclipse.emf.parsley.dsl.ui.wizard,
                org.eclipse.emf.parsley.dsl.ui.wizard.template
Bundle-Name: org.eclipse.emf.parsley.dsl.ui
Bundle-Version: 1.0.2.v20161121-1235
Bundle-Activator: org.eclipse.emf.parsley.dsl.ui.internal.DslActivator
```

FIGURE 3.2 – Fichier MANIFEST.MF.

Les différente parties de ce fichier sont :

— **Bundle-Name** : définit un nom lisible par l'homme pour ce bundle(dans la Figure 3.2 Bundle-Name : org.eclipse.emf.parsley.dsl.ui).

— **Bundle-SymbolicName** : le seule obligatoire pour qu'un jar soit un bundle OSGi, il s'agit d'un nom unique du bundle, car il permet au framework de différencier les bundles (dans la Figure 3.2 Bundle-SymbolicName : org.eclipse.emf .parsley.dsl.ui).

L'attribut singleton : indique que le bundle ne peut avoir qu'une seule version résolue dans un environnement. La valeur true indique que le bundle est un singleton. La valeur par défaut est false. Framework doit résoudre au plus un bundle lorsque plusieurs versions d'un bundle singleton portant le même nom symbolique sont installées. Les bundles singleton n'affectent pas la résolution des bundles non singleton portant le même nom symbolique.

— **Bundle-ManifestVersion** : indique la spécification OSGi à utiliser pour lire ce bundle (dans la Figure 3.2 la version égale 2).

— **Bundle-version** : indique la version de ce bundle (dans la Figure 3.2 la version 1.0.2.v20161121-1235).

— **Bundle-Activator** : permet de spécifier le nom de la classe exécutable dans le bundle (dans la Figure 3.2 org.eclipse.emf.parsley.dsl.ui.internal.DslActivator est la nom du la classe Activator) .Cette classe doit implémenter l'interface BundleActivator et ses méthodes start et stop qui seront appelées lors du démarrage (méthode start) et dee l'arrêt (méthode stop) du bundle .

— **Export-Package** : (interfaces fournis de bundle) indique quels packages Java contenus dans le bundle seront mis à la disposition des autres bundles (dans la Figure 3.2 les packages exportés par ce bundle sont :
rg.eclipse.emf.parsley.dsl.ui.contentassist,
org.eclipse.emf.parsley.dsl.ui.internal,
org.eclipse.emf.parsley.dsl.ui.quickfix,
org.eclipse.emf.parsley.dsl.ui.wizard,
org.eclipse.emf.parsley.dsl.ui.wizsard.template).
Les directives suivantes peuvent être utilisées dans l'en-tête Export-Package
  — version : spécifié la version de package nommés.
  — x-internal : utilisez cette option pour marquer un package exporté comme contenant des détails d'implémentation internes qui ne sont pas considérés comme des API.
  — x-friends : utilisez cette option pour marquer un package exporté comme accessible uniquement à une liste d'ensembles nommés.

— **Import-Package** : (interfaces requises)indique les packages Java requis de ce bundle pour répondre à ces dépendances requises. On peut spécifier la

version des packages exportés en utilisant l'attribut version (dans la Figure 3.2 les packages importés par ce bundle sont : org.apache.log4j, org.eclipse.pde.internal.ui.parts, org.eclipse.xtext.xbase.lib).

— **Require-Bundle** : la liste des bundles requis, en indiquant leur Symbolic-Name (dans la Figure 3.2 l'une des bundles requis par le bundle org.eclipse.emf.parsley.dsl.ui est org.eclipse.ui.editors). Les directives suivantes peuvent être utilisées dans l'en-tête Require-Bundle [47] :
  — visibility : si la valeur est private (valeur par défaut), tous les packages visibles des bundles requis ne sont pas réexportés. Si la valeur est reexport, les bundles qui ont besoin de ce bundle seront transitoirement avoir accès aux packages exportés de ces bundles requis. Autrement dit, si le bundle A nécessite le bundle B et que le bundle B nécessite le bundle C avec visibilty : = reexport, le bundle A aura accès à tous Les packages exportés du paquet C comme si le paquet A avait le bundle C requis.
  — resolution : si la valeur est mandatory (par défaut), le bundle requis doit exister pour que ce bundle puisse être résolu. Si la valeur est optionel , le bundle sera résolu même si le bundle requis n'existe pas.
  — bundle-version : la valeur de cet attribut est une plage de versions permettant de sélectionner la version de bundle du bundle requis.
— **Bundle-RequiredExecutionEnvironment (BREE)** : Spécifiez la version de Java requise pour exécuter le bundle. Si cette condition n'est pas remplie, le runtime OSGi ne charge pas le bundle.
— **Bundle-ActivationPolicy** : permettre à un bundle de spécifier une stratégie de démarrage. La seule stratégie défini est "Lazy". cette dernier indique que ce bundle ne doit être activé que si l'un de ses composants, à savoir les classes et les interfaces, est utilisé par d'autres bundles. S'il n'a pas défini le runtime d'Equinox n'active pas le bundle.
— **Bundle-ClassPath** : spécifie où charger les classes du bundle. Par défaut est '.' qui permet aux classes d'être chargées à partir de la racine du bundle. Vous pouvez également y ajouter des fichiers JAR. Ces fichiers sont appelés fichiers JAR imbriqués [43].

— **Fragment-Host** : permettent d'ajouter des fonctionnalités facultatives à un autre bundle appelé Host, au moment de l'exécution un fragment et fusionné avec son host et a une visibilité complète sur tous les package de son host[46].
Fragment : un fragment ressemble au bundle à la différence que leur contenu est fusionné de manière transparente au moment d'exécution avec son host bundle plutôt d'être autonome[46].

— **Bundle-Localization** : contient l'emplacement du paquet où les fichiers de localisation peuvent être trouvés. La valeur par défaut de Bundle-Localization est "OSGI-INF/l10n/bundle"[47]

Cycle de vie d'un bundle

Un bundle suit le cycle de vie représenté par le diagramme d'état de la figure 3.3.

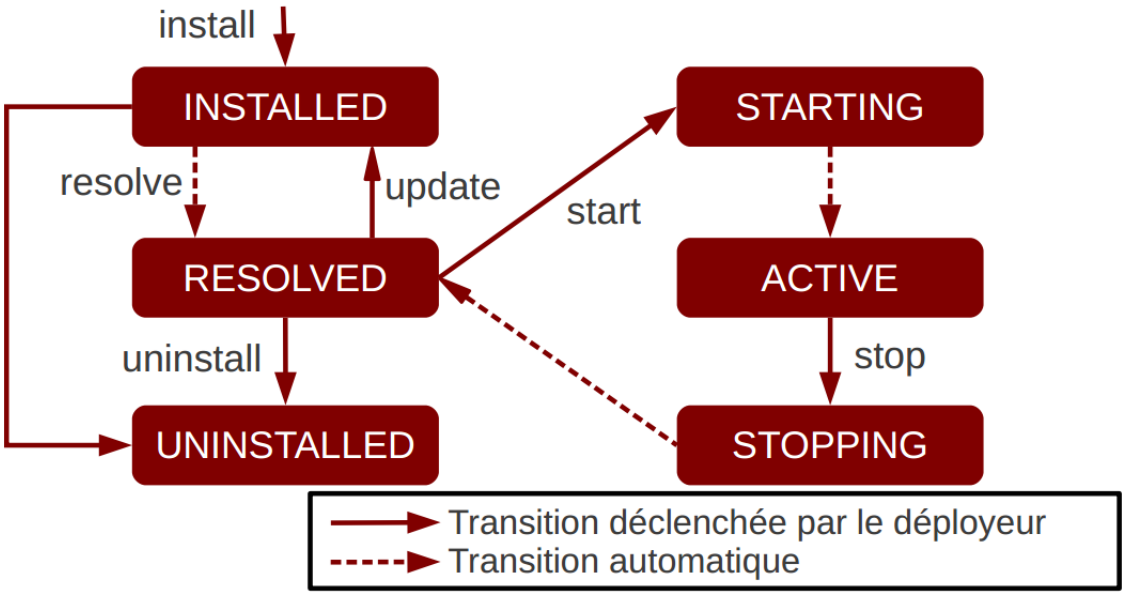


FIGURE 3.3 – Cycle de vie d'un bundle OSGi [50].

Les états de cycle de vie sont : INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING, UNINSTALLED. Le bundle change son état automatiquement ou par le développeur manuellement . La transition entre chaque état est marquée par la diffusion d'un BundleEvent à des BundleListeners enregistrés. L'événement identifie

la transition ou le type et le bundle concerné [46].

- **INSTALLED** : le bundle est installé dans le moteur d'exécution OSGi avec succès, n'a pas encore de chargeur de classe et leur service et leurs contributions au registre d'extensions ne sont pas encore traités.
- **RESOLVED** : Toutes les classes Java dont le bundle a besoin sont disponibles. Cet état indique que le bundle est soit prêt à être démarré ou s'est arrêté.
- **STARTING** : est l'état temporaire dans lequel est le bundle tant que l'exécution de la méthode start n'est pas encore exécutée.
- **ACTIVE** : lors de l'exécution de la méthode de démarrage "start" de la classe Activator le bundle soit dans l'état actif. Il peut rechercher des services, en enregistrer et démarrer des threads.
- **STOPPING** : un bundle actif qui revient à l'état arrêt si la méthode "stop" de la classe Activator est exécutée. Cette méthode doit désenregistrer les services fournis, relâcher les services utilises et arrêter les threads démarres.
- **UNINSTALLED** : Le bundle a été désinstallé, il ne peut pas passer dans un autre état.

Les bundles présent dans le système peuvent se comporter de manière inattendue en cas des exceptions[46].

3.2.3 Services OSGi

La spécification OSGi définit un service comme étant : "*A service is a normal Java object that is registered under one or more Java interfaces with the service registry. Bundles can register services, search for them, or receive notifications when their registration state changes*"[49].

Un service OSGi est défini par une interface Java, et une classe d'implémentation de cette dernière, les services sont enregistrés dans le Service Registry pour qu'ils puissent utiliser ou bien consommer par d'autre bundles. Un service peut y avoir plusieurs implémentations publiées simultanément pour la même interface, les services dans OSGi sont dynamiques et peuvent enregistrer et dés-enregistrer à l'exécution [48].

Évolution des services OSGi

Les services font partie intégrante de Framework OSGi et existent depuis le début. Au fil du temps, les installations autour des services ont évolué : [46]

Version 1 : **enregistrement de service et écoute d'événement** : la version initiale de la spécification OSGi prévoyait l'enregistrement des services et des API d'écoute comme moyen de travail avec le modèle de service OSGi. Cela nécessite généralement que le développeur de bundle crée et enregistre des services dans l'activateur de bundle. Un bundle nécessite un autre service ajoute un écouteur d'événements de service et répond aux événements d'enregistrement, de dés-enregistrement et de modification de service déclenchés par la structure OSGi.

Version 2 : Service Tracker : la version 2 de la spécification OSGi a introduit le Service Tracker. Comme son nom l'indique, Service Tracker suit l'ajout, la suppression et la modification de services utilisés par un bundle et élimine une grande partie du code d'auditeur en double et des conditions de concurrence.

Version 4 : **Services déclaratifs** : la version 4 de la spécification OSGi introduit les services déclaratifs, qui facilitent la gestion de l'acquisition dynamique et de la publication des services pour les développeurs de bundles en éliminant tout code de gestion de services. De plus, l'utilisation des services déclaratifs facilite l'utilisation des services OSGi en retardant le chargement des classes et la création d'objets.

Exemple de service : enregistrement de service et écoute d'évènement

La Figure 3.4 montre l'assemblage des composants OSGi pour montre le fonctionnement des services selon le principe de "publish-find-bind" :

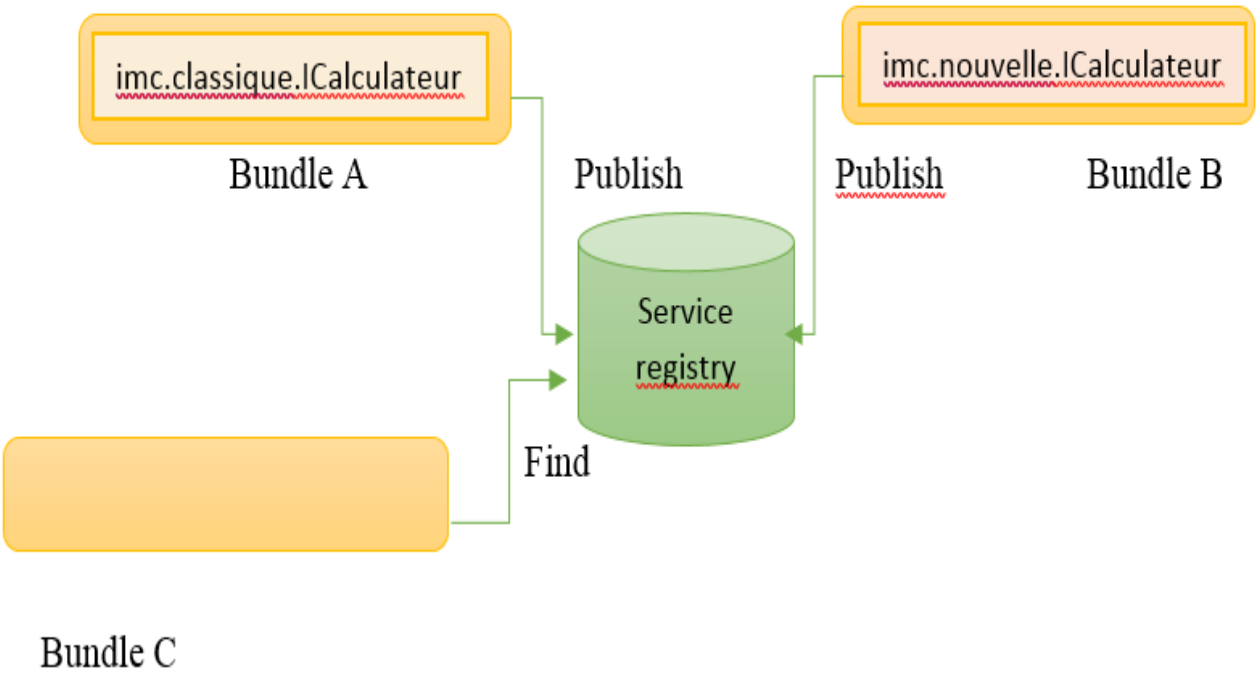


FIGURE 3.4 – Fonctionnement des services OSGi.

1. Le bundle fournisseur (dans ce cas le bundle **A**) du service **imc.classique.ICalculateur**, il va publier son contrat qui est une interface Java et un objet de la classe d'implémentation de l'interface auprès du **service registry**.
2. Le **service registry** va créer et maintenir une référence pour le service **imc.classique.ICalculateur**.
3. Le bundle **C** consommateur(dans ce cas la bundle C) va rechercher la référence de service **imc.classique.ICalculateur**, par la suite le service registry va retourner (ou non) la référence du ce service pour que le bunndle **C** puisse utiliser les fonctions de bundle **A**.

Structure d'un bundle fournis un service

Le bundle A qui fournit un service nommé **ICalculateur.Imc.Classique** doit :

1. Implémenter une interface Java **ICalculateur.Imc.Classique** (voir le code 3.1), qui représente le contrat du service.

