UNIVERSITE BRETAGNE LOIRE

الجامعة اللبنانية
UNIVERSITE LIBANAISE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Mira ABBOUD

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
Docteur de L'Université Libanaise
sous le sceau de l'Université Bretagne Loire

École doctorale : École Doctorale Mathématiques et STIC

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Soutenue le 25 Octobre 2017

Contribution à l'élaboration d'un processus d'extraction des architectures logicielles: Méta-modèle, méthode et outil

JURY

Président :	**M. Flavio OQUENDO**, Professeur, Université de Bretagne-Sud
Rapporteurs :	**M. Djamel BENSLIMANE**, Professeur, Université Claude Bernard Lyon I
	M^{me} Selmin NURCAN, Maître de conf., Université Paris 1 - Panthéon - Sorbonne
Examinateurs :	**M. Ihab SBEITY**, Maître de conf., Université Libanaise
	M. Pascal ANDRÉ, Maître de conf., Université de Nantes
Directeurs de thèse :	**M. Mourad OUSSALAH**, Professeur, Université de Nantes
	M. Mohamad DBOUK, Professeur, Université Libanaise
Co-directrice de thèse :	**M^{me} Hala NAJA**, Maître de conf., Université Libanaise

Remerciements

A l'issue de la rédaction de cette thèse, je suis convaincue que la thèse est loin d'être un travail solitaire. En effet, je n'aurais jamais pu réaliser ce travail doctoral sans le soutien d'un grand nombre de personnes dont la générosité, la bonne humeur et l'intérêt manifestés à l'égard de ma recherche m'ont permis de progresser dans cette phase délicate.

J'aimerais tout abord remercier mon directeur de thèse, M. Mourad OUSALAH, Professeur des Universités au Laboratoire des Sciences du Numérique de Nantes (LS2N) et à l'Université de Nantes, pour m'avoir choisi pour cette thèse et pour m'avoir accueilli au sein de l'équipe AeLOS. Je lui suis également reconnaissante pour m'avoir appris à être plus autonome tout au long de ce travail de recherche tout en me supportant en chaque phase, progrès et difficulté confronté durant ce travail. Ces remarques valorisantes m'ont sans aucun doute permis de préciser la problématique de la thèse et aboutir à cette finalité.

Je remercie également ma codirectrice de thèse, Mme Hala NAJA, Maître de conférences à l'Université Libanaise, pour son support et sa confiance en moi. Sans son encouragement, ses efforts cette thèse n'a pas vu lumière. Les paroles ne seront plus suffisantes pour vous dire merci. Les corrections des papiers, les remarques qui me poussent toujours vers l'avant sans oublié votre présence presque journalière pour des conversations à aspect théorique, logistique et particulier.

Je souhaite aussi remercier mon directeur de thèse au Liban, M. Mohamad Dbouk, Professeur des Universités à l'Université Libanaise, d'avoir accepté de me prendre sous sa direction au Liban. Je lui suis reconnaissant pour ses conseils avisés et son écoute qui ont été prépondérants pour la bonne réussite de cette thèse.

Je tiens à remercier de même, les rapporteurs de cette thèse M. Djamel Benslimane, Professeur des Universités à l'Université Claude Bernard Lyon I, et Mme Selmin Nurcan, Maître de conférences à l'Université Paris 1-Panthéon-Sorbonne, pour avoir accepter de valoriser ce travail doctoral, pour leurs multiples conseils et pour toutes les heures de lecture, de préparation pour en discuter au fond le manuscrit et le travail présenté.

J'associe à ces remerciements M. Ihab Sbeity, Maître de conférences à l'Université Libanaise, et M. Flavio Oquendo, Professeur des Universités à l'Université de Bretagne-Sud, et M. Pascal André, Maître de conférences à l'Université de Nantes, pour avoir accepté d'examiner mon travail et pour l'intérêt qu'ils ont porté à cette thèse.

Je souhaiterais exprimer ma gratitude aux Prof. Pierre COINTE, directeur du laboratoire LINA, et Prof. Mohamad KHALIL, directeur du centre AZM pour la recherche en biotechnologie et ses applications, qui m'ont accueilli au sein de leurs établissements. J'étais vraiment entourée par une famille qui m'encourage et qui me procure l'ambiance nécessaire pour le mieux de travail de recherche.

Finalement, ces remerciements seraient incomplets si je n'en adressais pas à ma famille. Parents, frères et amis, merci d'être avec moi, tout au long de ce parcours.

Résumé

Face à la complexité croissante des systèmes logiciels, les architectures logicielles sont apparues comme un allié précieux pour la conception et la maintenance de ces systèmes. Cependant, pour de nombreux systèmes, la représentation de leur architecture n'est pas fiable ; elle est soit indisponible, soit insuffisante ou soit non mise à jour. Pour pallier ce problème qui met en danger la maintenance, l'évolution, la réutilisation et la migration d'un système, l'extraction d'une architecture du système est souvent proposée comme une bonne alternative. L'extraction d'une architecture logicielle est définie comme la science de l'analyse et de la conversion du code source en une architecture logicielle. Cette thèse contribue à apporter une solution au problème d'inexistence d'outil de mesure pour les processus d'extraction d'une architecture logicielle. Ainsi, nous proposons un méta-modèle appelé SArEM *(Software Architecture Extraction Meta-model)* qui spécifie les différents processus d'extraction d'une architecture logicielle. Le méta-modèle est basé sur le méta-modèle SPEM et couvre les principaux concepts des processus d'extraction d'une architecture logicielle.

En outre, nous fournissons un outil qui permet aux architectes de construire leur propre processus, d'interagir avec les sorties générées et de découvrir une architecture logicielle conforme à leurs souhaits. Plus précisément, nous proposons une approche d'extraction d'une architecture logicielle appelée SAD *(Software Architecture Discovery)* basée sur ECD (Extraction de Connaissances à partir des Données). SAD consiste à considérer l'extraction d'une architecture logicielle comme un processus de découverte de nouvelles connaissances. Ainsi, notre contribution est articulée autour deux points : le premier point est la suggestion d'un processus générique pour l'extraction d'une architecture logicielle et le second point est l'élaboration d'une extension d'un outil ECD qui supporte l'exécution des processus d'extraction d'une architecture logicielle.

Abstract

Face to the exponential growth in the size and complexity of software systems, software architectures emerge as a valuable ally for the design and maintenance of these systems. However, for many systems, their architecture representation is not reliable ; it might be unavailable, insufficient, or out of date. To overcome this problem that puts the system maintenance, evolution, reuse and migration in danger, the extraction of the system architecture is proposed. The latter is defined as the science of analyzing and converting the source code to a software architecture. The thesis treats the gap of a measurement tool for software architecture extraction processes. We propose a meta-model called SArEM (Software Architecture Extraction Meta-model) that specifies the software architecture extraction processes. The meta-model is based on SPEM meta-model and covers the main concepts of software architecture extraction processes.

Furthermore, we provide a manner that allows the architects to build their own process, interact with the generated outputs and discover a software architecture that satisfies them. Specifically, we propose a software architecture extraction approach called SAD (Software Architecture Discovery) based on KDD (Knowledge Discovery in Databases). SAD consists in considering the extraction of a software architecture as a process of discovering new knowledge. Thus, the contribution is centered on two points : the first point is the suggestion of a generic software architecture extraction process and the second point is the elaboration of a KDD tool extension that supports the execution of software architecture extraction processes.