code source 3.1 – Interface imc.classique.ICalculateur

```
package imc.classique;
public Calculateur
{
    String calculateur(double poinds,double taille);
}
```

2. Une classe implémente l'interface ICalculateur voir le code 3.2

code source 3.2 – Classe imc.classique.ICalculateu

```
package imc.classique.impl;
import imc.classique.ICalculateur;
public Calculateur implements ICalculateur
{
   String calculateur(double poinds,double taille)
    {
    double imc=poids/taille^{2}
    if (imc< 18.5)
      return(Sous-poids:Poids trop faible, r serves
       insuffisantes en cas de maladie);
    else if (imc>18.5 && imc<24.9)
      return(Poids normal:Poids sant  essayez d'y r
       ester !);
    else if(imc>25.0 && imc<29.9)
       return(Surpoids:Il vous est conseill  de mai
       -grir pour atteindre un poids sant );
    else if(imc>30.0 && imc<39.9)
       return(Ob sit :Tr s forte probabilit  d'app
       -arition de diab te, de maladies cardiovascu
      -laires, d'infertilit  (chez les femmes) et
        de douleurs articulaires.);
    else if(imc> 40)
       return(Ob sit  morbide:Il vous est urgent
       de perdre du poids : votre sant  est en g
```

```
        -rand danger!);
      }
  }
```

3. Une classe Activator qui implémente l'interface BundleActivator ou fournis par le Framework OSGi, pour l'enregistrement de service dans le Service Registry avec la méthode start et le dés-enregistrement dans la méthode stop, le code montre l'enregistrement de service imc.classique.ICalculateur.

code source 3.3 – Classe Activator

```
  import org.osgi.framework.BundleActivator ;
  import org.osgi.framework.BundleContext ;
  import imc.classique.ICalculateur;
  import imc.classique.impl;
  public class Activator implements BundleActivator
  {
   public void start (BundleContext context)
   throws Exception
    {
     ICalculateur service = new Calculateur();
//enregistrer le service aupr s de Service Registry
  context.registerService(
     //l'interface que le service impl mente
   ICalculateur.class.getName(),
     //l'instance de service
   service,
      //les propri tes ou null
   null
  );
 System.out.println("ICalculateur est enregistrer");
      }
  }
```

4. Dans le fichier manifest du bundle fournisseur de service on doit déclarer que le bundle A fournis le package imc.classique.ICalculateur pour que les autre bundles peuvent utiliser ce service via l'entête Export-Package : imc.classique.ICalculateur.

Structure d'un bundle consomme un service

Un bundle peut rechercher et utiliser un service provenant d'un autre bundle. Dans ce cas le buncle C consomme le service imc.classique.ICalculateur qu'est fournis par le bundle A. Le bundle C doit déclarer dans leur fichier manifest Require-Bundle :A ou Import-Package :imc.classique.ICalculateur

Il est recommandé d'utiliser Import-Package plutôt que Require-Bundle car cela permet de créer un système plus souple et moins couplé, offrant aux concepteurs de systèmes la possibilité d'échanger les implémentations et les déploiements de fonctions en fonction de leurs besoins.

code source 3.4 – Classe Activator consommateur

```
Package imc.classique.cosommateur;
import org.osgi.framework.*;
import imc.classique.ICalculateur;
public class Activator implements BundleActivator
{
public void start(BundleContext ctx) throws Exception
{
//L'obtention de la r f rence de service ICalculateur
ServiceReference ref = ctx.getServiceReference(ICal
 -culateur.class.getName());
// test si le service est enregistrer dans le Service

Registry
if(ref!=null)
{
//l'obtention d'objet de service ICalculateur
ICalculateur s = (ICalculateur)ctx.getService(ref);
```

```
  /*test si le service a   t   d senregistr  depuis l'acqu
    -isition de ServiceReference*/
        if (s !=null)
         {
          System.out.println(s.calculateur(60,1.60));
         }
   }
}
```

3.3 Eclipse

Eclipse est une plate-forme d'outils universels un environnement de développement intégré (IDE) ouvert et extensible pour tout. Eclipse est l'une des initiatives les plus excitantes nées du monde du développement d'applications depuis longtemps et bénéficie du soutien considérable des plus grandes entreprises et organisations dans le secteur de la technologie. Eclipse est de plus en plus répandu dans les domaines commercial et universitaire [51, 44].

La Fondation Eclipse fournit à notre communauté mondiale d'individus et d'organisations un environnement mature, évolutif et propice aux échanges commerciaux pour la collaboration et l'innovation en matière de logiciels open source.

3.3.1 Architecture d'Eclipse

Eclipse utilise des plugins pour fournir toutes les fonctionnalités à l'intérieur et au-dessus du système d'exécution. Son système d'exécution est basé sur Equinox, une implémentation de la spécification du OSGi [52]. Les plugins peuvent être regroupés en features [53].

3.3.2 Versions d'Eclipse

Eclipse fournit des versions (*Releases*) régulières. Auparavant, il s'agissait d'une version importante par an publié en juin, suivie par des versions simultanée (*simul-*

Année de publication	Version	Nom de version	Nombre de distributions
2019	4.10	2018-12M1	14
2018	4.9	2018-09M1	14
2018	4.8	Photon R	14
2017	4.7	Oxygen	11
2016	4.6	Neon 3	14
2015	4.5	Mars 2	12
2014	4.4	Luna SR2	14
2013	4.3	Kepler SR2	14

TABLE 3.1 – Les versions et leurs noms, année de publication et le nombre de distribution.

taneous release) à des fins de maintenance. Chaque version est nommée par nom de planète, par exemple Luna pour version 4.4 et Oxygen pour la version 4.7, mais en 2018 la cadence est passée d'une version annuelle par une version tous les trois moins à partir de version nommé 2018-9 [53]. Le tableau 3.1 montre les versions Eclipse, son nom, l'année de publication et le nombre de distributions.

3.3.3 Distributions d'Eclipse

Eclipse Project Packaging (ERP), le projet de packaging Eclipse (EPP) fournit des distributions d'Eclipse (packages) ciblant différents profils de développeur, à mesure que le projet Eclipse évolue, de nouveaux packages apparaissent et certains autres disparaissent en fonction des intérêts et des besoins de la communauté [53] Les distributions fournies par le EPP pour la version Kepler sont [54] :

1. **Eclipse Modeling Tools**

   Cette distribution fournit une cadre et des outils permettant d'exploiter des modèles : un modeleur graphique Ecore (diagramme de type classe), un utilitaire de génération de code Java pour les applications RCP et EMF Framework, une prise en charge de la comparaison de modèles, une prise en charge des schémas XSD, OCL et des modélisateurs graphiques. Il comprend un SDK complet, des outils de développement et du code source.

2. **Eclipse IDE for Automotive Software Developers**

   Cette distribution contient des Framework et des outils utilisés pour le développement de logiciels automobiles embarqués, outre la plate-forme Eclipse, les outils de développement Java et l'environnement de développement de plug-in, il comprend les formats EMF, GMF, Xtext, Xpand, UML2, certains autres outils de développement C/ C ++, éditeurs et outils XML/ XSD.

3. **Eclipse IDE for Java EE Developers**

   Une distribution pour les développeurs Java créant des applications Java EE et Web, y compris un IDE Java, des outils pour Java EE, JPA, JSF, Mylyn, EGit, intégration Maven, développement JavaScript, editeurs et outis XML et d'autre.

4. **Eclipse IDE for Java Developers**

   Cette distribution contient les outils essentiels pour tout développeur Java, y compris un IDE Java, un client CVS, un client Git, un éditeur XML, Mylyn, l'intégration Maven, WindowBuilder, intégration Maven pour Eclipse, éditeurs et outils XML.

5. **Eclipse IDE for Testers**

   Cette distribution contient des fonctionnalités Eclipse prenant en charge le

processus d'assurance qualité du développement logiciel, telles que Jubula et Mylyn.

6. **Eclipse Standard**

   Cette distribution contient la plate-forme Eclipse et tous les outils nécessaires à son développement et à son débogage : outils de développement pour Java et plugin, support de Git et CVS, y compris la documentation source et celle du développeur.

7. **Eclipse IDE for C / C ++ Developers**

   Un environnent de développent intégrer pour les développeurs C / C ++ avec intégration Mylyn.

8. **Eclipse IDE Java and DSL developers**

   Cette distribution contient les outils essentiels pour les développeurs Java et DSL, notamment un IDE Java, Xtend, un framework DSL (Xtext), un client Git, un éditeur XML et l'intégration Maven.

9. **Eclipse for Parallel Application Developers**

   Cette distribution contient outils pour C, C ++, Fortran et UPC, y compris MPI, OpenMP, OpenACC, un débogueur parallèle et la création, l'exécution et la surveillance à distance d ?applications éditeurs et outils XML.

10. **Eclipse for RCP and RAP Developers**

    Ce package contient un ensemble complet d'outils pour les développeurs souhaitant créer des plug-ins Eclipse, des applications client enrichi ou une plate-forme d'application distante (RCP + RAP), ainsi que Mylyn, l'intégration m2e Maven et un éditeur XML. Outre le fournisseur CVS Eclipse Team, il contient également l'outil EGit permettant d'accéder aux systèmes de contrôle de version Git, WindowBuilder, éditeurs et outils XML.

11. **Eclipse Scouts Developers**

    Ce package permet de développer des applications métier basées sur Java Eclipse qui s'exécutent sur le bureau, dans les navigateurs et sur les appareils mobiles. Ce package inclut un SDK complet, des outils de développement utiles et du code source, environnement de développement de plugin Eclipse.

12. **Eclipse IDE for Java and Report Developers**

Ce package contient les outils Java EE et l'outil de création de rapports BIRT permettant aux développeurs Java de créer des applications Java EE et Web ayant également des besoins en matière de création de rapports, outils de développement JavaScript et éditeurs et outils XML.

3.3.4 Comment choisir une distribution d'Eclipse

Eclipse fournis plusieurs distributions, chacune d'elles est dédiée à un domaine de programmation spécifique. Le choix d'une distribution d'Eclipse se fait selon le besoin du programmeur. Il est donc important d'avoir une idée avant d'entamer à télécharger l'application Eclipse, car cela va déterminer la distribution dont on a besoin, et ce qui est inutile dans notre contexte applicatif. Par exemple si il y'a quelqu'un est intéressé par la modélisation, il doit sélectionner la distribution dédiée à la modélisation *Eclipse Modeling Tools*.

3.3.5 Installation et organisation d'une distribution d'Eclipse

La liste des distributions est disponible sur la page http ://www.eclipse.org/downloads/ , nous devons choisir et télécharger la distribution qui nous paraisse intéressante . Le fichier téléchargé est une simple archive au format ZIP, que nous l'ouvrons avec n'importe quel utilitaire adéquat. Une fois cette archive extraite, cette distribution a une structure de répertoires suivante Figure 3.5

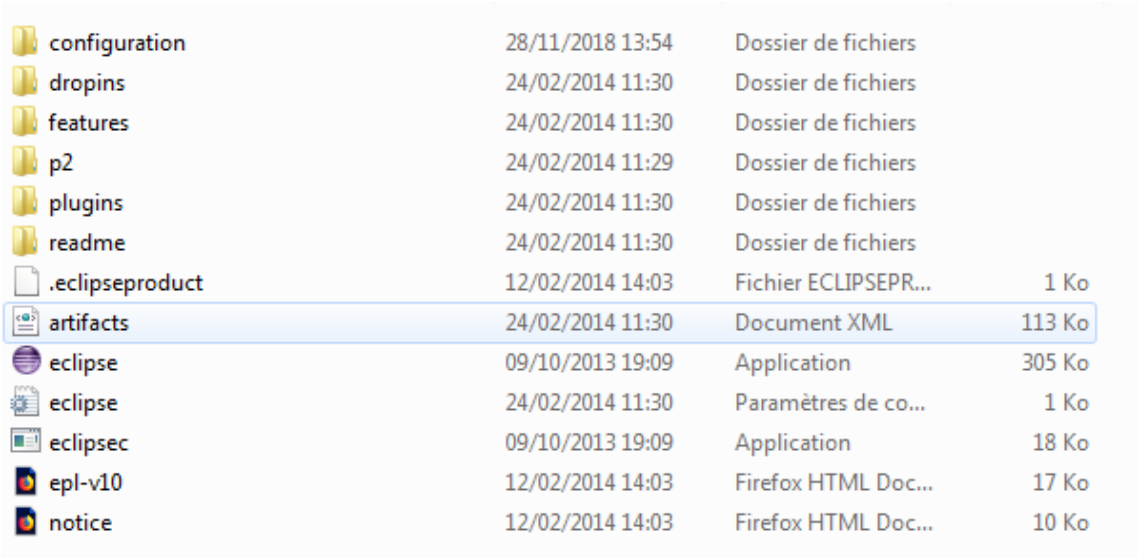


FIGURE 3.5 – Organisation d'une distribution d'Eclipse.

— **configuration** : un répertoire contient un fichier de configuration nommé config.ini sera lit au démarrage de moteur d'exécution. Ce dernier est un fichier de propriétés standard pouvant configurer de nombreux aspects du moteur d'exécution[55]. Le répertoire configuration/org.eclipse.equinox.simpleconfigurator contient fichier bundels.info qui contient une liste de tous les plug-ins installés dans le système actuel [56], dont chaque ligne représente un bundle par les informations suivantes : symbolicName, version du bundle, un chemin relatif ou absolu path du fichier jar, un chiffre indique le niveau de démarrage du bundle et false/ true pour indiquant si le bundle doit être démarré ou non au démarrage d'Eclipse.

— **dropins** : un répertoire initialement vide, mais par la suite peut contenir des plugins installés manuellement. Le processus typique consiste à télécharger un plugin et à extraire cette archive dans le répertoire dropins.

— **features** : un répertoire regroupe les features du distribution d'Eclipse. Les features ne contiennent aucun code. Ils décrivent simplement un ensemble de plug-ins qui fournissent les fonctionnalités de la feature et des informations sur la façon de la mettre à jour. Les feautres sont décrit à l'aide d'un fichier manifeste de feature, feature.xml.

— **p2** : Equinox p2 est une composante du projet Equinox. p2 fournit une plate-forme de provisioning pour les applications basées sur Eclipse. Ici, dans le wiki, vous pouvez trouver des informations sur la conception et le développement en cours (par exemple, procédures, plans, etc.). Pour tous les articles de wiki liés à Equinox, voir l'index des articles d'Equinox. Pour les articles spécifiques à p2, voir l'Index des articles d'Equinox p2.

  p2 stocke dans les fichiers de profil la liste des états d'installation précédents. Chaque changement d'installation ajoute une nouvelle entrée. Cela signifie un nouveau fichier .profile ou .profile.gz.

— **plugins** : un répertoire contient tous les plugins de eclipse soit en format Jar ou un répertoire. Par la suite, Eclipse stockera tous les plugins que vous installez en utilisant "Help" -> "install new software".

— **readme** : un répertoire contient le document html qui comporte des notes de publication du projet Eclipse comme la version, les environnements d'exploitation cibles, la compatibilité avec les versions précédentes, les problèmes connus, le lancement d'Eclipse, la mise à niveau d'un espace de travail à partir d'une version précédente et l'interopérabilité avec les versions précédentes

— **.eclipseproduct**

— **artifacts**

— **eclipse** : c'est le fichier exécutable d'eclipse.

— **eclipse** : eclipse.ini est un fichier texte contenant des options de ligne de commande qui sont ajoutées à la ligne de commande utilisée lors du démarrage d'Eclipse. Il existe de nombreuses options disponibles. L'une des options les plus recommandées consiste à spécifier une machine virtuelle Java sur laquelle Eclipse doit s'exécuter [57]

— **eclipsec**

— **epl-v10** : un document html décrit la licence public d'Eclipse.

— **notice** : un document html décrit le contrat d'utilisation du logiciel Eclipse.

3.3.6 Comment installer des composants dans Eclipse

Eclipse permet à ces utilisateurs d'ajouter les plugins qui manquent dans leur distribution choisi. Par exemple si un utilisateur installer la distribution Eclipse Modeling Tools et veut travailler avec Android, il doit suivre les étapes suivantes pour ajouter Android Development Tools (ADT), un environnement de développement Eclipse permettant de développer des applications pour le système Android.

1. Démarrez Eclipse, puis sélectionnez **Help** > **Install New Software**.
2. Cliquez sur **Add** dans le coin supérieur droit.
3. Dans la boîte de dialogue Add Repository qui apparaît, entrez le nom et l'URL de composant qui tu a besoin.

   Dans ce cas on veut développer des applications Android sous Eclipse, en doit installer ADT. Dans ce cas en enter "ADT Plugin" pour le nom et l'URL suivante pour l'emplacement : comme montre la Figure 3.6

https ://dl-ssl.google.com/android/eclipse/

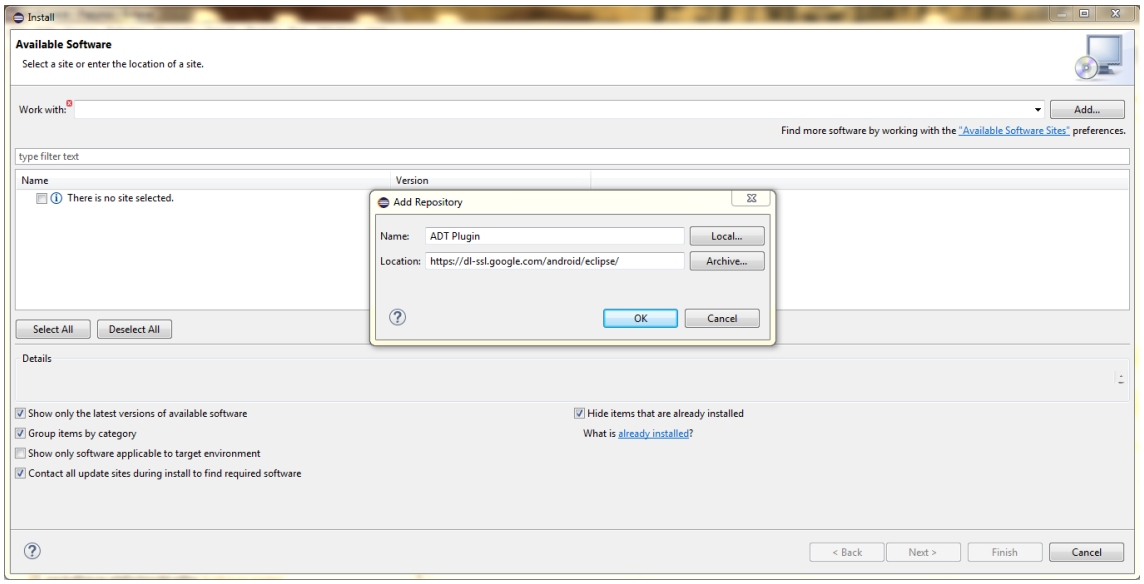


FIGURE 3.6 – Installation de ADT sous Eclipse.

Si vous ne parvenez pas à obtenir le plug-in, essayez d'utiliser "http" dans l'URL d'emplacement, au lieu de "https" (https est préférable pour des rai-

sons de sécurité).

4. Cliquez sur **OK**.
5. Dans la boîte de dialogue Logiciels disponibles, cochez la case en regard de Outils de développement, puis cliquez sur **Next**.
6. Dans la fenêtre suivante, vous verrez une liste des outils à télécharger. Cliquez sur **Next**. Lisez et acceptez les contrats de licence, puis cliquez sur **Finish**.
7. Si vous recevez un avertissement de sécurité indiquant que l'authenticité ou la validité du logiciel ne peut pas être établie, cliquez sur **OK**.
8. Une fois l'installation terminée, redémarrez Eclipse.

3.4 Conclusion

Dans ce chapitre, nous avons présenté la spécification OSGi.C'est une plateforme de services fondé sur le langage Java. Elle permet d'étendre les capacités de modularité de Java en introduisant le concept de "bundle" et "service". En outre, nous avons présenté Eclipse. Il utilise Equinox une implémentation de la spécification OSGi comme un environnement d'exécution. A la fin de ce chapitre nous avons présenté les produits ou les distributions Eclipse.

Dans le chapitre suivant nous allons voir le processus proposé et les détails pour réaliser un adaptateur pour les distributions d'Eclipse.

Chapitre 4

Développement d'adaptateur pour les distributions d'Eclipse

4.1 Introduction

Dans les chapitres précédents, nous avons expliqué l'approche d'extraction de ligne de produit logiciels et son implémentation dans le Framework BUT4Reuse et Eclipse .

Dans ce chapitre, nous allons définir un processus générique pour l'extraction d'une ligne de produit logiciels à partir des variantes d'applications et leurs traces d'exécutions. Nous allons appliquer ce processus pour les distributions d'Eclipse. En suite, nous allons présenter les détails de notre adaptateur pour ce type d'artefacts. L'objectif est l'extraction d'une LdP et la dérivation d'un produit Eclipse personnalisé et minimal.

4.2 Processus proposé

La Figure 4.1 présente le processus proposé pour l'extraction d'une LdP à partir des variantes applications. Ce processus comporte les étapes suivants :

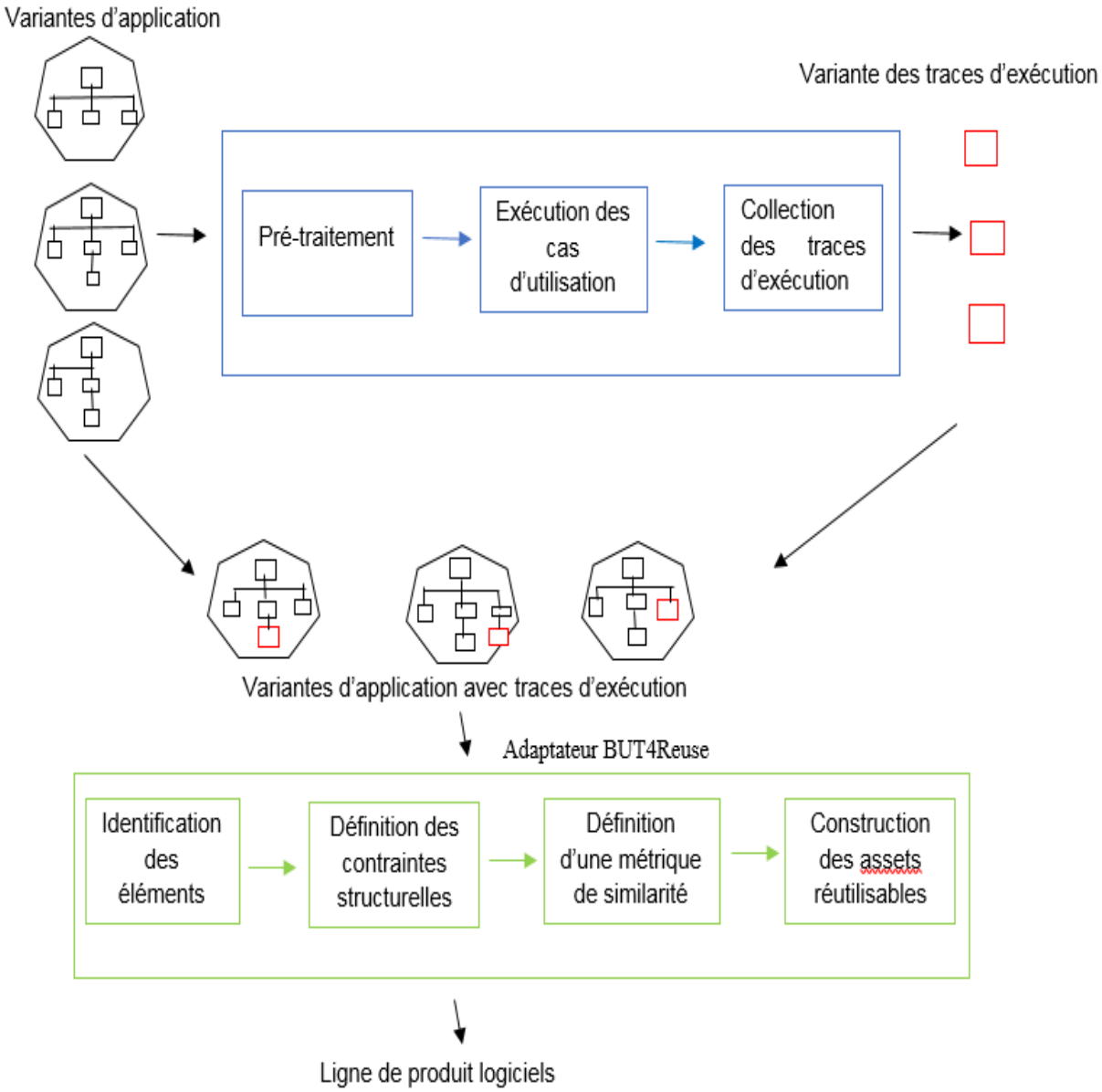


FIGURE 4.1 – Processus Proposé

1. Faire un pré-traitement sur les variantes d'application (ces variantes d'application telles que, les distributions d'Eclipse, les codes sources, ...). Le pré-traitement de chaque application peut présenter l'installation ou (et) développement des composants pour l'analyse dynamique de ces artefacts pour obtenir les traces d'exécution.

2. L'exécution des cas d'utilisation sur chaque variante. Un cas d'utilisation est une séquence d'actions dont un acteur interagit avec le système et exécute

ses fonctionnalités. Par exemple, pour une distribution d'Eclipse IDE for Java EE Developers, on peut distinguer les cas d'utilisation suivants : l'utilisateur après le lancement de l'application, il peut créer un nouveau projet Java, ajouter des classes, des interfaces, sauvegarder les changements, compiler et exécuter un programme...

3. La collection des traces d'exécution sont enregistrés dans un fichier généré par notre outil.
4. Associer le fichier de trace d'exécution à sa variante correspondante.
5. Utiliser l'adaptateur BUT4Reuse pour les variantes en entrée (distributions Eclipse pour notre cas). L'adaptateur a comme entrée les variantes avec les traces d'exécutions obtenues dans les étapes précédentes.

4.3 Application de processus sur les distributions d'Eclipse

Nous allons appliquer le processus présenté précédemment pour l'extraction de ligne de produits logiciels sur les distributions d'Eclipse pour la génération d'une distribution minimale et personnalisée d'Eclipse.

4.3.1 Méta modèle pour les traces d'exécution

Tout d'abord, nous avons défini un méta modèle pour les traces d'exécutions. Il est présenté dans la Figure 4.2. Il est composé de quatre méta-classes dont la classe principale est **Traces**.

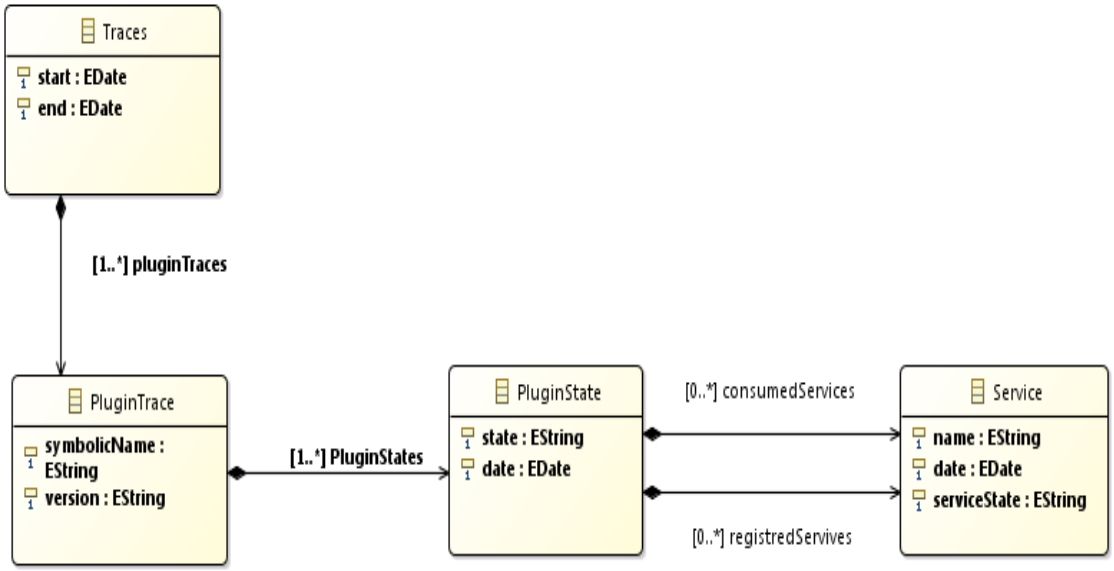


FIGURE 4.2 – Méta modèle de traces d'exécution.

Traces : Une instance de cette classe représente le modèle de trace d'exécution des cas d'utilisation pour une variante d'eclipse. Elle se compose d'un ensemble d'éléments de type *PluginTrace*. Les attributs de cette dernière sont *start* et *end*. Ils représentent respectivement le temps de début et fin de l'exécution d'une variante Eclipse.

PluginTrace : c'est une méta classe qui représente les traces d'exécution d'un seul plugin. L'attribut *symbolicName* prend le symbolicName de ce plugin et l'attribut *version* prend sa version. Chaque PluginTrace est composé d'un ou plusieurs PluginState qui représente les états d'un plugin. Les états possibles d'un plugin sont installed, resolved, uninstalled, starting, active, stopping. L'attribut *date* représente la date du changement de l'état. Chaque pluginState a zéro à plusieurs *consumedServices* qui sont les services consommés et zéro à plusieurs *registredServices* qui représentent les services fournis par ce plugin dans cet état.

Service : c'est une méta classe qui représente un service. Elle est caractérisé par l'attribut *name* (le nom de service), *serviceState* (c'est l'état de ce service) et *date* (c'est la date d'enregistrement de ce service).

4.3.2 Méta modèle EMF et la génération de code source

Le méta modèle EMF consiste en deux parties : les fichiers de description Ecore et Genmodel. Le fichier ecore contient les informations de classes définies. Le fichier Genmodel contient des informations supplémentaires pour la génération de code, par exemple les informations de chemin et de fichier. Le fichier Genmodel contient également le paramètre de contrôle comment le code doit être généré.

I. La création de méta modèle Ecore

- Dans notre plugin, cliquer droite pour créer un répertoire nommé "model".
- Cliquer droit sur le répertoire model : new -> other-> Ecore Model-> next.
- Nommer ce modèle 'traces.ecore', puis sélectionner EPackage et cliquer finish.
- Sélectionner le package du modèle, et donner les propriétés (Name,URI,Prefix).
- Créer les éléments de notre méta modèle tel que EClass (Traces, PluginTrace, PluginState, Service), ERefrence (pluginTraces, PluginStates, registredServices, consomedServices), EAttribute (symbolicName, date,...).

  La Figure 4.3 représente les éléments de notre modèle Ecore.