Table des matières

Table des figures

Liste des tableaux

1

Introduction

Dans le domaine de la maintenance des systèmes informatiques, l'extraction des architectures logicielles constitue un domaine qui a pris de plus en plus d'importance au cours des dernières années. En effet, l'extraction d'une architecture logicielle est un processus qui a pour but de créer une représentation du système à un haut niveau d'abstraction. L'importance de ce processus repose sur la nécessité incontournable d'existence d'une représentation architecturale, faute de quoi un système à forte composante logicielle est confronté à la pénalité de "mort" car sa maintenance, évolution et compréhension sont fortement compromises.

Contexte

Une architecture logicielle est considérée comme étant la ou les structure(s) du système où chaque structure comprend des éléments logiciels (tels que les classes, les groupes des classes, les composants, ...), des propriétés extérieurement visibles de ces éléments et des relations entre eux (tels que des appels des fonctions, des relations entre les classes, ...) [Len et al., 2003]. Chaque système possède une architecture, de la même manière que chaque bâtiment, pont et navire cuirassé ont une architecture. Selon la norme ISO/IEC/IEEE 42010 :2011 [Commission et others, 2011], une architecture logicielle est représentée concrètement par une description d'architecture. Cette dernière est un artefact documentant l'architecture de manière à ce que les intervenants du système puissent comprendre le système et a un important impact sur la qualité, la performance, la maintenance et le succès global du système.

Cependant, pour de nombreux systèmes, leur description d'architecture n'est pas fiable : elle est soit indisponible, soit incomplète. En fait, si nous sommes dans un monde idéal, une architecture logicielle est documentée durant tout le cycle de vie du système et elle est régulièrement mise à jour avec les modifications du système. Mais, en réalité, les tâches de documentation d'une architecture logicielle sont coûteuses et prennent extrêmement du temps [Gorton, 2011]. Par conséquent, la plupart des environnements de développement négligent les tâches de documentation des systèmes et se concentrent uniquement sur leur codage. Ainsi, comme résultat, la plupart des systèmes existants ne possèdent pas une représentation de leur architecture alors que d'autres ont une représentation incomplète. Pour résoudre ce problème, l'extraction de l'architecture du système est proposée. L'extraction d'une architecture logicielle est considérée comme le processus d'analyse de plusieurs inputs tels que le code source du système afin d'identifier

une représentation architecturale du système. Cette représentation décrit le système à un haut niveau d'abstraction facilitant ainsi la compréhension globale du système.

Premièrement, nos travaux s'intéressent à la modélisation des processus d'extraction d'une architecture logicielle. La modélisation est la génération des méta-modèles ; elle n'est autre qu'une spécification d'une chose destinée à une certaine fin. En fait, dans le domaine d'extraction d'architectures logicielles, beaucoup des recherches ont été réalisées ; cependant, malgré l'existence d'une pléthore d'approches qui proposent des processus d'extraction, aucun travail n'a été établi sur la modélisation des processus d'extraction. Par la suite, aucune des approches existantes dans la littérature n'a clarifié les concepts fondamentaux associés au processus d'extraction. Nous trouvons alors que notre travail trace un chemin particulier et nouveau dans le domaine d'extraction des architectures logicielles vu qu'il vise principalement à modéliser ces processus.

Deuxièmement, nos travaux s'intéressent notamment à l'extraction d'une architecture logicielle à partir d'un système. Nous souhaitons fournir une méthode qui permet à un architecte de construire un processus d'extraction selon ses besoins et qui prend en compte son point de vue et expertise. En fait, bien que les méthodes automatiques se sont révélées très efficaces pour l'extraction, l'intégration de l'architecte et son interaction avec le processus d'extraction restent des facteurs importants pour l'obtention d'une architecture qui le satisfait.

Présentation de la problématique

Dans l'ingénierie des logiciels, nous faisons face à l'augmentation exponentielle de la taille, la complexité et la mauvaise structuration du code source ; ce qui rend l'extraction de l'architecture un défi majeur. Malgré les travaux relatifs à l'extraction d'une architecture logicielle du système, deux axes dans l'extraction ne sont pas bien traités :

Au niveau de la modélisation des processus d'extraction d'une architecture logicielle

Plusieurs approches d'extraction d'une architecture logicielle fournissent une solution au problème d'absence d'une description d'architecture logicielle ; Chacune a ses exigences, sa méthode, son niveau d'automatisation, sa solution, ses limites et autres... Cette diversité d'approches offre alors à l'architecte une grande plage de méthodes et outils qui lui permettent d'extraire l'architecture de son système. Cependant, aucun moyen ou outil ne permet l'évaluation ou la qualification de ces processus, ce qui rend la sélection d'un processus d'extraction d'une architecture une activité difficile. En effet, les différentes approches d'extraction sont présentées à chaque fois en utilisant un style différent. Par exemple, il y a celles qui présentent l'approche en la décomposant selon les principes utilisés, d'autres selon les activités du processus et d'autres qui présentent l'approche en se focalisant sur un point particulier du processus et négligeant le reste du processus. Cette diversité de manière de présentation interdit à l'architecte de comparer lisiblement les processus d'extraction dans le but de choisir l'un d'entre eux.

Au niveau des processus d'extraction d'une architecture logicielle

Plusieurs processus d'extraction ont été conçus afin d'offrir une solution à l'absence d'une description d'architecture décrivant le système. En gros, ces processus suivent l'une des trois stratégies :

1. **Le groupement des entités du code source.** Cette stratégie consiste à analyser les entités d'un système comme les fonctions et leurs relations, puis à grouper ces entités selon certains critères pour former des éléments architecturaux à haut niveau. Le point fort d'un processus qui suit cette stratégie est son bon niveau d'automatisation et sa capacité à grouper les entités automatiquement en se basant sur des algorithmes de partitionnement (*clustering*). Néanmoins, ces processus [Mancoridis et al., 1998, Mitchell et Mancoridis, 2006, Mitchell et Mancoridis, 2008, Rajalakshmi, M, 2014, Mahdavi et al., 2003a, Mahdavi et al., 2003b, Maqbool et Babri, 2004, Maqbool et Babri, 2007, Praditwong et al., 2011, Kumari et Srinivas, 2013, Kumari et Srinivas, 2016, Qiu et al., 2015] ne prennent pas en considération les connaissances des architectes acquis de son expertise et les documentations existantes. Ce qui aboutit à une architecture nécessitant des étapes de raffinement pour qu'elle devienne conforme aux souhaits de l'architecte.

2. **La correspondance des entités logicielles à une architecture conceptuelle proposée par l'architecte.** Cette stratégie consiste à correspondre les entités du code à une architecture conceptuelle proposée par l'architecte ; celle-ci est une architecture à haut niveau qui représente les éléments architecturaux (sans les entités qui les composent). Le point fort d'un processus qui suit cette stratégie est qu'il fournit une architecture selon l'intention de l'architecte. Cependant, ces processus [Guo et al., 1999, Bowman et al., 1999, Sartipi et Kontogiannis, 2001, Sartipi et Kontogiannis, 2003, Murphy et al., 2001, Koschke et Simon, 2003] sont fortement basés sur l'expertise de l'architecte et sont quasi-manuels vu qu'ils nécessitent la définition d'un plan (d'un *map*) qui permet la correspondance entre les entités du système et les éléments architecturaux de l'architecture conceptuelle proposée par l'architecte. Ce qui interdit l'application de ces processus sur les systèmes de grandes tailles.