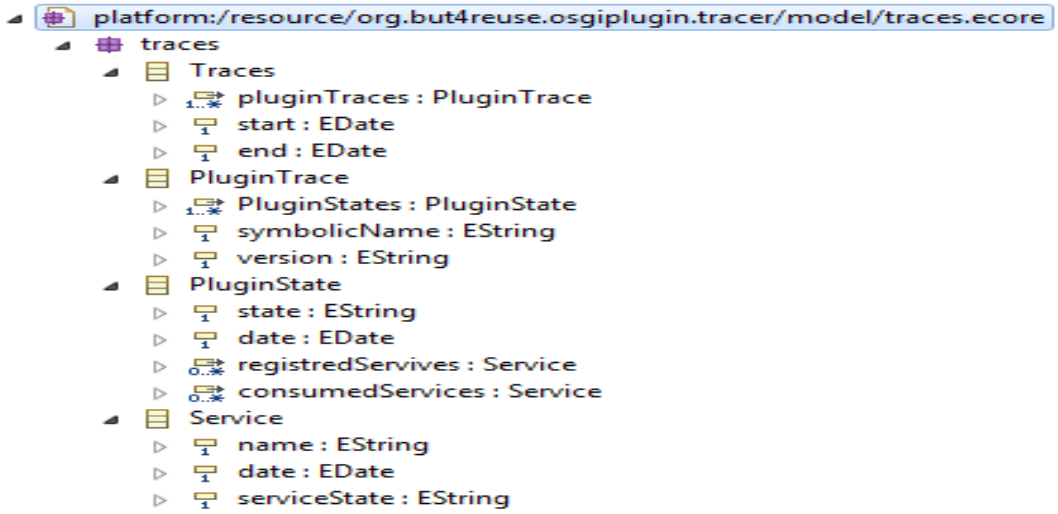


FIGURE 4.3 – Modèle Ecore traces.ecore

Exemple de création des éléments :

☐ Pour la création d'une EClass PluginTrace, cliquer droite sur le package traces -> New child ->Eclass. Dans les propriétés de cette Eclass : saisir le nom

PluginTrace dans la case name.

- ☐ Pour la création de l'Ettribute symbolicName de la Eclass PluginTrace : cliquer droite sur Eclass PluginTrace -> New child -> Ettribute. Dans les propriétés de cet Ettribute :
  - — Saisir son nom SymbolicName dans la case Name.
  - — Saisir le type Estring dans la case Etype.
  - — Donner son cardinalité (1) : le min (1) dans la case Lower Bound et le max (1) dans la case Upper Bound.
- ☐ Pour la création d ?un EReference PluginTraces de la classe Traces : cliquer à droite sur cette classe -> New child -> EReference .Dans les propriétés de cet EReference :
  - — Saisir le nom PluginTraces dans la case Name.
  - — Sélectionner le type PluginTrace dans la case Etype.
  - — Donner son cardinalité (1...*) : le min (0) dans la case Lower Bound et le max (-1) dans la case Upper Bound.
  - — Sélectionner True dans la case containment, pour avoir une relation de type composition.

II. Création de Genmodel et génération de code Java Sur la base de fichier Genmodel, nous pouvons générer du code Java. En cliquant avec le bouton droit sur le noeud racine traces du fichier traces.genmodel et sélectionné Generate Model Code traces comme la Figure 4.4 Cela crée l'implémentation Java du modèle EMF dans notre model Ecore.

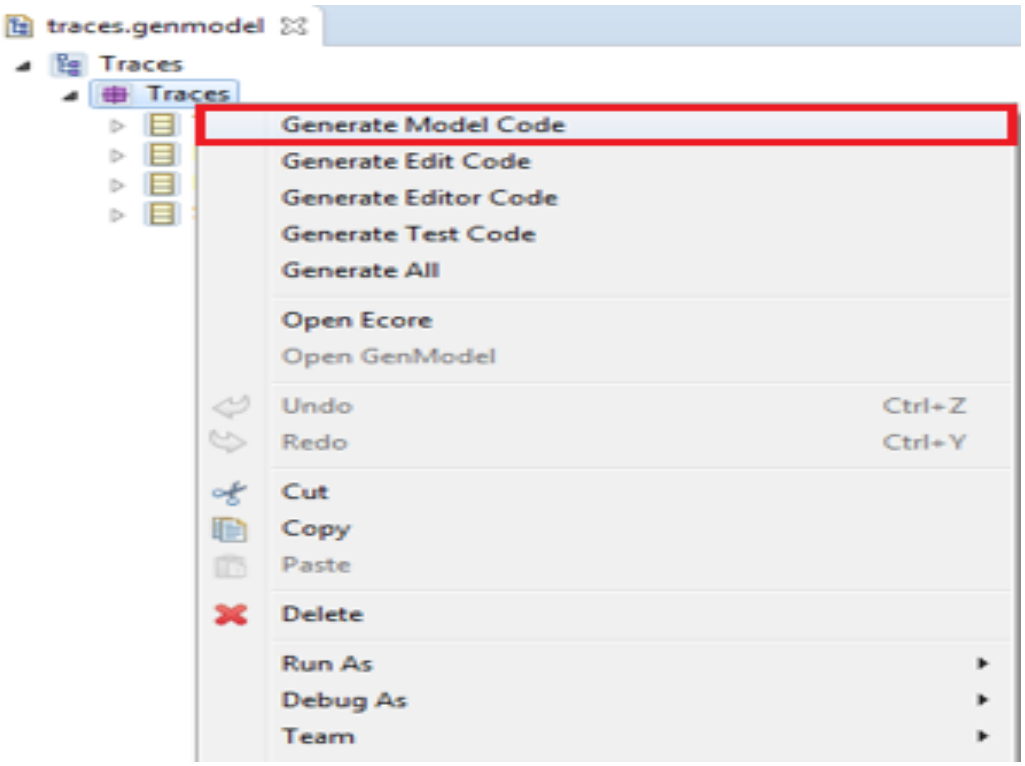


FIGURE 4.4 – Génération de code de méta-modèle traces.

III. Exécution de notre composant

Pour exécuter notre composant *osgi.but4reuse.osgiplugin.tracer* qui est responsable de sauvegarder des traces d'exécution, nous devons configurer une exécution comme elle montrée dans la Figure 4.5.

Tout d'abord nous devons sélectionner notre plugin *osgi.but4reuse.osgiplugin.tracer* et donner une valeur inférieure à 4 dans la propriété Start Level et true pour Auto-Start afin de démarrer automatiquement notre plugin et en premier temps. Cela permet de récupérer tous les évènements. Ensuite, cliquer sur Apply pour sauvegarder cette configuration et puis sur Run pour l'exécution.

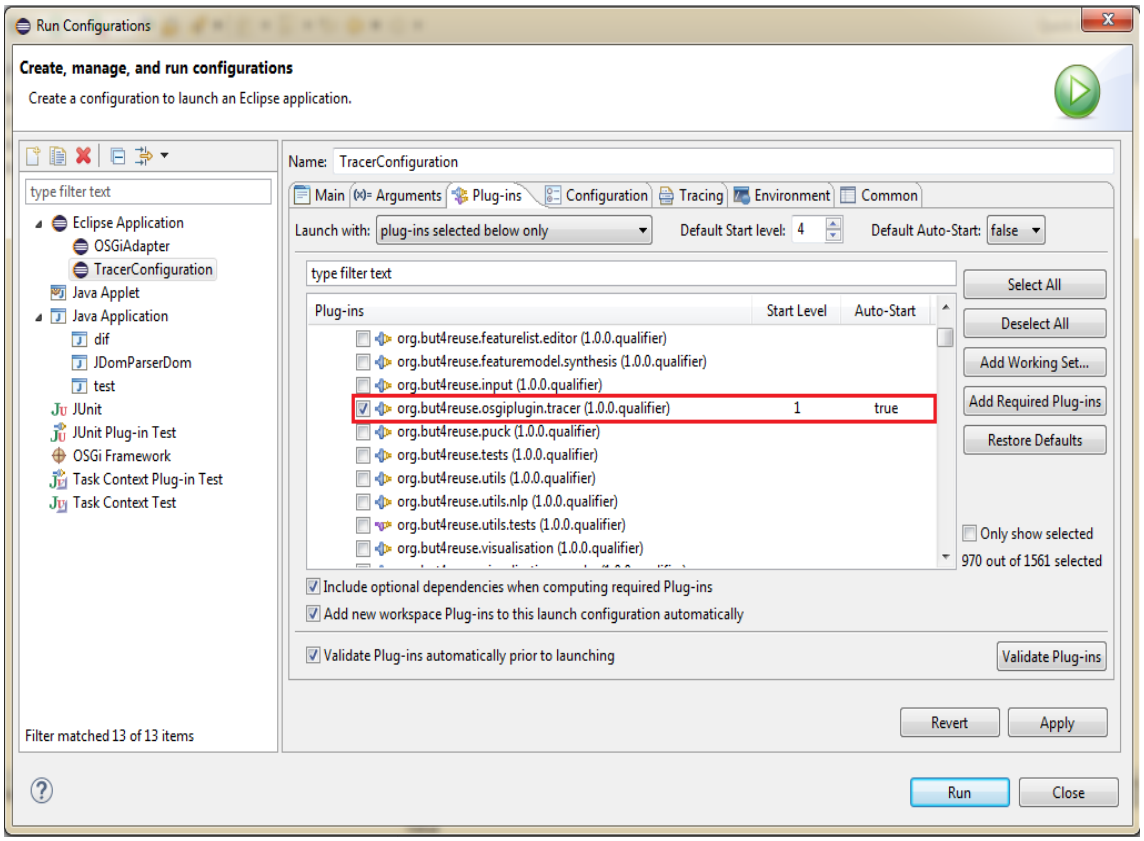


FIGURE 4.5 – Configuration d'exécution pour notre composant tracer.

4.4 Développement d'adaptateur pour les distributions d'Eclipse

Pour la réalisation de notre adaptateur des distributions Eclipse dans le Framework BUT4Reuse, nous avons défini une extension du plugin *org.but4reuse.adapters* en suivant les étapes de conception d'adaptateur (chapitre 2).

4.4.1 Définition d'extension de *org.but4reuse.adapters*

L'extension est un mécanisme proposé par Eclipse permettant de configurer et/ou étendre un plugin qui définit le point d'extension. La déclaration des points d'extension et points d'extension se trouvent dans le fichier *plugin.xml*[58].

La définition d'un point d'extension par un plugin nécessite la définition d'un contrat qui se présente sous forme d'un schéma XML qui contient la structure des

méta données qui sont fournis par les extensions, avec un identifiant de point d'extension et un nom lisible par l'homme.

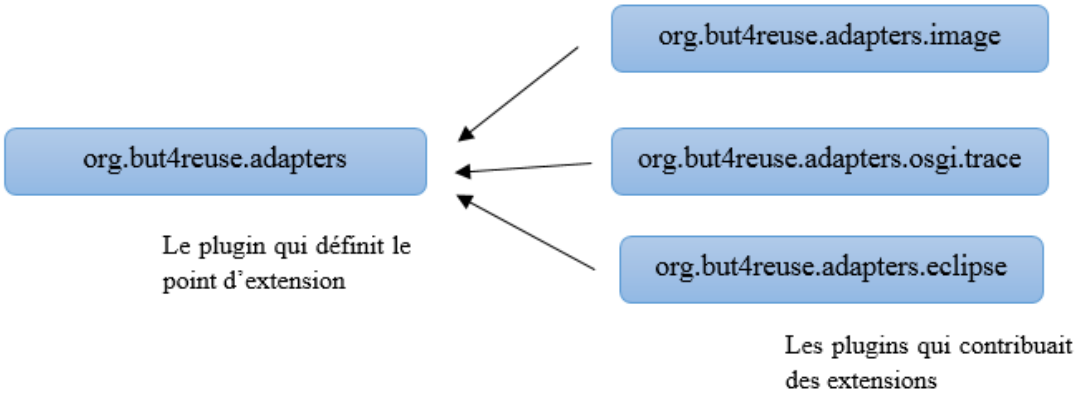


FIGURE 4.6 – Relation entre point d'extension et des extensions[58]

Le But4Reuse permet de développer des nouveaux adaptateurs pour différents types d'artefacts de manière flexible grâce à leur plugin *org.but4reuse.adapters* qui définit un point d'extension que les développeurs peuvent étendre pour produire leurs propore adaptateurs. La Figure 4.6 montre la relation entre ce plugin et les plugins qui font des extensions.

Nous avons défini une extension de *org.but4reuse.adapters* pour réaliser notre adaptateur. Nous avons nommé le plugin d'extension *org.but4reuse.adapters.osgi.trace* en respectant le schéma XML en définissant la structure de ce point d'extension comme nous l'avons montré dans la Figure 4.7.

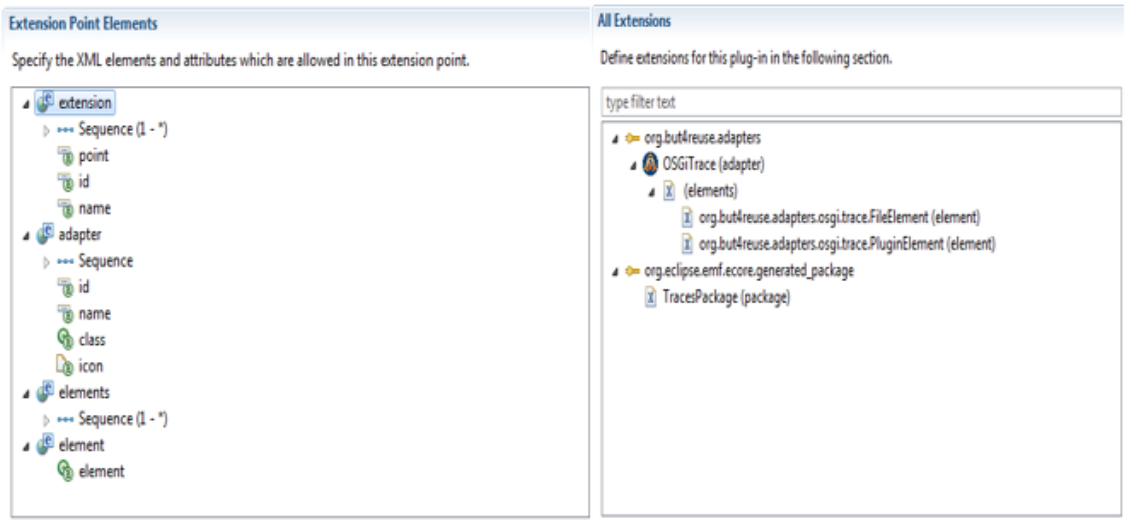


FIGURE 4.7 – Structure de point d'extension org.but4reuse.adapters et l'extension org.but4reuse.adapters.osgi.trace

Dans la réalisation de notre adaptateur d'Eclipse, nous avons suivi les activités expliquées précédemment dans le chapitre 2 pour la conception d'un adaptateur pour un type d'artefacts. Avant de présenter les détails d'implémentation de notre adaptateur, nous allons présenter un extrait de l'architecture de BUT4Reuse. Figure 4.8 montre les dépendances de type Require_Bundle entre les 50 % de plugins qui représentent le coeur de BUT4Reuse. Nous remarquons que le plugin **org.but4reuse.adapters.osgi.trace** a requis deux plugins. Le premier plugin est **org.but4reuse.adapters** à cause d'extension définis pour le point d'extension org.but4reuse.adapters. Le deuxième plugin est **osgi.but4reuse.osgiplugin.tracer** pour qu'il puisse lire les fichiers de traces d'exécutions. Nous remarquons également que le plugin org.but4reuse.adapters a des dépendances directes ou indirectes avec tous les plugins de Framework.

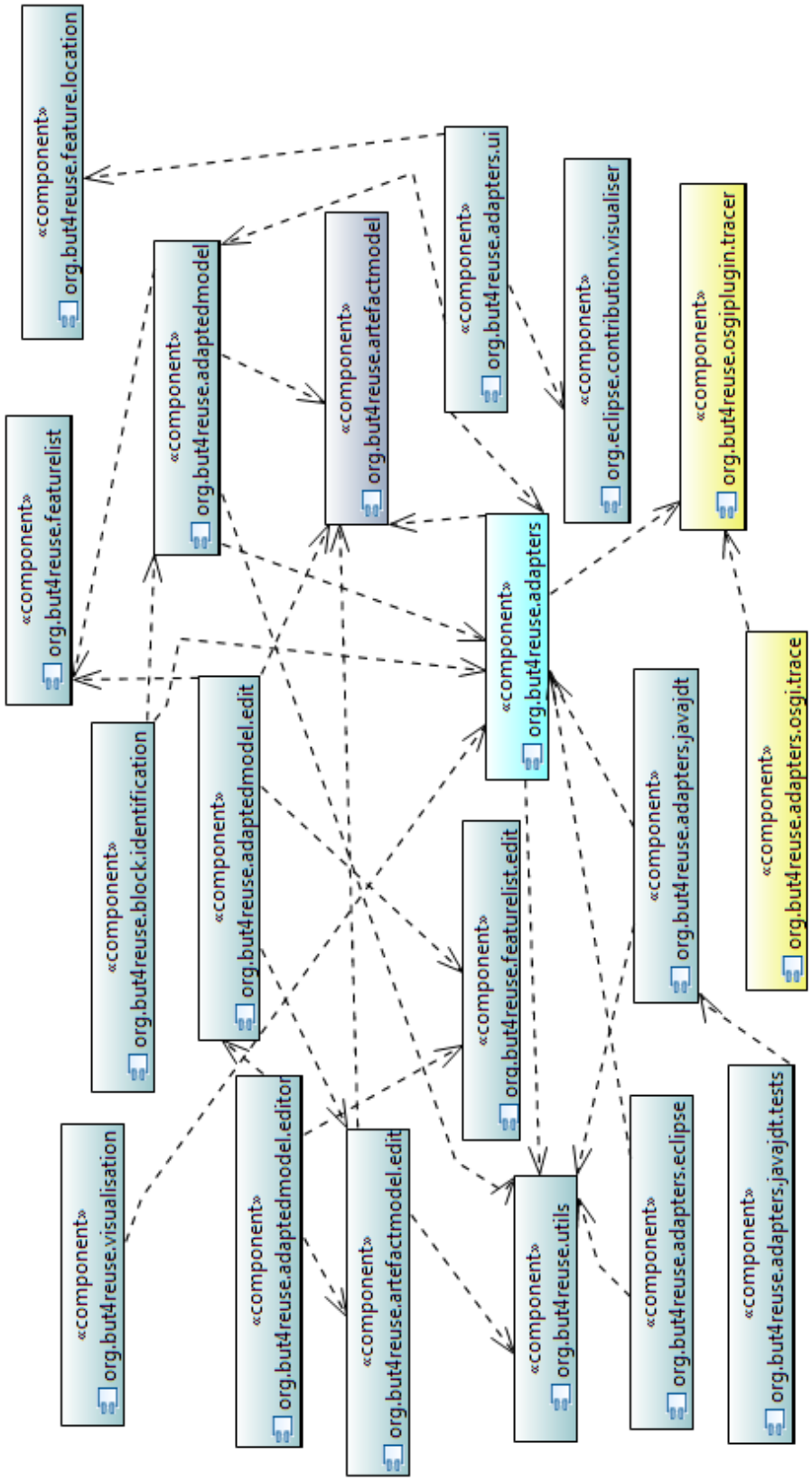


FIGURE 4.8 – Architecture de système.

4.4.2 Identification d'éléments

L'identification d'éléments consiste à décomposer l'artefact en un ensemble d'éléments qui le compose. Comme nous avons expliqué avant, une distribution Eclipse est composé de plusieurs ressources représentées par les éléments suivants :

- **PluginElement** pour représenter un plugin d'une distribution.
  - ***pluginSymbName*** : un attribut de type String représente le symbolicName d'un plugin.
  - ***pluginVersion*** : un attribut de type String représente la version d'un plugin.
  - ***bundleInfoLine*** : un attribut de type String représente les informations d'un plugin qui se retrouvent dans le fichier bundles.info d'une distribution.
  - ***fragmentHost*** : un attribut de type String représente le Fragment-Host d'un plugin.
  - ***name*** : un attribut de type String représente le BundleName d'un plugin.
  - ***absolutePath*** : un attribut de type String représente le absolu path d'un plugin.
  - ***isJar*** : un attribut de type boolean représente un plugin est un Jar ou non.
  - ***require_Bundles*** : une liste de type String représente les require_Bundle d'un plugin.
  - ***import_Packages*** : une liste de type String représente les import_Packages d'un plugin.
  - ***export_Packages*** : une liste de type String représente les export_Packages d'un plugin.
  - ***concreteConsumedServices*** : une liste de type ServiceElement représente les services concrets consommés par un plugin.
  - ***concreteRegistredServices*** : une liste de type ServiceElement représente les services concrets fournis par un plugin.
  - ***consumedServices*** : une liste de type ServiceElement représente les services consommés par un plugin.

— ***registredServices*** : une liste de type ServiceElement représente les services fournis par un plugin.

— ***durationActivity*** : un attribut de type double premièrement en calcule le pourcentage de durée d'un plugin lorsqu'il est dans l'état active par rapport au durée d'exécution total.

Nous allons ajouter à notre adaptateur Eclipse tous les attributs écrit en bleu.

Nous avons rempli les attributs :

— importPackages et exportPackages par l' analyse de fichier manifest du plugin.

— consumedServices et registedServices par l'analyse de code binaire du plugin.

— concreteConsumedServices, concreteRegistredServices et durationActivity par l'analyse de fichier de trace d'exécution du plugin.

☐ **FileElement** pour représenter tout le reste des ressources d'un distribution Eclipse. Un FileElement est défini par les attributs :

— ***uri*** : un attribut de type URI représente l'uri d'une distribution Eclipse.

— ***relativeURI*** : un attribut de type URI représente l'uri relative d'un ressource.

Nous avons ajouté l'élément

☐ **ServiceElement** représente un service d'un plugin. Un ServiceElement est définit par l'attribut :

— ***nameService*** : un attribut de type string représente le nom du service.

Afin de décomposer un artefact Eclipse, l'adaptateur effectue une traversée de l'arborescence du dossier racine Eclipse en obtenant les éléments PluginElement et FileElement.

4.4.3 Définition des contraintes structurelles

La définition des contraintes structurelles consiste à identifier les dépendances structurelles des éléments. Les dépendances des éléments de type **PluginElement** sont identifiées par les attributs de fichier manifest require_Bundle, Fragment_Host où un plugin (PluginElement) indique les SymbolicNames de plugins (PluginEle-

ment) qu'il dépend. Nous avons aussi traité les dépendances structurelles définies par l'attribut import_package dans le manifest d'un plugin, l'id de cette dépendance est "import package". Par ailleurs, les éléments de type **FileElement** dépendent de leur élément de fichier parent correspondant. L'identifiant de cet dépendance est "conteneur".

4.4.4 Définition de la métrique de similarité

La définition d'une métrique de similarité entre n'importe quelle paire d'éléments de même type. Un élément doit comparer sa similarité avec un autre élément obtenant une valeur allant de zéro (complètement différent) à un (identique).

Similarité de FileElement

La similarité entre une paire éléments de type FileElement est calculée en fonction de valeur de l'attribut 'relativeURI', si les deux valeurs sont égaux alors les deux FileElements sont similaires donc la fonction de similarité de FileElement retourne (1) , sinon retourne (0), c'est à dire les valeurs de 'relativeURI' différents alors les deux FileElements sont différents.

Similarité de ServicesElement

La similarité entre une paire éléments de type serviceElement est calculé en fonction de valeur de l'attribut serviceName , si les deux valeurs sont égaux alors les deux serviceElements sont similaire, donc la fonction de similarité de serviceElement retourne (1), sinon retourne (0). Dans ce cas, les deux serviceElements sont différents.

Similarité de PluginElement

Pour calculer la similarité des éléments de type PluginElement, en plus de la comparaison des symbolicNames, nous avons pris en considération les similarité suivantes :

— La similarité entre la durée d'activité de deux plugins. Elle est calculé en fonction de valeur d'attribut durationActivity de deux pluginElements par la formule suivante :

simlarity_durationActivity= 1-((pluginElementA.durationActivity - pluginElementB.durationActivity) /100) ; cette formule retourne une valeur entre 0 et 1 représente la similarité de durationActivity. Par exemple, si "pluginElementA.durationActivity = 60" et "pluginElementB.durationActivity = 0" alors "simlarity_durationActivity = 0,4".

La formule retourne une valeur double entre 0 et 1.

- La similarité entre les concreteConsumedServices.
- La similarité entre les concreteRegistredServices.
- La similarité entre les consumedServices.
- La similarité entre les registredServices.

Les similarités des concreteConsumedServices, concreteRegistredServices, consumedServices et registredServices égal à la moyen des similarités des éléments "serviceElement" des deux PluginElements.

Donc, la similarité entre une paire de PluginElement est calculé en fonction de la moyenne de tous les sous similarité expliquées précédemment. La valeur finale de la similarité est la moyenne de simlarity_symbolicName,simlarity_ durationActivity,similarty_concreteConsumedServices,similarty_ concreteRegistredServices,similarty_consumedServices et similarty _registredServices.

4.4.5 Construction des assets réutilisable

Nous avons utilisé la même méthode de la construction défini dans l'ancienne version d'adaptateur Eclipse (implémenté avec le Framework But4reuse). La construction d'une version d'Eclipse personnalisé a été mis en œuvre par l'ancien adaptateur en copiant les plugins et les fichiers associés aux blocs sélectionnés. Dans le cadre de cette construction, le fichier de configuration bundles.info a été ajusté si une installation Eclipse est en cours de construction.

4.5 Conclusion

Dans ce chapitre, nous avons défini un processus générique pour l'extraction d'une ligne de produits logiciels à partir des variantes d'applications et ces traces

d'exécutions en exploitant l'adaptateur BUT4Reuse spécialisé à ce type d'application. En suite, nous avons spécialisé ce processus pour les distributions d'Eclipse. Nous avons proposé un méta-modèle pour collecter les informations extraites par l'analyse de traces d'exécution. A la fin, nous avons présenté notre adaptateur des distributions d'Eclipse.

Chapitre 5

Évaluation

5.1 Introduction

Dans ce chapitre, nous allons montrer par un exemple illustratif notre processus. En outre, nous allons présenter les détails d'évaluation de notre adaptateur. L'expérimentation est effectué sur les variantes d'Eclipse. Nous utilisons quelques métriques pour faire cette évaluation. Nous évaluons les performances de notre solution par le calcul du rappel et de précision. Ensuite, nous comparons nos résultats avec ceux qui sont obtenus en exécutant l'adaptateur de[38]. Enfin, nous comparons la version minimale générée par les deux adaptateurs.

5.2 Illustration du processus par un exemple

Dans cette section, nous allons appliquer notre processus sur notre adaptateur des distributions d'Eclipse en utilisant un exemple illustratif.

5.2.1 Pré-traitement et collection des traces d'exécution

Dans cet exemple nous allons utiliser deux distributions Eclipse Oxygen 3a Windows 64 bits [1] Eclipse modling et jee comme des variantes d'artefacts. Tout d'abord, nous allons démarrer chaque distribution dans un temps de 20 minutes sans exécution des cas d'utilisations dans l'objectif de connaître les plugins actives et les

1. https ://www.eclipse.org/downloads/packages/release/oxygen/3a

service fournis et requis au démarrage d'Eclipse.

Après la collection des traces d'exécution, nous allons ajouter à chaque variante sa trace d'exécution (un répertoire nommé trace contenant le fichier tarce.xmi). La Figure 5.1 est un extrait d'un fichier trace.xmi. La nouvelle organisation d'une distribution d'Eclipse est illustré dans la figure 5.2.

```
<?xml version="1.0" encoding="ASCII"?>
<traces:Traces xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:traces="platform:/ressource/org.but4reuse.osgiplugin.tracer/model/traces.ecore"
start="2019-01-28T23:31:59.728+0100" end="2019-01-28T23:32:33.287+0100">
  <pluginTraces symbolicName="org.eclipse.osgi" version="3.11.2.v20161107-1947">
    <PluginStates state="Starting" date="2019-01-28T23:31:59.734+0100"/>
    <PluginStates state="Active" date="2019-01-28T23:32:00.083+0100">
      <registredServives name="org.osgi.service.log.LogReaderService" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.osgi.service.log.LogService" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.framework.log.FrameworkLog" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.datalocation.Location" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.environment.EnvironmentInfo" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.osgi.service.packageadmin.PackageAdmin" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.osgi.service.startlevel.StartLevel" date="2019-01-28T23:32:00.083+0100" serviceState="registered"/>
      <registredServives name="org.osgi.service.permissionadmin.PermissionAdmin" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.osgi.service.condpermadmin.ConditionalPermissionAdmin" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.osgi.service.resolver.Resolver" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.debug.DebugOptions" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="java.lang.ClassLoader" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.urlconversion.URLConverter" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.localization.BundleLocalization" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="javax.xml.parsers.SAXParserFactory" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="javax.xml.parsers.DocumentBuilderFactory" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.security.TrustEngine" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.signedcontent.SignedContentFactory" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.resolver.PlatformAdmin" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.debug.DebugOptionsListener" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <registredServives name="org.eclipse.osgi.service.runnable.StartupMonitor" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.eclipse.osgi.signedcontent.SignedContentFactory" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.eclipse.osgi.service.debug.DebugOptionsListener" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.eclipse.osgi.service.datalocation.Location" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.eclipse.osgi.service.environment.EnvironmentInfo" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.osgi.service.url.URLStreamHandlerService" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.eclipse.osgi.framework.log.FrameworkLog" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
      <consumedServices name="org.eclipse.osgi.service.runnable.StartupMonitor" date="2019-01-28T23:32:00.084+0100" serviceState="registered"/>
    </PluginStates>
  </pluginTraces>
  <pluginTraces symbolicName="org.eclipse.osgi.compatibility.state" version="1.0.200.v20160504-1419">
```

FIGURE 5.1 – Extrait d'un fichier trace.xmi.

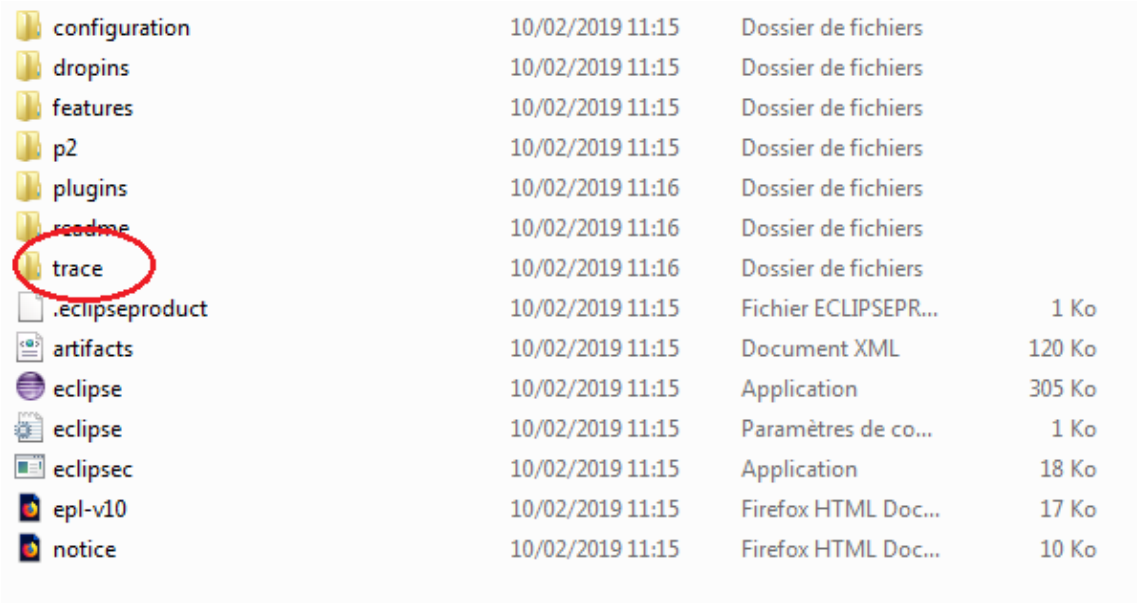


FIGURE 5.2 – Nouvelle organisation d'une distribution d'Eclipse.