3. **La conciliation d'une architecture extraite avec une architecture conceptuelle proposée par l'architecte.** Cette stratégie inclut les deux stratégies précédentes. Elle consiste tout d'abord à établir une abstraction des entités logicielles en utilisant la stratégie de groupement des entités du code, puis à concilier ces abstractions à une architecture conceptuelle proposée par l'architecte. Le point fort d'un processus qui suit cette stratégie est qu'il fournit une architecture selon l'intention de l'architecte et en même temps sa capacité à grouper les entités automatiquement en se basant sur des algorithmes de partitionnement. Cependant, ces processus [Riva, 2000, Ding et Medvidovic, 2001, Medvidovic et al., 2003] souffrent des limitations suivantes : (1) La conciliation entre une l'architecture conceptuelle et l'architecture extraite n'a pas une méthode précise à suivre et n'est pas supportée par des outils. (2) Ces processus ne permettent pas à un architecte d'intégrer efficacement ses connaissances dans le processus d'extraction ; ils sont limités à l'intégration du point de vue de l'architecte sous forme d'une architecture conceptuelle. (3) Ces processus n'autorisent pas l'interaction de l'architecte avec les artefacts manipulés et la configuration du processus selon ses besoins.

Nous pensons qu'un processus d'extraction doit être construit progressivement et guidé par un architecte. Cela ne signifie pas que le processus doit être construit manuellement, au contraire un processus d'extraction efficace doit être exécuté d'une manière automatique tout en prenant en considération le point de vue de l'architecte et ses recommandations. En se basant sur ces considérations, notre objectif, dans cette thèse, est de proposer une approche qui permet l'intégration de l'architecte et lui permet de découvrir progressivement une architecture du système qui le satisfait avec un bon niveau d'automatisation.

Les contributions de la thèse

Dans le cadre de cette thèse, nous nous sommes intéressés à la problématique des processus d'extraction d'une architecture logicielle à partir d'un système existant. Nous avons étudié différentes facettes de cette problématique en proposant :

1. **Un méta-modèle pour la spécification des processus d'extraction d'une architecture logicielle**
   La proposition d'un méta-modèle dénoté SArEM (Software Architecture Extraction Meta-model) qui joue le rôle d'un outil d'analyse, d'évaluation et de comparaison des approches d'extraction d'une architecture logicielle existantes. Le méta-modèle a les caractéristiques suivantes :

   - SArEM est basé sur SPEM (Software & Systems Process Engineering Meta-Model) [OMG et Notation, 2008], qui est un méta-modèle d'ingénierie des processus logiciels ainsi qu'un modèle conceptuel qui fournit les concepts nécessaires à la modélisation des processus de développement. Alors que SPEM définit des concepts génériques pour la spécification de tout type de processus logiciel, notre méta-modèle SArEM hérite ces concepts et les étend par des nouveaux concepts spécifiques à l'extraction d'une architecture logicielle.
   - SArEM est conforme à la plupart des processus d'extraction existants surtout ceux qui sont les plus cités et qui sont considérés comme base pour le principe d'extraction. En fait, un méta-modèle qui spécifie les concepts d'un processus d'extraction d'une architecture logicielle doit couvrir les concepts généraux qui existent dans la plupart des processus d'extraction. Pour atteindre cet objectif, nous avons étudié un nombre important des processus d'extraction ; suite à cette étude, nous avons élaboré un méta-modèle contenant des concepts présents dans la plupart des processus d'extraction existants.

2. **Un modèle de processus d'extraction d'une architecture logicielle**
   La proposition d'un modèle de processus, nommé le modèle SAD *(Software Architecture Discovery)*, qui est composé de 9 étapes. Le modèle SAD est comme un processus générique qu'un architecte peut instancier afin de construire un processus particulier conforme à ses souhaits. Principalement, il permet l'intégration du point de vue d'un architecte en lui permettant de proposer une architecture conceptuelle et d'ajouter d'autres informations durant le processus. En addition, le modèle donne à l'architecte l'opportunité de configurer plusieurs paramètres durant l'exécution du processus et de raffiner tout au long du processus les résultats obtenus. Ainsi, le modèle répond aux limites des processus existants basés sur la stratégie de conciliation.

3. **Une étude des relations qui existent entre le modèle SAD et le modèle ECD**
   Nous avons étudié les relations qui existent entre le modèle SAD proposé et le modèle d'extraction de connaissances à partir de données (ECD), connu en anglais par KDD *(Knowledge Discovery in Databases)* [Fayyad et al., 1996b, Fayyad et al., 1996a]. Principalement l'étude aide à exécuter des instances (processus) du modèle SAD d'une manière automatique. En fait, ECD est un modèle de processus interactif et supporté par des outils automatiques qui permet l'extraction des nouvelles informations et connaissances à partir des données. Devant ces caractéristiques majeures d'ECD, l'interaction et le support par des outils, nous avons étudié les relations qui existent entre le modèle ECD et le modèle

SAD, et la possibilité d'exécuter des instances de notre modèle proposé SAD en utilisant les outils d'ECD.

Suite à cette étude, nous avons trouvé des écarts dans le modèle d'ECD interdisant l'exécution des processus qui suivent le modèle SAD en utilisant un outil ECD. Face à ce manque, nous avons proposé une extension pour un outil d'ECD nommé KNIME [Berthold et al., 2009] ; cette extension remédie aux limitations existantes dans KNIME et permet ainsi à un architecte d'exécuter un processus SAD comme étant un processus d'ECD. Avec une telle solution, basée sur ECD, l'architecte peut interagir avec le processus d'extraction, et découvrir progressivement une architecture qui le satisfait tout en bénéficiant des fonctionnalités classiques d'ECD.

L'organisation du document

Ce document est structuré en deux parties. La première présente les travaux connexes et la deuxième partie décrit notre proposition.

Partie 1 : État de l'art. La première partie de ce manuscrit est consacrée à l'état de l'art. Elle définit les notations d'architectures logicielles et le processus d'extraction d'une architecture logicielle. Aussi, elle présente une synthèse et une analyse des approches connexes à notre contribution. Cette partie est organisée en deux chapitres :

- Le chapitre 2 intitulé "Les travaux connexes" expose la terminologie associée aux architectures logicielles et présente également une synthèse bibliographique à propos des approches d'extraction d'une architecture logicielle. Plus spécifiquement, ce chapitre présente : (1) les définitions les plus admises sur les architectures logicielles, (2) les travaux existants sur le thème d'extraction d'une architecture logicielle. Ces travaux sont classés sous 3 catégories : les méta-modèles, les méthodes et les outils proposés dans la littérature pour l'extraction d'une architecture logicielle.
- Le chapitre 3 intitulé "Analyse comparative et limitations" présente une analyse comparative entre les différentes approches d'extraction d'une architecture logicielle afin de déterminer leurs défaillances majeures et leurs limitations. L'analyse comparative est établie en se basant sur le modèle McCall qui est un modèle de mesure de qualité pour les produits logiciels. Ce chapitre contient alors un modèle de qualité basé sur McCall, et la projection des différentes approches existantes sur le modèle.