5.2.2 Application de l'adaptateur OSGiTrace

Pour l'application de notre adaptateur OSGiTrace sur les deux variantes d'Eclipse nous suivons ces étapes :

1. **Choisir la Perspective BUT4Reuse**

   La perspective BUT4Reuse dans l'IDE Eclipse vous permettra d'accéder rapidement aux vues et aux assistants BUT4Reuse. Il n'est pas obligatoire de l'activer, mais cela peut faciliter votre travail. Pour l'activer :

   Window -> Open perspective -> Other... -> BUT4Reuse

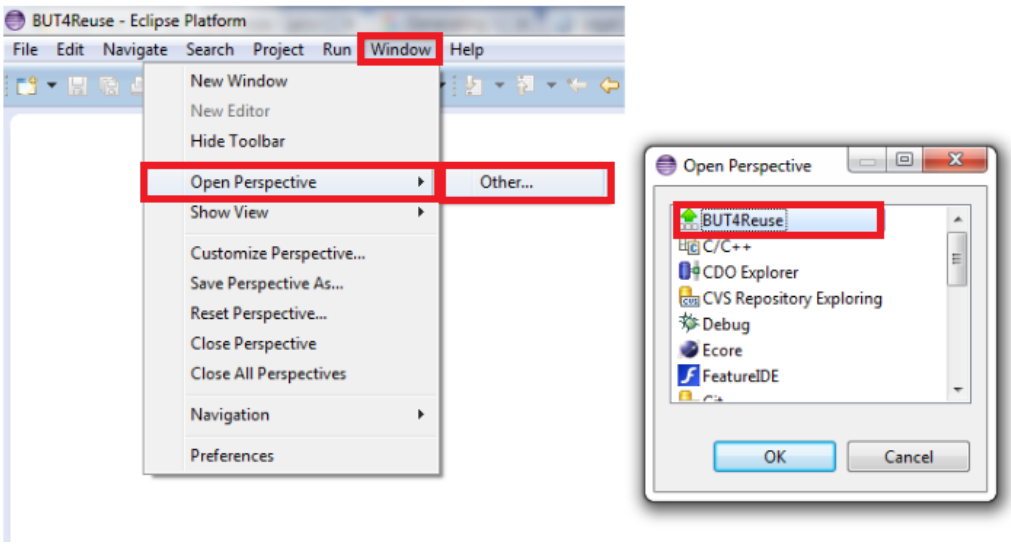


FIGURE 5.3 – BUT4Reuse perspective.

BUT4Reuse preferences

Ces préférences vous permettent de sélectionner les algorithmes que vous souhaitez utiliser ainsi que leur configuration.

Window -> Preferences -> BUT4Reuse

2. **Création d'un projet**

   BUT4Reuse ne nécessite aucune nature de projet particulière. Créez simplement un projet général.

   File -> New... -> Other... -> General -> Project.

   Si la perspective BUT4Reuse est active, vous pouvez accéder directement à l'Assistant de création.

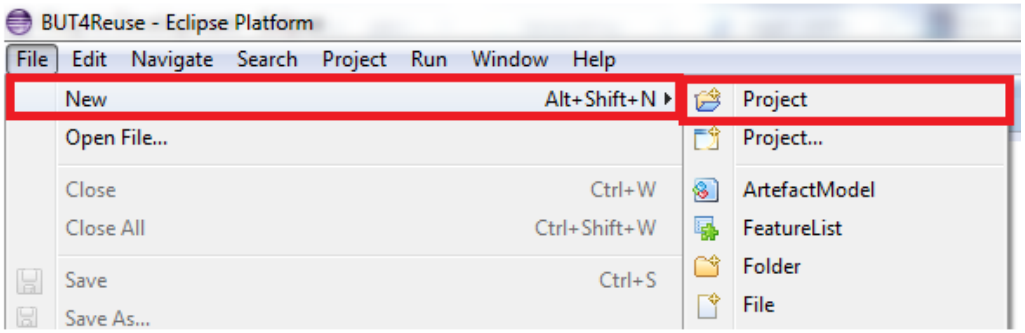


FIGURE 5.4 – créer un projet.

3. **Création d'un fichier ArtefactModel**

   New ... -> Other ... -> ArtefactModel.

4. **Ajout des artefacts Eclipse à la vue "inputDrop" de ArtefactModel**

   Les artefacts peuvent être ajoutés manuellement ou directement dans l'éditeur ArtefactModel. Vous disposez également Window -> Show view -> Other -> Input Drop vie pour ajouter les variantes d'artefact. Sélectionnez les deux distributions d'Eclipse dans le navigateur Eclipse ou sur votre bureau et déposez-les dans l'éditeur. De cette façon, les URI des artefacts seront automatiquement définis.

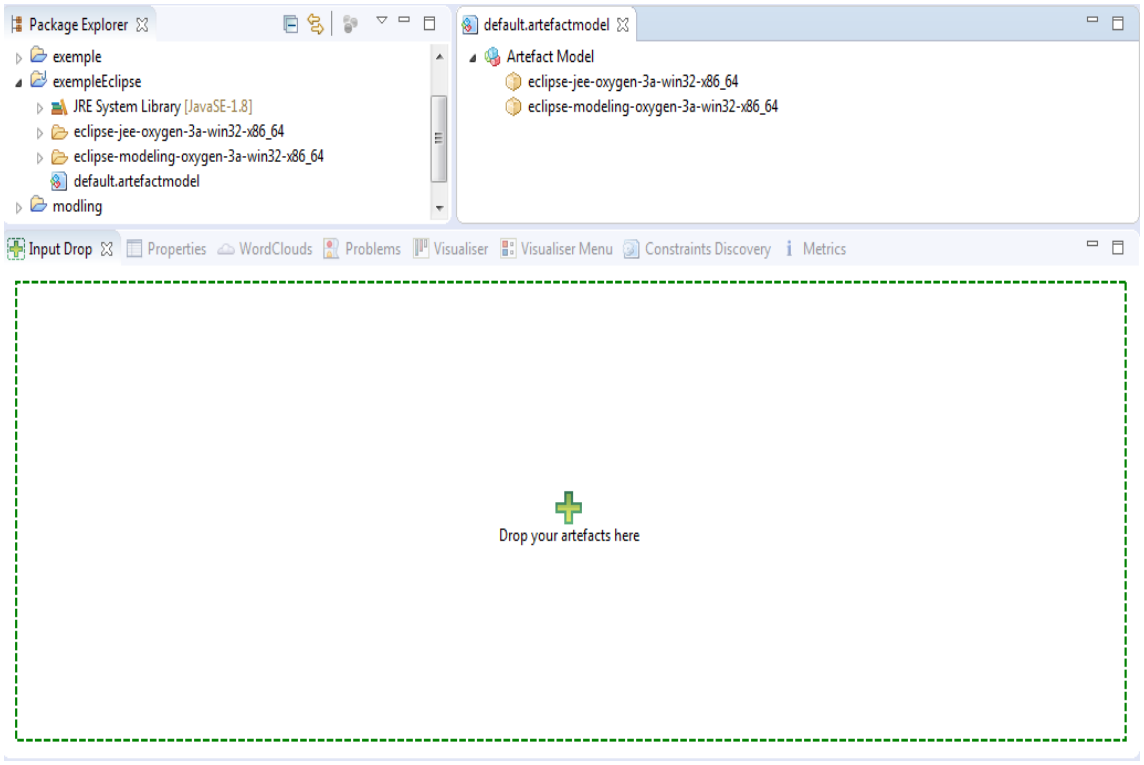


FIGURE 5.5 – Ajout d'artefacts Eclipse à ArtefactMode.

5. **identifier les points communs et les points variables** Lancer l'identification de features dans l'éditeur ArtefactModel, sélectionnez l'élément ArtefactModel (la racine) et sélectionnez "Feature Identification". Là encore, il nous sera demandé d'utiliser l'adaptateur et nous ne sélectionnerons que OSGiTrace puis cliquer sur OK.

   ArtefactModel -> Feature Identification -> OSGiTrace -> OK.

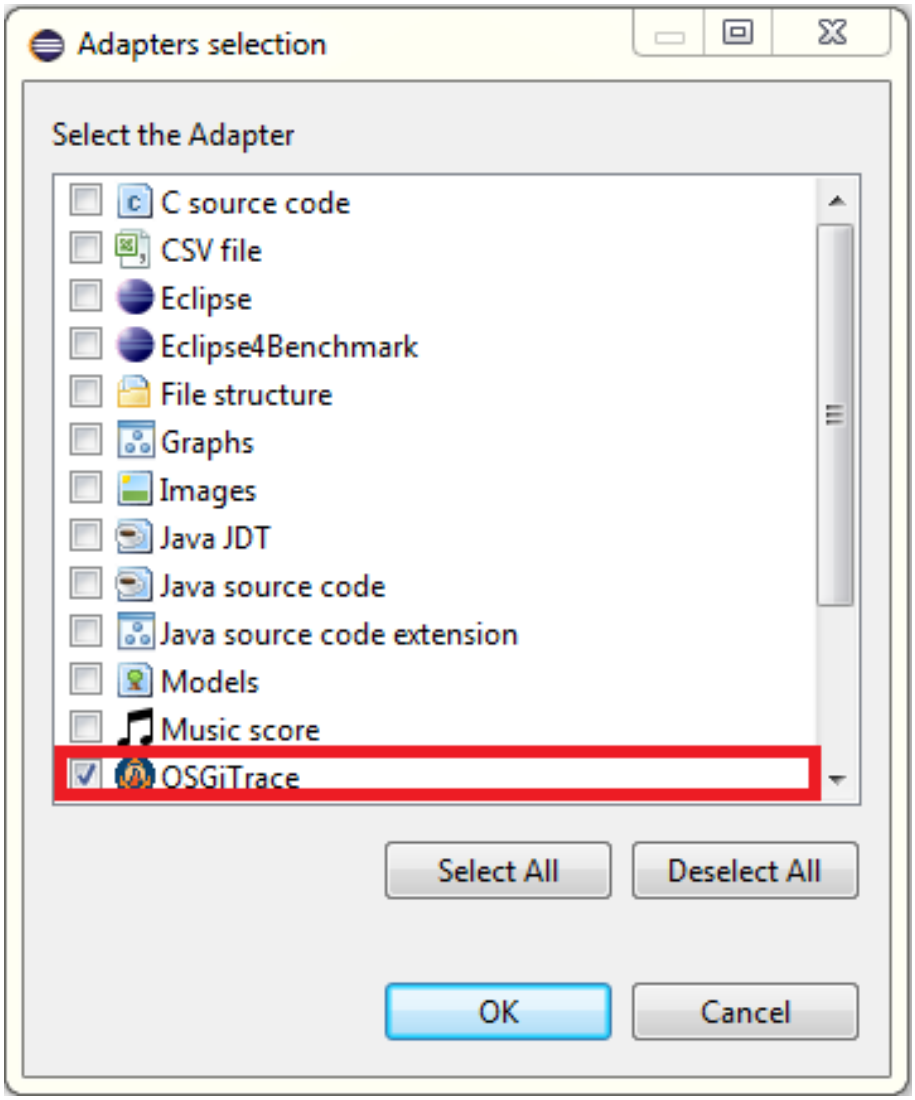


FIGURE 5.6 – Sélection de l'adaptateur OSGiTrace.

Autre Opérations BUT4Reuse

BUT4Reuse définit plusieurs opérations tel que :

Construire un bloc ou un ensemble de blocs

Dans la vue Menu Visualiser, sélectionnez un bloc (ou un ensemble), puis vous cliquez sur l'action dans la même vue Construct et vous introduisez un URI de sortie. L'URI par défaut apparu est : platform : / resource / nomProjet /.

Construire un ensemble de blocs séparément

Dans la vue Menu Visualiser,vous sélectionnez un ensemble de blocs, puis vous cliquez sur l'action Construct separately dans la même vue et vous introduisez un URI de sortie. Les blocs seront construits dans des dossiers séparés. Chaque dossier aura le nom de bloc correspondant.

Reconstruire tous les artefacts

Dans la vue Visualiser (Window -> Show view -> Other... -> Visualiser -> Visualiser), un bouton permet de reconstruire automatiquement tous les artefacts en fonction des informations des blocs. Il peut être utilisé pour vérifier si la reconstruction correspond aux artefacts d'origine.

Changer les seuils de similarité

Window -> Preferences -> BUT4Reuse -> Similarity

Sélection de l'algorithme d'identification de bloc

Window -> Preferences -> BUT4Reuse -> Block identification

Sélection des algorithmes de découverte des contraintes

Window -> Preferences -> BUT4Reuse -> Constraints discovery

Sélection de l'algorithme de localisation des features

Window -> Preferences -> BUT4Reuse -> Feature location

Sélection des visualisations actives

Window -> Preferences -> BUT4Reuse -> Visualisations. Les visualisations non sélectionnées ne seront ni affichées ni mises à jour.

Modifier le nom du bloc

vous sélectionnez le ou les blocs dans le menu Visualiseur(Window -> Show view -> Other... -> Visualiser -> Visualiser Menu), puis vous cliquez sur l'action Show

Elements(icône de la loupe). Il y a une zone de texte pour changer le nom du bloc. Après cela, toutes les visualisations seront mises à jour automatiquement.

Fusionner des blocs

Sélectionnez les blocs dans le menu Visualiser(Window -> Show view -> Other... -> Visualiser -> Visualiser Menu), puisvous cliquez sur l'action Merge les blocs sélectionnés. Après cela, toutes les visualisations seront mises à jour automatiquement.

Visualisation graphique

Après le lancement de l'identification ou de l'emplacement de features, un fichier graphml est créé dans le dossier de l'artefactmodel. Ce format de graphe peut être ouvert par exemple avec http ://gephi.github.io/. Les noeuds du graphique sont les éléments et les arêtes sont les dépendances.

Synthèse de Feature model

Dans le menu Visualiser (Window -> Show view -> Other... -> Visualiser -> Visualiser Menu), puis vous cliquez sur l'action Create Feature model. Vous Sélectionnez le dossier de sortie et un feature model sera créé pour chacun des algorithmes de synthèse de feature model enregistrés.

5.3 Méthodologie d'évaluation

Pour effectuer notre évaluation, nous allons suivre ces étapes :

1. **Évaluation des performances**

   Nous évaluons les performances de notre solution par le calcul du rappel et précision. L'efficacité des méthodes de récupération de l'information est couramment mesurée par leur précision (precision) et rappel(recall). Pour une requête donnée, la précision est le pourcentage de liens récupérés qui sont pertinents au nombre total des liens récupérés. Le rappel est le pourcentage de liens récupérés qui sont pertinents pour le total nombre de liens pertinents[60].