Partie 2 : Notre contribution. La deuxième partie expose notre approche d'extraction d'une architecture logicielle à partir d'un système existant. Cette partie est organisée en trois chapitres :

- Le chapitre 4 intitulé "Le méta-modèle SArEM" présente notre méta-modèle élaboré pour la spécification des processus d'extraction d'une architecture logicielle. Le chapitre révèle tout d'abord l'importance d'un tel méta-modèle et résume le principe de SPEM sur lequel notre méta-modèle est basé. Puis, la méthodologie adoptée pour l'élaboration du SArEM est exposée. Ensuite, le cœur de notre contribution est présenté, plus spécifiquement, les éléments du SArEM sont définis. De plus, le chapitre fournit la validation de notre méta-modèle tout en présentant une étude de conformité du SArEM aux processus d'extraction d'une architecture logicielle. Finalement, une synthèse sur la qualité de notre méta-modèle est fournie.

- Le chapitre 5 intitulé "SAD : Un modèle de processus d'extraction d'une architecture logicielle" présente notre méthode pour l'extraction d'une architecture logicielle. Le chapitre expose la méthode sous forme d'un modèle d'extraction d'une architecture logicielle en termes d'activités, des artefacts et des rôles. En addition, il présente le modèle tout en utilisant une représentation graphique et un formalisme mathématique. A la fin du chapitre, une synthèse résume les apports de notre modèle par rapport à d'autres méthodes d'extraction.
- Le chapitre 6 intitulé "Une extension d'ECD pour l'extraction des architectures logicielles" présente l'extension d'un outil ECD pour supporter l'exécution d'un processus SAD. Tout d'abord, le chapitre résume le modèle de processus ECD comme il a été défini par Fayyad et al. [Fayyad et al., 1996b]. Puis, une étude des relations entre le modèle SAD et le modèle ECD est présenté ; cette étude compare les éléments des deux modèles en termes d'activités, artefacts et rôles. Ensuite, le chapitre présente l'extension de l'outil KNIME qui donne une sémantique d'extraction d'une architecture logicielle à l'outil KNIME. Finalement, les apports de l'extension par rapport à d'autres outils d'extraction sont résumés.

Enfin, nous concluons ce manuscrit par un bilan sur nos contributions, leurs limites et une présentation des perspectives d'ouverture et d'extension de notre travail.

I

État de l’art

2

Les travaux connexes

2.1 Introduction

Dans le chapitre précédent, nous avons présenté le problème que nous traitons dans cette thèse. Pour mieux comprendre ce problème, nous synthétisons dans ce chapitre les travaux les plus connus sur les architectures logicielles et la notion de composant (qui est un élément définissant l'architecture). De plus, ce chapitre expose les approches existantes sur l'extraction d'une architecture logicielle à partir d'un système. Ces différentes approches sont catégorisées selon la problématique étudiée ; ainsi, elles sont organisées en trois catégories : Les méta-modèles, les méthodes et les outils d'extraction d'une architecture logicielle.

2.2 Les architectures logicielles

Face à la croissance permanente de la taille et la complexité des systèmes, les architectures logicielles ont émergé comme un allié précieux pour la conception et la maintenance de ces systèmes. Discipline née depuis 1972 [Dijkstra, 1972], épanouie en 1990, les architectures logicielles ont été considérées la route entre 'aucune conception' et 'un design complet' de l'application. Cette liaison de grande portée définit alors une solution qui répond aux exigences techniques et opérationnelles du système.

Le mot 'architecture logicielle' a été référé la première fois en 1972. Disjkestra [Dijkstra, 1972] a été le premier qui a mentionné les architectures logicielles avec le début de la crise logicielle causé par la complexité logicielle. Partant de là et jusqu'à la fin des années 1980, le principe de l'architecture est apparu dans le sens d'une structure physique d'un système informatique. Au début des années 90, une série de recherches commence à considérer l'architecture logicielle une discipline distincte. Commençant par Winson Roya et Walker Royce [Royce et Royce, 1991] qui ont publié un article intitulé l'architecture logicielle. Puis Perry et Wolf [Perry et Wolf, 1992], Garlfan et Shaw [Garlan et Shaw, 1994] qui sont largement connus par leurs contributions fondamentales dans ce domaine. Motivés par la réduction des coûts, les travaux relatifs au domaine de l'architecture logicielle commencent à englober le concept du composant qui est caractérisé par la réutilisation. Par la suite, la notion du composant est basée sur la création d'unités logicielles réutilisables et indépendantes. L'architecture logicielle se

concentre alors sur la décomposition du système en plusieurs composants qui interagissement pour obtenir à la fin un système de haute qualité.

Vu que les composants, les connecteurs et les contraintes ne sont pas suffisants pour décrire toutes les informations concernant le mécanisme d'interaction du système, beaucoup des nouvelles définitions ont été fournies. En 2000, le premier standard de l'architecture logicielle a été proposé dans IEEE 1471-2000 [Hilliard, 2000] en enregistrant une des intuitions les plus compréhensibles dans ce domaine. De plus, parmi les travaux les plus connus dans ce domaine, on trouve [Len et al., 2003], [Bachmann et al., 2011] qui ont défini l'architecture logicielle en se basant sur celle du standard [Hilliard, 2000].

Quelques dizaines de définitions ont été proposées depuis 68 afin de déterminer une architecture logicielle. Chacune a apporté un véritable intérêt à la discipline de l'architecture logicielle. Devant cette diversité, on présente dans cette section celles les plus acceptées et les plus couramment admises. Ces différentes définitions aident à comprendre l'output final que les approches d'extraction visent à extraire.

2.2.1 La définition de Perry et Wolf

Perry et Wolf ont défini l'architecture logicielle [Perry et Wolf, 1992] comme un ensemble d'éléments architecturaux ayant une forme particulière, elle est exprimée par la formule suivante :

$$ArchitectureLogicielle = \{Elements, Forme, Raisonnement\}$$

Premièrement, les éléments architecturaux sont catégorisés en 3 classes : les éléments de données qui stockent l'information à traiter, les éléments de traitement qui agissent sur les éléments de données et les éléments de connexion qui permettent de connecter les différents éléments de l'architecture. Deuxièmement, la forme architecturale représente à la fois les contraintes sur les éléments pris unitairement et la disposition des éléments entre eux. Et troisièmement, le raisonnement désignant un ensemble des critères qui motivent l'architecte à choisir des éléments et une forme plutôt que d'autres. Ces critères peuvent être en relation avec des attributs de qualité comme la performance et le temps...

2.2.2 La définition de Garlan et Shaw

La définition de Garlan et Shaw [Garlan et Shaw, 1994] peut être résumée par la formule suivante :

$$ArchitectureLogicielle = \{Composants, Connecteurs, Contraintes\}$$

Le terme 'composants' qui signifie des entités logicielles. Le terme 'connecteurs' qui définit les interactions entre les composants telles que l'appel des procédures, et le terme 'contraintes' définissant comment les composants et les éléments peuvent être combinés. De plus, Garlan et Shaw ont mentionné que l'architecture peut être représentée en un graphe où les nœuds représentent les composants et les arcs représentent les connecteurs. Ce qui est important dans cette définition, et qui est défaillant dans la définition précédente est que Garlan et Shaw ont isolé le principe des composants de celui des connecteurs.

2.2.3 La définition de Garlan et Perry

L'architecture au sens de Garlan et Perry [Garlan et Perry, 1995] est la structure des composants d'un système, les interactions entre eux, les principes et les règles qui gouvernent la conception et l'évolution de ces composants et relations.