Ces mesures ont des valeurs comprises dans une plage [0, 1]. Une plus grande précision et rappel signifient de meilleurs résultats[59]

2. **Évaluation de la qualité de variants dérivés**

   Pour évaluer la qualité de variants dérivés, nous comparons nos résultats avec ceux qui sont obtenus en exécutant l'adaptateur de[38]. Pour cela, nous allons utiliser les 12 distributions d'Eclipse Kepler SR2 windows 64 comme une entrée pour identifier les blocs avec les deux adaptateurs. Par la suite, nous allons dériver une variante minimal qui contient uniquement le bloc 0 pour faire une comparaison quantitative et qualitative de ce dernier.

5.4 Collection de données

Dans cette expérimentation, nous avons utilisé les douze distributions Eclipse Kepler SR2 Windows 64 pour évaluer les performances de notre solution par le calcul du rappel et précision. En suite, nous comparons nos résultats avec ceux qui sont obtenus en exécutant l'adaptateur de[38].

5.4.1 Pré-traitement, collection des traces et application de l'adaptateur OSGiTrace

Dans cette expérimentation nous avons utilisé les douze distributions Eclipse Kepler SR2 Windows 64 bits [2] comme des variantes d'artefacts. Tout d'abords, on doit installer le Plugin Development Environment (PDE) [3] pour les distributions Eclipse IDE for Testers, C++, Parallel, Java. Le PDE fournit des outils pour créer, développer, tester, déboguer, construire et déployer des plugins. Dans notre outil, nous avons besoin de PDE pour la génération et l'enregistrement des traces d'exécution. Par la suite nous faisons le démarrage de chaque distribution dans un temps précis (20 minutes) sans exécuter les cas d'utilisations dans l'objectif de connaître les plugins actives et les services fournis et requis au démarrage d'Eclipse. Le Tableau 5.1 représente la taille en Méga Octet, le nombre de plugins et le nombre des features

2. http ://eclipse.org/downloads/packages/release/Kepler/SR2
3. pde.visualization Update Site - http ://download.eclipse.org/eclipse/pde/visualization/updates

Distribution	Taille(Mo)	Plugins	features
Ecl*69ipse Modeling Tools	343.1	541	176
Eclipse IDE for Automotive Software Developers	232.7	516	86
Eclipse IDE for Java EE Developers	298.1	541	156
Eclipse IDE for Java Developers	197.8	422	61
Eclipse IDE for Testers	150	337	31
Eclipse Standard	230.5	246	31
Eclipse IDE for C / C ++ Developers	205.7	406	64
Eclipse IDE Java and DSL developers	314.8	432	105
Eclipse for Parallel Application Developers	359.8	685	104
Eclipse for RCP et RAP Developers	277	450	73
Eclipse Scouts Developers	346.9	497	92
Eclipse IDE for Java and Report Developers	342.8	891	140

TABLE 5.1 – Les distributions d'Eclipse Kepler, taille, nombre des plugins, et le nombre de features.

pour chaque distribution Eclipce Kepler. Après la collection des traces d'exécution, nous allons ajouter à chaque distribution sa trace d'exécution. Les 12 fichiers "trace" peuvent être téléchargés d'un lien que nous avons crées sur github. [4]

Pour l'application de notre adaptateur OSGiTrace sur les variantes d'Eclipse, nous suivons les mêmes étapes expliquées dans l'exemple d'application de l'adaptateur OSGiTrace.

5.4.2 Évaluation des performances

La précision est calculée en fonction de : True Positive(TP) et False Positive(FP) comme présente l'équation 5.1. Le rappel est calculé en fonction de True Positive et False Negative(FN) comme présenté dans l'équation 5.2

$$\textbf{Précision} = \frac{TP}{TP + FP} \tag{5.1}$$

$$\textbf{Rappel} = \frac{TP}{TP + FN} \tag{5.2}$$

4. https ://github.com/asmarechid/DistributionEclipseTraces.git

Par exemple, lorsqu'un moteur de recherche retourne 30 pages web dont seulement 20 sont pertinentes (TP) et 10 ne le sont pas (FP), mais qu'il omet 40 autres pages pertinentes (FN), sa précision est de 20/(20+10) = 2/3 et son rappel vaut 20/(20+40) = 1/3.

Pour nous, la précision et le rappel sont calculés pour chaque bloc. Pour avoir un résultat global de la précision et rappel nous utilisons la moyenne de toutes les blocs.

Tout d'abord, nous avons implémenté un programme pour spécifier tout les features dans les variantes d'artefact Eclipse Kepler et identifiés les intersections des features entre ces artefacts. Le résultats de location des features sur les 12 distributions Eclipse Kepler sont disponible sur github. [5]

Les 12 distributions d'Eclipse Kepler SR2 windows 64 bits que nous avons utilisé contiennent 426 features. L'adaptateur OSGiTrace a réussi à identifier 202 blocs en utilisant l'algorithme Interedendent elements mais seulement 110 blocs contient des plugins, pour cela nous allons calculer le rappel et la précision pour les blocs.

Nous avons défini les True Positives, False Positives et False Negavives comme suite :

- **True Positives (TP)** : les plugins d'un bloc qui représentent les plugins des features (les plugins réellement corrects).
- **False Positives (FP)** :les plugins identifiés par l'adaptateur dans le bloc qui ne représente pas les features(les plugins n'appartient pas à les features).
- **False Negatives (FN)** : les plugins représentent les feature qui manquant dans le bloc.

5. https ://github.com/asmarechid/featuresInDistibutionsEclipseKepler.git

Numéro de Bloc	TP	FP	FN	Rappel	Précision
000	154	8	49	0,95	0,75
013	5	0	4	0,55	1
014	25	11	4	0,69	0,86
045	18	0	0	1	1
064	28	4	5	0,87	0,84
123	2	1	0	0,66	1
128	2	3	0	0,4	1
145	9	6	3	0,6	0,75
196	163	70	8	0,69	0,95
197	30	8	0	0,78	1
200	1	0	1	0,5	1
201	63	0	0	1	1
Moyenne	/	/	/	0,82	0,92

TABLE 5.2 – Calcul de Rappel et de décision.

Le Tableau 5.2 décrit les valeurs de précision et de rappel obtenues pour quelque blocs et la moyenne de précision et de rappel. Les mesures montrent que le taux de précision est relativement élevé pour les blocs. Ce qui montre l'efficacité de notre outil. Cependant, les valeurs de rappel obtenues montrent que le rappel est relativement bon sauf les cas des blocs numéro 013, 128 et 200 où le rappel est faible à cause des FP sont élevés ou égaux avec FP comme le dans le bloc 200 dont ce bloc contient uniquement 2 plugins org.eclipse.epp.package.standard et org.eclipse.team.cvs.core, le premier représente TP et l'autre FN. Un autre cas justifie la valeur de rappel obtenu est un plugin appartient à plusieurs feautres.

5.4.3 Évaluation de la qualité de variants dérivés et Comparaison des résultats

Pour évaluer la qualité de variants dérivés, nous comparons nos résultats avec ceux qui sont obtenus en exécutant l'adaptateur de[38]. Pour cela, nous avons utilisé

les 12 distributions Eclipse Kepler SR2 windows 64 comme entrée pour identifier les blocs avec les deux adaptateurs. Par la suite, nous avons dérivé une variante minimale qui contient uniquement le bloc 0 pour faire une comparaison quantitative et qualitative de ce dernier.

Cette section présente une comparaison des résultats obtenu par ancienne version de l'adaptateur Eclipse avec les résultats de notre adaptateur OSGiTrace. Les performances sont mesurés en termes de temps d'exécution à l'aide d'un ordinateur portable hp avec un processeur Intel (R) Core (TM) CPU i3-6006U @ 2.00GHz 2.00GHz, 4 Go de RAM, avec Windows 7 64 bits. Nous avons estimé la moyenne de 3 exécutions. Nous avons utilisé les 12 Eclipse Kepler SR2 Windows 64 bits après l'ajout de PDE pour l'adaptateur Eclipse et pour l'adaptateur OSGiTrace nous avons utilisé les mêmes distributions avec les fichiers des traces d'exécution en tant que des artefacts entrés pour les adaptateur.

Adaptateur	**OSGiTrace**	**Eclipse**
Nombre minimum d'éléments par distribution	1951	1949
Nombre maximum d'éléments par distribution	5341	5339
Nombre moyen d'éléments par distribution	3649.	3647
Nombre de blocks	202	163
Nombre minimum d'éléments par block	1	1
Nombre maximum d'éléments par block	3228	3079
Nombre moyen d'éléments par block	107	129
Nombre de contraints de block	1227	520
Temps pour adapté les distributions(ms)	709029	129032

TABLE 5.3 – Résultats obtenu par l'adaptateur Eclipse et OSGiTrace.

Le Tableau 5.3 résume les résultats obtenu par les deux adaptateurs lors de l'utilisation des variantes d'Eclipse Kepler par la configuration par défaut de BUT4Reuse.

Le nombre moyen d'éléments par artefacts identifiés par notre adaptateur est 3649 qui égale le nombre d'éléments identifiés par l'adaptateur Eclipse 3647 en plus élément répertoire trace et le fichier trace.xmi que nous avons ajouté au distribution. Notre adaptateur pends 11 minutes et 81 secondes pour décomposer les 12 variantes

d'Eclipse en des Plugin et File Elements. l'autre adaptateur pendre 3 minutes et 15 seconde . Notre adaptateur pendre plus de temps que l'autre à cause d'analyse des byte code et des fichiers trace.

Nous avons utilisé la configuration par défaut de BUT4Reuse. L'algorithme d'identification de bloc utilisé est les éléments interdépendants. Pour l'adaptateur OSGiTrace 202 blocs étaient identifiés. La moyenne d'éléments par bloc était de 107. L'algorithme d'identification des blocs n'a pas pris 1792 millisecondes.

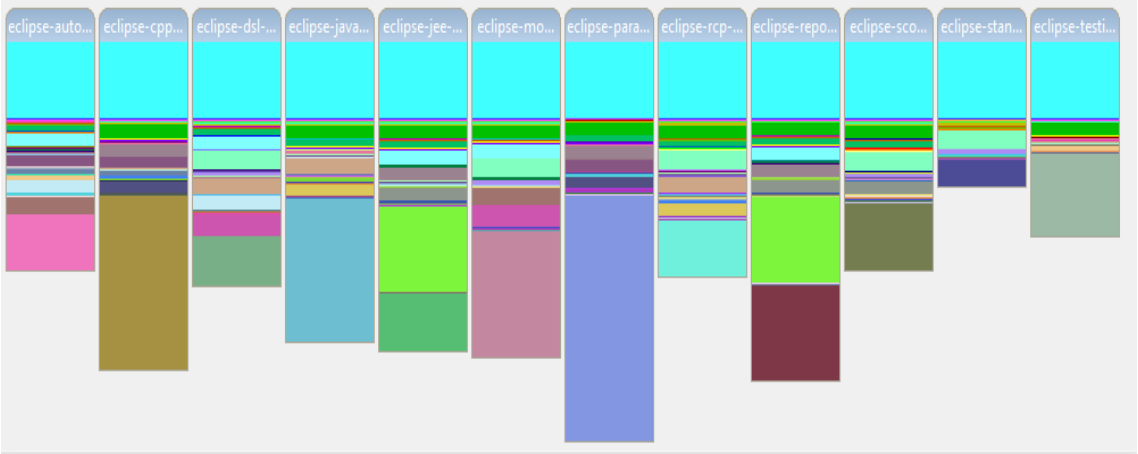


FIGURE 5.7 – Visualisation montrant les blocs (couleurs) sur les artefacts (barres) distributeurs généré par l'adaptateur OSGiTrace.

Pour l'adaptateur Eclipse, 163 blocs étaient identifiés. La moyenne d'éléments par bloc est de 129 éléments . L'algorithme d'identification des blocs n'a pas pris 1427 millisecondes.

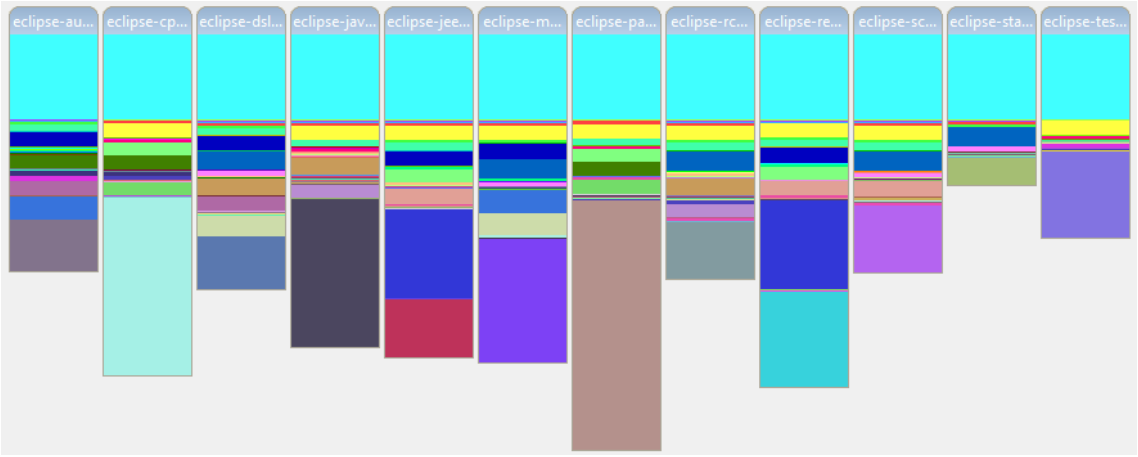


FIGURE 5.8 – Visualisation montrant les blocs (couleurs) sur les artefacts (barres) distributeurs généré par l'adaptateur Eclipse.

La figure 5.7 une visualisation montre les blocs sur les distributions. L'adaptateur OSGiTrace capable à identifier la variabilité plus que l'adaptateur Eclipse grâce à la métrique de similarité utilisée qui prendre en considération non seulement la comparaison des symbolicName mais aussi les services fournis et requis concerte et potentiel et la durée d'activité.

Le nombre des contraintes de bloc est différent entre les deux adaptateurs. Le nombre de contraintes de bloc obtenus par notre adaptateur est inférieur que l'autre adaptateur 1227 »> 520 car nous avons identifié les dépendances entre les plugins de type import_package plus-que les dépendances de type require_bundle.

5.4.4 Comparaison quantitative et qualitative de bloc 0

Le bloc 0 représente le noyau des distribuions d'Eclipse. Il contient les plugins et fichiers commun entre tout les distributions. Le bloc 0 identifié par notre adaptateur se compose de 889 FileElement et 148 plugins. Le bloc 0 identifie par l'ancien version de l'adaptateur se compose de 887 FileElement et 210 PluginElement. Nous remarquons que le le nombre de FileElement augmente par 2 FileElement qui présentent le répertoire trace et fichier trace.xmi. Par contre le nombre de PluginElement diminuer grâce à la métrique de similarité utilisé qui pendre en considération les services fournis, requis(concerte et potentiel) et la durée d'activité extrait par l'analyse des

bytes code et des traces d'exécution. Le tableau 5.4 résume la taille et le contenu de bloc 0 obtenu par le deux adaptateurs.

	Adaptateur OSGiTrace	**Adaptateur Eclipse**
Taille(Mo)	48,3	94,7
FileElement	889	887
PluginElement	148	210

TABLE 5.4 – Comparison de la taille et le contenu de bloc 0.

Les plugins manquent dans le bloc 0 identifié par notre adaptateur de distributions d'Eclipse par rapport au bloc 0 identifié par l'ancien version d'adaptateur sont les plugins responsables aux opérations de recherche et d'aide.

Une version minimale d'Eclipse contient uniquement le bloc numéro 0 qui est le bloc commun entre touts les distributions. Nous pouvons construire une version minimale d'Eclipse par le sélection de bloc 0 dans le Menu Visualisation , puis nous cliquons sur l'action dans la même vue Construct comme montre la Figure5.9 et introduisez un URI de sortie.

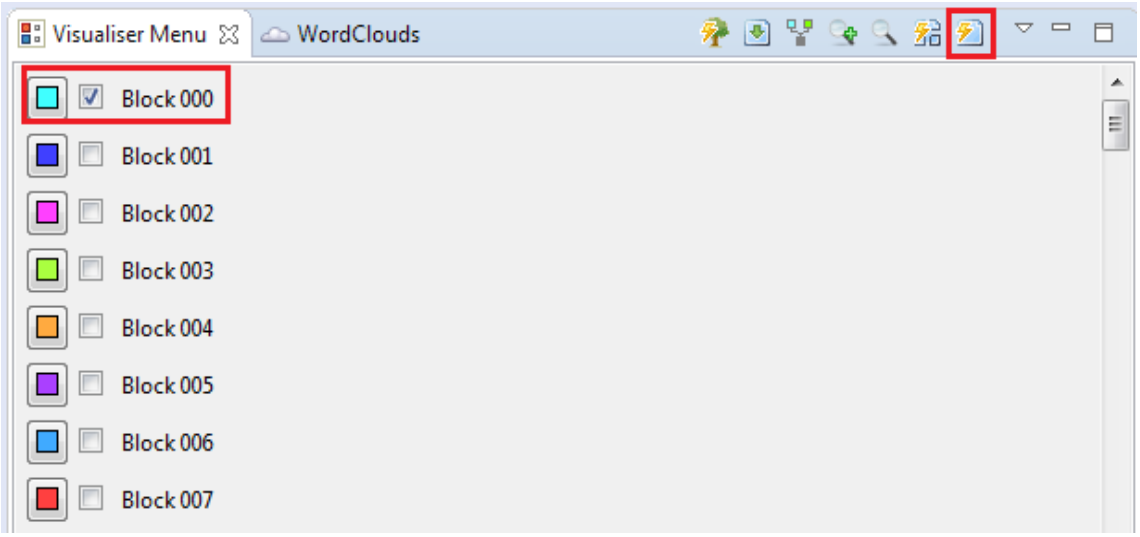


FIGURE 5.9 – Construire une version minimal.

La taille d'une version minimale générée par notre adaptateur est 51,7 Mo, en revanche la taille de cette version minimale générée par l'autre adaptateur égale 98,7 Mo. Il s'agit d'une grande déférence entre la taille de deux versions.

5.5 conclusion

Dans ce chapitre, nous avons présenté un exemple illustratif de notre processus. En outre, nous avons évalué notre adaptateur par le calcul de rappel et de décision. Ensuite nous avons comparé les résultats obtenus par notre adaptateur et celle de l'ancien version d'adaptateur Eclipse en utilisant 12 distributions d'Eclipse Kepler. Enfin, nous avons comparé la version minimale générée par les deux adaptateurs.

FileElement	ServiceElement	PluginElement
uri	serviceName	SymbolicName
relativeUri		pluginVersion
		bundleInfoLine
		fragmentHost
		name
		absolutePath
		isJar
		concreteConsumedServices
		concreteRegistredServices
		require_Bundles
		import_packages
		export_packages
		consumedServices
		registredServices
		durationActivity

Conclusion générale

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie logicielle et exactement dans le domaine de réutilisation de logiciel. LdP est une stratégie de la réutilisation de logiciel qui définie comme un ensemble de systèmes partageant un ensemble de features communes et variables satisfaisants des besoins spécifiques pour un domaine particulier L'avantage de ce dernier consiste à réduire les coûts de production et la livraison plus rapide des systèmes personnalisé.

Notre objectif dans ce travail était de développer un adaptateur pour les distributions d'Eclipse en exploitant les traces d'exécution de ces derniers. Nous avons proposé un processus pour la génération des traces d'exécution et l'extraction de LdP, cela se fait à partir d'un ensemble de variantes d'application. Ce processus peut être adapté facilement à des différents types d'artefacts. Nous avons appliqué le processus sur les distributions Eclipse, nous avons proposé un méta modèle pour la collection des traces d'exécution. Par la suite, nous avons implémenté un adaptateur qu'il a comme entrée le code source et Byte-code des distributions Eclipse et les traces d'exécution qui sont générées en exécutant un ensemble de use cases. La sortie c'est une LdP. Enfin, nous avons mené une expérimentation pour évaluer le process et l'adaptateur. Les résultats obtenus par le calcul du rappel et de la décision ont montré l'efficacité de notre solution.

Nous envisageons d'enrichir ce travail par :

— L'application de notre processus proposé sur une autre application tel que les applications Android. Par la suite implémenter un adaptateur Android qui prend en considération les traces d'exécution.

— L'implémentation d'un algorithme d'identification et de localisation des fea-

tures d'Eclipse.

— Ajouté au Framework BUT4Reuse une fonctionnalité de sauvegarder la visualisation artefacts on blocs mieux de faire des capture écran.

— Améliorer les performances de notre adaptateur.

Bibliographie

[1] Charles W. Krueger. "Software reuse", ACM Comput. Surv, 24, 1992.

[2] William Frakes, Virginia Tech et Carol Terry "Software Reuse : Metrics and Models", 28, No. 2, June 1996.

[3] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides , "Design Patterns Element s of Reusable Object-Oriented Software", Addison-Wesley,1994.

[4] Jesus Bisbal, Deirdre Lawless, Bing Wu, and Jane Grimson, "Legacy Information Systems : Issues and Directions", IEEE Software,1999.

[5] Yin, Tingjie, Yingying Wang, Xuxin Chu, Ying Fu, Lei Wang, Jianping Zhou, Xiaomeng Tang, Ji-yong Liu and Meirong Huo,"Free Adriamycin-Loaded pH/-Reduction Dual-Responsive Hyaluronic Acid-Adriamycin Prodrug Micelles for Efficient Cancer Therapy", ACS applied materials and interfaces 10 42 (2018) : 35693-35704 .

[6] Chen, I. J.,Injazz J. Chen. (1999). "Planning for ERP systems :analysis and future trend. Business Process Management Journal, 7(5), 374 ?386. http ://-doi.org/10.1108/14637150110406768

[7] S. Chardigny, "Extraction of a component-based software architecture from an object-oriented system. exploration approach",2009.

[8] C. Benoit, "Ingénierie Dirigée par les Modèles (IDM)-État de l'art". 2008.

[9] WorldWideWeb Consortium (W3C), "W3c working group. note 11 : Web services architecture" https ://www.w3.org/TR/ws-arch/stakeholder,November 2000. Consulté le :13.02.2019.

[10] https ://www.computerhope.com/jargon/p/proggene.htm,consulté le 13.02.2019.

[11] P. Clements et L. Northrop," Software Product Lines : Practices and Patterns", 1er éd. Addison Wesley, 2001.

[12] Linda M. Northrop and Paul C. Clements, Felix Bachmann, John Bergey, Gary Chastek, Sholom Cohen, Patrick Donohoe, Lawrence Jones, Robert Krut, Reed Little, John McGregor, et Liam O'Brien, A FRAMEWORK FOR SOFTWARE PRODUCT LINE PRACTICE, VERSION 5.0, SOFTWARE ENGINEERING INSTITUTE,CARNEGIE ,MELLON UNIVERSITY,December 2012.

[13] D. M. Weiss et C. T. R. Lai," Software Product-Line Engineering : A Family-Based Software Development Process". Addison-Wesley Professional, 1999.

[14] F.Coallier, R.Champagne ."A Product Line engineering practices model", Science of Computer Programming,2005.

[15] Frank van der Linden, Klaus Schmid, and Eelco Rommes. Software product lines in action - the best industrial practice in product line engineering. Springer, 2007.

[16] K. Pohl, G. Böckle, et F. V. D. Linden, "Software Product Line Engineering : Foundations, Principles, and Techniques". Birkhäuser, 2005.

[17] Gunter Halmans and Klaus Pohl." Communicating the variability of a software-product family to customers". Informatik - Forschung und Entwicklung, 18 :113-131, 2004.

[18] Felix Bachmann and Paul Clements,"Variability in software product lines", Technical report, Software Engineering Institute, Carnegie Mellon University, 2005.Technical Report CMU/SEI-2005-TR-012.

[19] D.Benavides, S.Segura, and A. Ruiz-Cortés, "Automated Analysis of Feature Models", 20 Years Later : A Literature Review, 35(6) :615-636, 2010.

[20] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, et A. S. Peterson, "Caractéristique-Oriented Domain Analysis (FODA) Feasibility Study", Technical report CMU/SEI TR-21, USA, 1990.

[21] Sven Apel, Christian Lengauer, Bernhard Moller, and Christian Kastner. "An algebra for features and feature composition", In Proceedings of the 12th international conference on Algebraic Methodology and Software Technology, AMAST 2008, pages 36-50, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, et A. S. Peterson, « Feature-Oriented Domain Analysis (FODA) Feasibility Study », Technical report, CMU/SEI TR-21, USA, nov. 1990.

[23] K. Czarnecki et U. Eisenecker, Generative Programming : Methods, Techniques and Applications, 1er éd. USA : Addison Wesley, 2000.

[24] Linda Northrop, "Software product line adoption roadmap", Technical Report CMU/SEI- 2004-TR-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2004.

[25] Charles W. Krueger. "Easing the transition to software mass customization In Software Product-Family Engineering", 4th International Workshop,PFE 2001, Bilbao, Spain,October 3-5, 2001, Revised Papers, volume 2290 of Lecture Notes in Computer Science,pages 282 ?293. Springer, 2001.

[26] Martinez, J. (2016)."Exploration des variantes d'artefacts logiciels pour une analyse et une migration vers des lignes de produits".

[27] Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides, Design Patterns-Elements of "Reusable Object-Oriented Software",Addison-Wesley, 1995

[28] M. Fayad, Douglas C. Schmidt," Object-Oriented Application Frameworks", communications of the ACM, Volume 40 Issue 10,page 32-38, Oct. 1997 .

[29] J. Poulin, "Measuring Software Reuse-Principles, Practices, and Economic Models",Addison-Wesley, 1997.

[30] Tewfik Ziadi and Jean-Marc Jézéquel. "Product line engineering with the uml : Deriving products". In K. Pohl, editor, Software Product Lines, pages 557-586.Springer Verlag, 2006.

[31] Tewfik Ziadi, Luz Frias, Marcos Aurélio Almeida da Silva, and Mikal Ziane. Feature identification from the source code of product variants. In 16th European Conference on Software Maintenance and Reengineering, CSMR 2012,

Szeged, Hungary, March 27-30, 2012, pages 417–422. IEEE Computer Society, 2012.

[32] Mark A. Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann and Ian H. Witten. The WEKA data mining software : an update. SIGKDD Explorations, 11(1), 2009.

[33] Gene Smith. Tagging : People-powered Metadata for the Social Web. New Riders Publishing, 2007.

[34] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. Extraction of feature models from formal contexts. In Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, A ugust 22-26, 2011. Workshop Proceedings (Volume2), page 4. ACM, 2011.

[35] Ra'Fat AL-msie'deen, Marianne Huchard, Abdelhak-Djamel D. Seriai, Christelle Urtado, and Sylvain Vauttier. Concept lattices : A representation space to structure software variability. In Information and Communication Systems (ICICS), 2014 5th International Conference on, pages 1–6, April 2014.

[36] Anas Shatnawi, Abdelhak Seriai, and Houari A. Sahraoui. Recovering architectural variability of a family of product variants. CoRR, abs/1606.00137, 2016.

[37] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves LeTraon. Bottom-up adoption of software product lines : a generic and extensible approach. In Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, pages 101-110. ACM, 2015

[38] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves LeTraon. Bottom-up adoption of software product lines : a generic and extensible approach.

[39] Sven Apel, Christian Kastner, and Christian Lengauer. FEATUREHOUSE : Languageindependent, automated software composition. In Proceedings of the 31st International Conference on Software Engineering, ICSE '09, pages 221–231, Washington, DC, USA, 2009. IEEE Computer Society.

[40] Loïc Girault, Cédric Besse, and Mikal Ziane. Puck : an architecture refatoring tool. https ://pages.lip6.fr/puck.

[41] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. FeatureIDE : An extensible framework for feature-oriented software development. Science of Computer Programming, 79(0), 2014.

[42] OSGi.org. https ://www.osgi.org/developer/architecture/ ,consulté le 18/10/2018.

[43] LarsVogeletSimon Scholz(c),OSGiModularityTutorial, http ://www.vogella.com/tutorials/OSGi/article.html,consulté le :20/10/2018.

[44] Eclipse,https ://www.eclipse.org/ ,consulté le 26/01/2019.

[45] Apache Felix. http ://felix.apache.org/site/index.html.

[46] Jeff McAffer, Paul VanderLei et Simon Archer, OSGi and Equinox : Creating Highly Modular Java Systems, The Eclipse Series.Dans les éditions de Jeff McAffer, Erich Gamma et John Weigand.Addison Wesley, 2010.

[47] Alliance. "The OSGi Alliance OSGi Core", June, 2014.

[48] Lars Vogel, OSGiservicesTutoriel, http ://www.vogella.com/tutorials/OSGiServices/article.html le :22/10/2018.

[49] N.Bartlett, "OSGi In Practice". Bd January, 11, 229,11 2009.

[50] Chouki TIBERMACINE, Programmation par composants avec OSGi, cours Master, 2013-2014, http ://www.lirmm.fr/ tibermacin/ens/gmin30f/.

[51] Lee Nackman, John Wiegand et Addison Wesley, Eclipse Plug-ins, 3rd edition, Eric Clayberg et Dan Rubel, The Eclipse Series. Dans les éditions de Erich Gamma, 2009.

[52] OSGi, https ://www.osgi.org/osgi-compliance/osgi-certification/osgi-certified-products/, consulté le 25/01/2019.

[53] Eclipse ide- tutoriel. http ://www.vogella.com/tutorials/Eclipse/article.html/, consulté le 26/01/2019.

[54] Eclipse-kepler, https ://www.eclipse.org/downloads/packages/release/kepler/sr2 , consulté 29/01/2018.

[55] http ://help.eclipse.org/help33/topic/org.eclipse.pde.doc.user/guide/tools/editors /product_editor/configuration.htm , consulté le 17/03/2018.

[56] https ://wiki.eclipse.org/Equinox/p2/Getting_Started, consulté le 18/03/2018.

[57] https ://wiki.eclipse.org/Eclipse.ini , consulté le 18/03/2018.

[58] Vogella : Eclipse extension points and extensions - tutorial. http ://www.vogella. com/tutorials/EclipseExtensionPoint/article.html. Consulté le 18/02/2018.

[59] G. Salton et M. J. McGill, "Introduction to Modern Information Retrieval", USA : McGraw-Hill, Inc, 1986.

[60] Hamzeh Eyal Salman, Abdelhak-Djamel Seriai, and Christophe Dony. Feature-to-code traceability in a collection of software variants : Combining formal concept analysis and information retrieval. In Intern. Conf. on Inform. Reuse and Integr.IRI, pages 209-216, 2013.