En mettant la lumière sur le mot 'structure', on constate que l'architecture a été considérée une structure de plusieurs composants reliés ; ce qui implique que le niveau de conception d'une architecture logicielle va au-delà de celui des données et des algorithmes, il est analogue à un niveau abstrait donnant une vue globale sur la disposition des unités du système et de leurs relations. De plus, cette définition dévoile également la nécessité des principes de conception et d'évolution dans l'architecture ; ces principes indiquent que la proposition d'une architecture doit suivre des guides formelles, des règles ou des approches.

2.2.4 La définition selon le standard IEEE 1471 :2000

Le premier standard qui a défini d'une manière formelle l'architecture logicielle et les terminologies liées à ce domaine est le standard IEEE-Std-1471-2000 *Recommended Practice for Architectural Description for Software-Intensive Systems* [Hilliard, 2000]. Le but de ce standard est d'établir un cadre formel pour le concept de l'architecture logicielle.

La norme définit le système comme un ensemble de composants organisés afin d'accomplir une ou plusieurs fonction(s). Un système n'est pas limité à des applications individuelles, mais il les englobe et couvre toutes sortes d'agrégation de logiciels qui interagissent conjointement afin de répondre à un ensemble spécifique de besoins. L'organisation de ces éléments architecturaux est notée architecture logicielle.

De plus, parmi les traces importantes de cette norme la séparation entre l'architecture logicielle et la représentation de cette architecture. Une architecture logicielle est représentée ou décrite en utilisant une description architecturale (notée AD). Cette dernière est un artefact qui documente l'architecture d'une manière que les intervenants du système peuvent comprendre l'architecture et démontrer qu'elle a rencontré leurs préoccupations. Avec cette disjonction des 2 principes, la norme a insisté que chaque système a une architecture, mais il ne s'ensuit pas nécessairement qu'elle est documentée.

2.2.5 La définition de Len et al.

Selon Len et al. [Len et al., 2003], l'architecture logicielle d'un système est la ou les structure(s) du système constituée(s) des éléments logiciels, des propriétés extérieurement visibles de ces éléments et des relations entre eux. Cette définition insiste sur les propriétés extérieurement visibles des éléments constituant le système, et souligne aussi qu'un système peut avoir une ou plusieurs structures. Dans ce qui suit on met le doigt alors sur les terminologies utilisées fortement dans cette définition ; ces terminologies sont : la structure, les éléments logiciels et les propriétés du système.

Premièrement, d'après la définition, il est clair qu'un système a une ou plusieurs structures dont chacune incarne des informations à propos des éléments et leurs interactions. Bass et al. explique que chaque structure donne une manière d'interaction des éléments du système ; Par exemple la structure peut être statique, en ce sens qu'elle met l'accent sur la façon dont les éléments sont reliés pour fournir les fonctionnalités du système. D'autres structures sont beaucoup plus axées sur la manière dont les éléments interagissent les uns avec les autres à l'exécution pour exécuter la fonction du système. Dans le livre [Rozanski et Woods, 2012], Rozanski et Woods ont plus précisé la structure en dévoilant qu'il existe 2 types de structure : (1) La structure statique qui définit les éléments du système et les relations entre eux. Et (2) La structure dynamique qui définit les éléments du système et les interactions entre eux au moment de l'exécution.

Deuxièmement, les éléments logiciels, aussi nommés les éléments architecturaux ou tout simplement éléments, sont les parties qui constituent le système. Par exemple, un élément peut

être une librairie, un composant, un processus, une base de données .... La définition souligne que l'architecture omet spécifiquement certaines informations sur des éléments (qui ne concernent pas leurs interactions). Un élément interagit avec d'autres éléments au moyen d'interfaces ; ces derniers séparent les détails d'un élément en parties publiques et privées. L'architecture concerne le côté publique de cette division tandis que les détails privés des éléments ne sont pas architecturaux.

Troisièmement, les propriétés extérieurement visibles d'un système sont les comportements des éléments. Un comportement d'un élément peut être observé ou discerné du point de vue d'un autre élément, c.à.d. en traitant le système comme étant une boite noire, et ce que les autres éléments peuvent observer caractérise cette propriété.

2.2.6 La définition de Bachmann et al.

Bachmann et al. [Bachmann et al., 2011] ont défini l'architecture comme l'ensemble des structures nécessaires pour raisonner du système, comprenant des éléments logiciels, les relations entre eux, et les propriétés des deux.

La définition met l'accent une fois encore sur la pluralité de structures dans chaque système logiciel qui sont la clé de la réalisation du système ; les structures sont constituées d'éléments, des relations entre eux et les propriétés importantes des deux. Par cette définition, Bachmann précise que pour la compréhension d'un système, de ses éléments et leurs interactions, il suffit de comprendre les structures principales qui reflètent le système, en d'autres termes comprendre son architecture.

2.2.7 La définition selon le standard ISO/IEC/IEEE 42010 :2011

Le standard ISO/IEC/IEEE 42010 :2011 [ISO, 2011] est une nouvelle version du standard IEEE-Std-1471-2000 présenté dans la section 2.2.4 ; il est considéré l'une des contributions les plus modernes dans le domaine des architectures logicielles. Selon le standard, l'architecture est définie comme étant les concepts ou propriétés fondamentaux d'un système dans son environnement incarné dans ses éléments, ses relations et les principes de son conception et évolution.

Tout d'abord, le standard a donné une grande importance à l'architecture en la considérant comme tout ce qui est fondamental (essentiel) pour le système ; ces choses fondamentales peuvent prendre plusieurs formes comme les éléments, leurs relations, et les principes de conception et évolution. Deuxièmement, les termes 'concepts' et 'propriétés' soulignent que l'architecture est une conception et une perception des propriétés du système. De plus, le standard a insisté via ces termes que l'architecture est abstraite et que chaque système a une architecture même si cette architecture n'est pas décrite. Et troisièmement, concernant l'environnement, chaque système est incarné dans son environnement d'une manière que le système agit sur cet environnement et vice versa. Ainsi, l'environnement détermine la totalité des influences sur le système.

En addition, le standard a défini la description architecturale (AD) comme un artefact qui exprime une architecture. Les architectes et les autres acteurs du système utilisent la description architecturale pour comprendre, analyser et comparer les architectures, et souvent comme "plans" pour la planification et la construction. Le méta-modèle présenté dans la figure 2.1 capte les termes et les concepts du système, son architecture et la description architecturale.

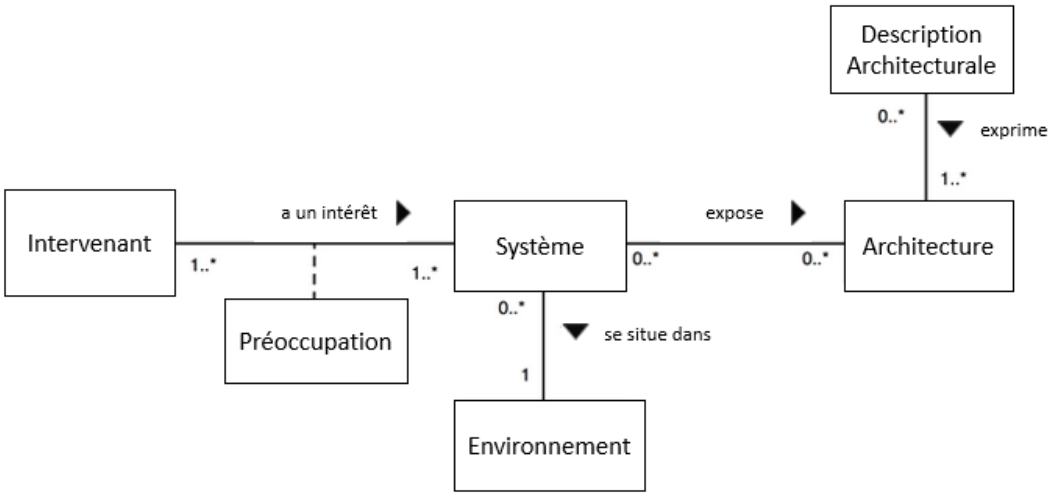

FIGURE 2.1 : Les termes et les concepts liés à l'architecture logicielle et l'AD : extrait de [ISO, 2011].

2.3 Les composants

La plupart des définitions citées ci-dessus focalise sur la notion de composants pour déterminer une architecture logicielle. Ainsi, nous présentons dans ce qui suit les principales contributions qui englobent le principe d'un composant surtout celles qui mettent en valeur ses propriétés.

2.3.1 Le composant selon Szyperski

Selon Szyperski [Szyperski, 2002], un composant est une unité de composition ayant des interfaces et des dépendances explicites avec le contexte. Il peut être déployé indépendamment et composable par des autres composants.

Cette définition englobe plusieurs termes dont chacun d'eux énonce explicitement les caractéristiques d'un composant. En effet, être une unité de composition signifie que l'unité est capable de se composer avec d'autres éléments. Cette collaboration entre les unités s'établit grâce à des interfaces pour chaque composant. De même, Szyperski souligne qu'un composant dépend d'un contexte, par exemple une connexion à la base de données. Cependant, même en présence d'une telle dépendance, un composant ne cesse pas d'être indépendant ; une modification dans l'implémentation d'un composant ne doit pas provoquer des changements dans les autres composants. De plus, cette définition souligne que le composant est sujet de composition, c.à.d. le composant peut être créé par un côté et réutilisé par un autre.

2.3.2 Le composant selon Lüer et Van Der Hoek

Lüer et Van Der Hoek [Lüer et Van Der Hoek, 2002] définissent un composant comme un élément logiciel qui (a) encapsule une implémentation réutilisable d'une fonctionnalité, (b) peut être composé sans modification et (c) adhère à un modèle de composant. Cette définition, qui souligne les mêmes caractéristiques d'un composant selon la définition de Councill [Councill et Heineman, 2001], précise qu'un composant encapsule, ou cache l'implémentation d'une fonctionnalité ; c'est comme si le composant est une boite qui fournit un service d'une manière opaque, favorisant ainsi sa réutilisation. De plus, comme la définition précédente, un composant possède la propriété de composition. D'autre part, Lüer et al. mentionnent en gros, qu'un

modèle de composant est nécessaire pour se mettre d'accord sur une norme permettant la collaboration entre les composants comme une norme pour spécifier les paramètres de passages entre les composants.

2.3.3 Le composant selon le standard de modélisation UML

Dans [Specification, 2007], un composant représente une partie modulaire du système qui encapsule son contenu et dont la manifestation est remplaçable dans son environnement.

Dans cette définition, une partie modulaire signifie que le composant représente un bloc de construction du système *building block*. De plus, la définition révèle le concept d'encapsulation et d'indépendance. Notons que le standard a clarifié encore que le comportement d'un composant est défini par le biais des interfaces fournies et des interfaces requises. Les interfaces fournies sont celles exposées par le composant à son environnement. Les interfaces requises d'un composant sont les interfaces que doivent avoir les autres composants de l'environnement afin que le composant puisse fournir ses fonctionnalités.

2.4 L'extraction des architectures logicielles

L'extraction d'une architecture logicielle est considérée comme le processus d'analyse de plusieurs inputs tels que le code source du système et les documentations existantes afin d'identifier une représentation architecturale du système. En fait, de nombreuses définitions ont été proposées pour décrire l'extraction d'une architecture logicielle à partir d'un système existant ; cette diversité de définitions a été causée par le manque d'un concept unique pour l'architecture logicielle d'une part et la spécification du but de l'extraction d'autre part.

Chikofsky [Chikofsky et Cross, 1990] a défini la rétro-ingénierie, qui est un domaine englobant l'extraction d'une architecture logicielle, la ré-documentation et l'exploration du code, comme étant un processus d'analyse d'un système pour identifier ses composants et leurs relations et créer des représentations du système sous une autre forme ou à un niveau d'abstraction supérieur. Krikhaar [Krikhaar, 1997] a suivi cette définition et a ajouté que l'extraction concerne toute activité qui rend explicite l'existence d'une architecture logicielle du système. Jazayeri et al. [Jazayeri et al., 2000] décrivent l'extraction en tant que techniques et processus utilisés pour découvrir l'architecture d'un système à partir des informations disponibles. Pour Riva [Riva, 2000], l'ingénierie inverse concerne la tâche de récupérer les décisions de conception que les développeurs ont pris lors du développement du système. Ainsi, nous résumons l'extraction d'une architecture comme un processus visant à identifier une architecture du système courant et générer une représentation de cette architecture. Cette représentation aide l'architecte, les concepteurs et développeurs à comprendre la structure de leurs systèmes.

Nous avons classifié les approches liées à l'extraction d'une architecture logicielle en 3 catégories, selon la problématique étudiée.

- Les méta-modèles des processus d'extraction d'une architecture logicielle. Cette catégorie englobe toute cadre *(framework)* ou modèle conceptuel visant à mettre un cadre formel pour l'extraction d'une architecture logicielle.

- Les méthodes d'extraction d'une architecture logicielle. Cette catégorie inclut toute méthode, méthodologie, manière, approche ou processus visant à extraire une architecture logicielle à partir d'un système existant.

- Les outils d'extraction d'une architecture logicielle. Cette catégorie englobe tout outil ou technique visant à supporter une méthode ou même une étape de la méthode d'extraction d'une architecture logicielle.

2.5 Les méta-modèles des processus d'extraction d'une architecture logicielle

Un méta-modèle des processus d'extraction d'une architecture logicielle détaille les concepts utilisés dans les processus d'extraction, spécifie ce que doit être mis dans un processus d'extraction. Partant de l'objectif d'un méta-modèle, nous classifions les travaux sous deux catégories : Les modèles conceptuels et les états de l'art.

2.5.1 Les modèles conceptuels des processus d'extraction d'une architecture logicielle

Dans cette section nous présentons les modèles conceptuels ; ces derniers visent à détailler les niveaux d'abstractions sur lesquels opèrent un processus d'extraction. Donc, ces modèles définissent indirectement des concepts liés aux processus d'extraction.

Le modèle conceptuel du fer à cheval *(Horseshoe)*

En se basant sur le travail de Woods et al. [Woods et al., 1998], kazman et al. [Kazman et al., 1998] proposent un modèle générique, appelé Horseshoe, pour l'intégration des outils de la ré-ingénierie. Cette dernière consiste à reconstruire le système selon une nouvelle architecture, elle inclut l'extraction d'une architecture du système à partir du code source, la transformation de l'architecture extraite vers une nouvelle architecture désirée et la ré-ingénierie du système selon la nouvelle architecture. Ainsi selon ce modèle, la ré-ingénierie consiste de 3 processus qui opèrent sur 4 niveaux de la représentation d'un système. Le premier processus consiste à passer du niveau inférieur de la représentation du système (qui est le niveau du code source) à un niveau supérieur de la représentation du système (qui est le niveau architectural). Le deuxième processus consiste à passer d'une représentation architecturale à une autre, qui est la représentation architecturale désirée. Et le troisième processus consiste à passer de l'architecture au niveau inférieur. La figure 2.2 illustre ce modèle (les 3 processus et les 4 niveaux sur lesquels ils opèrent). Ainsi, ce modèle n'est pas limité à l'extraction, mais il englobe le principe de la ré-ingénierie dont l'extraction est une phase (le premier processus). Les quatre niveaux de la représentation d'un système définis par le modèle de Horseshoe sont les suivants :

- La représentation du texte source : une représentation textuelle du système ; cette représentation est le code source du système.
- La représentation de la structure du code : une représentation syntaxique qui est plus formelle que la représentation précédente ; cette représentation inclut l'arbre de syntaxe, le contrôle et le flux des données...
- La représentation au niveau fonctionnel : une représentation des relations entre les fonctions, les données et les fichiers.
- La représentation architecturale : une représentation des éléments architecturaux (comme les composants) et leurs connecteurs. Plus spécifiquement, à ce niveau les fonctions, les

données, les fichiers qui constituent la représentation précédente sont assemblés en des composants.

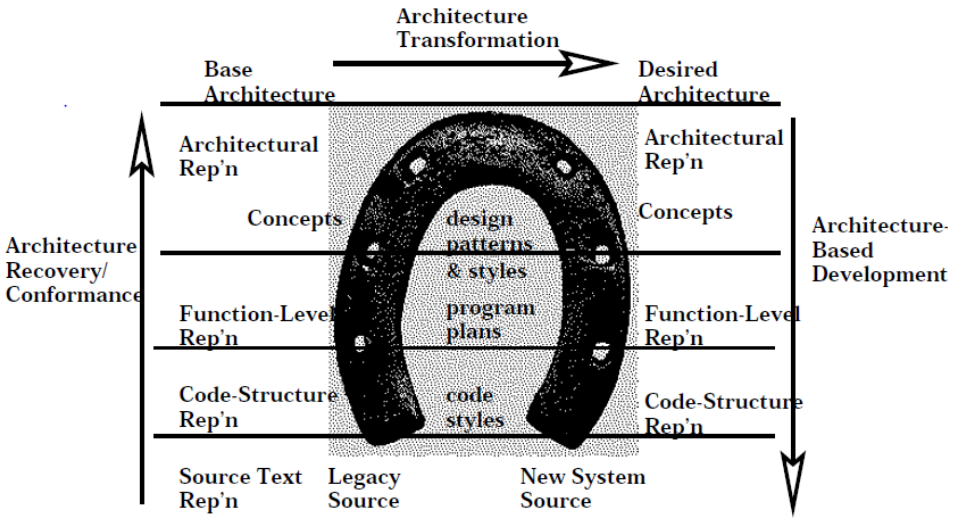

FIGURE 2.2 : Le modèle conceptuel du fer à cheval : extrait du [Kazman et al., 1998].

Le modèle conceptuel proposé par Koschke

Koschke [Koschke, 2000] propose une framework pour unifier, classifier et comparer les techniques d'extraction. Koschke a trouvé une lacune dans le modèle proposé par Kazman et al. en considérant le modèle d'une perspective architecturale. Il a amélioré alors le modèle de Horseshoe pour créer un modèle formé de 2 parties contenant 2 niveaux chacune.

- La première partie est au niveau du code source : Selon Koschke une représentation de l'architecture supprime généralement des déclarations et des expressions locales présentées au niveau du code, mais, les déclarations globales des constantes, des variables, des fonctions et des types définies par l'utilisateur doivent exister dans une telle représentation. Pour cela, Koschke a créé un niveau inférieur du code source englobant les expressions et les instructions locales et un autre niveau supérieur du code source contenant les entités globales du code source.

- La deuxième partie est au niveau architectural. Elle est composée de 2 niveaux. Le premier est le niveau inférieur de l'architecture où les entités du niveau supérieur du code source sont regroupées. Ce niveau-là constitue selon Koschke le niveau intermédiaire entre le niveau du code source et le niveau architectural supérieur puisqu'il donne des informations supplémentaires à celle présentées dans le code source et aide également à donner l'architecture en termes des éléments architecturaux. El le dernier niveau, qui est le niveau architectural supérieur, représente l'architecture du système en termes de sous-systèmes et de connecteurs.

  Koschke, comme Kazman, a défini alors un modèle conceptuel de l'extraction composé de 4 niveaux. Selon ces modèles, le processus d'extraction est considéré une démarche qui débute au niveau du code source et se termine au niveau de l'architecture logicielle du système.

Le modèle conceptuel proposé par Chardigny

Chardigny [Chardigny, 2009] propose un modèle basé sur 4 niveaux conceptuels : (1) le niveau du code source qui est le premier niveau selon le modèle de Kazman et l'union des deux niveaux du code source selon le modèle de Koschke ; (2) le niveau du modèle du code source. A ce niveau, le code source est représenté dans un modèle où les entités sont des éléments du code source ; (3) le niveau de correspondance qui comprend les groupes des entités du niveau modèle du code source ; ce niveau correspond au niveau architectural inférieur de Koschke ; et (4) le niveau architectural qui contient les éléments architecturaux (les composants et les connecteurs). Il correspond aux derniers niveaux des deux modèles précédents.

Selon Chardigny, les modèles précédents décrivent uniquement les processus qui passe d'un niveau inférieur vers un niveau supérieur, alors qu'il existe des processus qui utilisent une approche hybride, partant à la fois du niveau haut et bas. Ainsi, Chardigny a complété son modèle conceptuel en décrivant deux chemins de passage entre les niveaux : Le premier chemin consiste à passer du niveau le plus bas vers le niveau architectural. Et le second chemin, commence du niveau du code source, puis le niveau du modèle source et le niveau architectural et enfin le niveau de correspondance. Ainsi, le premier chemin décrit les processus d'extraction qui suivent la stratégie du groupement des entités du code source, et le deuxième chemin ceux qui suivent la stratégie de conciliation entre une architecture extraite et une architecture conceptuelle.

Les niveaux d'abstraction proposés par Riva

Claudio Riva [Riva, 2000] distingue 5 niveaux d'abstraction qui décrivent le principe de la rétro-ingénierie. Les niveaux sont classés sous 2 domaines : le domaine du problème qui met en évidence les propriétés fonctionnelles et non fonctionnelles du système, il englobe le niveau des besoins (*Requirements*) et celui des fonctionnalités (*Features*) ; et le domaine de la solution qui spécifie comment les tâches doivent être réalisées, il englobe ainsi les niveaux qui expriment comment le problème doit être résolu.

- Les niveaux appartenant au domaine du problème : Le premier niveau d'abstraction est *Requirements* qui exprime les besoins fonctionnels et non fonctionnels du système. Le second est *Features* où les besoins fonctionnels sont groupés à des fonctionnalités du système, par exemple, pour un système d'un mobile : *speed dials* et *call lists* sont des fonctionnalités. Riva considère ce niveau comme étant le point de contact entre le problème et le domaine de la solution.

- Les niveaux appartenant au domaine de la solution : Le niveau de l'architecture logicielle (qui est le troisième niveau) capte les décisions prises pour faire l'implémentation des besoins fonctionnels et satisfaire les paramètres de qualité du système. A ce niveau, le système est représenté sous forme des composants et des relations entre eux. Le quatrième niveau est le *Design* qui précise comment les composants satisfont leurs spécifications architecturales, par exemple le composant est spécifié en termes des classes orientées objets, la base de données en termes d'entités et des relations. Le dernier niveau concerne l'implémentation représentant ainsi le code source.

Donc, selon les niveaux d'abstraction de Claudio Riva, un processus d'extraction d'une architecture logicielle prend le sens opposé de l'ingénierie ; il commence au niveau de l'implémentation, puis le design et enfin le niveau de l'architecture logicielle.

2.5.2 Les états de l'art sur les processus d'extraction d'une architecture logicielle

Cette section présente les états de l'art visant à comparer les approches d'extraction de point de vue d'un architecte. Ces travaux aident indirectement un architecte à choisir un processus d'extraction parmi une plage d'approches d'extraction. Ainsi, nous avons classifié les états de l'art sur les processus d'extraction sous la catégorie des méta-modèles pour deux raisons : Premièrement, les cadres de comparaison fournissent certains concepts pour le principe d'extraction d'une architecture logicielle. Et deuxièmement, la comparaison de certaines approches aide l'architecte à choisir une approche selon certains critères, ce qui est l'un des objectifs d'un méta-modèle.

L'état de l'art de Ducasse et Pollet

Ducasse et Pollet [Pollet et al., 2007, Ducasse et Pollet, 2009] présentent un état de l'art sur les processus de reconstruction d'une architecture logicielle. Leur travail considère plusieurs approches d'extraction et les projette sur certains critères. Précisément, le cadre de comparaison est composé de 5 axes : L'objectif du processus de reconstruction, son type, ses inputs, ses outputs et les techniques utilisés.

Le premier axe est l'objectif du processus de reconstruction, ce qui est le but de la l'architecture reconstruite. Pour chaque approche, Ducasse et Pollet précisent si l'objectif de l'output final est la ré-documentation, la réutilisation, la migration, la co-évolution, l'analyse, l'evolution ou encore l'obtention des artefacts qui fournissent des informations si précieuses aux ingénieurs. Le deuxième axe est celui du type du processus d'extraction. Un processus peut être l'un des trois types : ascendant *(bottom-up)*, descendant *(top-down)* ou hybride. Les processus ascendants commencent par des connaissances de bas niveau pour récupérer l'architecture. À partir du code source, ces processus augmentent progressivement le niveau d'abstraction pour arriver à une architecture. Tandis que les processus descendants commencent par des connaissances de haut niveau telles que les besoins fonctionnels du système ou les styles architecturaux et visent à découvrir l'architecture en formulant des hypothèses conceptuelles et en les associant au code source. Les processus hybrides combinent le principe des processus ascendants avec celui des descendants : D'une part, les connaissances de bas niveau sont abstraites en utilisant diverses techniques et d'autre part, les connaissances de haut niveau sont raffinées et confrontées aux connaissances abstraites. Le troisième axe est l'input. Selon Ducasse et Pollet la plupart des approches de reconstruction s'appuient sur le code source et l'expertise humaine ; d'autres exploitent des sources d'information architecturales comme les styles architecturaux et les points de vue. Le quatrième axe est celui de l'output. Ce dernier peut être des vues architecturales, une architecture ou des informations sur la conformité de l'architecture et l'implémentation. Et le dernier axe concerne les techniques : les techniques sont classifiées selon leur niveau d'automatisation.

Ainsi, Ducasse et Pollet ont fourni une comparaison de certaines approches, ce qui aide un architecte à choisir, selon un nombre de critères, parmi les approches prises en considération.

L'état de l'art de De Silva et Balasubramaniam

Dans [De Silva et Balasubramaniam, 2012], les auteurs ont classifié les travaux qui visent à contrôler le problème d'érosion entre la représentation architecturale et l'implémentation. Ils ont construit leur étude en classant principalement les approches en trois catégories qui tentent successivement de minimiser, prévenir et réparer l'érosion de l'architecture. Chacune de ces catégories est correspondue à une stratégie pour lutter contre l'érosion. Nous focalisons dans ce

qui suit sur les stratégies des approches de réparation vu que ces stratégies sont liées au principe d'extraction d'une architecture logicielle.

Les approches visant à réparer l'érosion entre la representation architecturale et l'implémentation adoptent une stratégie de restauration de l'architecture. Cette stratégie comprend 3 sous-stratégies : La récupération de l'architecture, la découverte de l'architecture et la réconciliation de l'architecture. (1) La stratégie de récupération consiste à extraire une architecture à partir du code source et d'autres artefacts. (2) La stratégie de la découverte d'une architecture consiste à construire un modèle de l'architecture prévue, puis à extraire l'architecture du code selon celle prévue. (3) La réconciliation d'architecture consiste à faire correspondre l'implémentation avec une architecture prévue. Cette stratégie inclut souvent la stratégie de récupération et celle de découverte d'une architecture.

Ainsi, le travail de De Silva et Balasubramaniam a catégorisé les approches selon la stratégie adoptée. Cette classification facilite à un architecte le choix d'une approche parmi les trois catégories.

2.6 Les méthodes d'extraction d'une architecture logicielle

Généralement, une méthode est définie comme une forme particulière de procédure pour accomplir ou approcher quelque chose. Une méthode d'extraction d'une architecture logicielle peut être considérée comme un processus ayant pour but d'extraire une architecture du système. Se basant sur la problématique énoncée dans l'introduction de ce manuscrit, nous avons classifié les méthodes en 3 catégories :

- Les méthodes basées sur la stratégie de groupement des entités du code source.
- Les méthodes basées sur la stratégie de correspondance des entités logicielles à une architecture conceptuelle.
- Les méthodes basées sur la stratégie de conciliation d'une architecture extraite avec une architecture conceptuelle.

2.6.1 Les méthodes basées sur la stratégie du groupement des entités du code source

Les méthodes fondées sur le groupement des entités du code source tentent d'analyser un système donné en regroupant ses entités selon certains critères. Généralement, les critères sont les suivants :(1) Avoir une cohésion forte entre les entités d'un même groupe et (2) Avoir un couplage faible entre les entités de deux groupes différents. Ainsi, ces méthodes donnent comme résultat des groupes d'entités ayant une faible dépendance entre eux ; chacun de ces groupes représente un élément architectural. Ainsi, l'architecture extraite est en termes d'éléments architecturaux et les entités qui les composent.

La méthode d'extraction proposée par Mancoridis et al.

Mancoridis est le premier qui a introduit le problème d'extraction d'une architecture logicielle comme étant un problème de partitionnent des modules où les algorithmes d'optimisation (comme les algorithmes de Hill climbing, du recuit simulé et les algorithmes génétiques) peuvent être appliqués [Mancoridis et al., 1998, Mancoridis et al., 1999, Mitchell et al., 2001, Mitchell et Mancoridis, 2006, Mitchell et Mancoridis, 2008, Doval et al., 1999]. Cette idée a abouti à l'élaboration d'un outil appelé Bunch pour le regroupement des modules (la

section 2.7.2 souligne les concepts de cet outil). Dans cette section, nous détaillons les étapes à suivre pour identifier une architecture à partir du code source.

La première étape consiste à extraire un graphe, appelé MDG (Module Dependency Graph), qui représente les dépendances des modules. Un module peut être une fonction, un fichier ou une classe et une dépendance entre deux modules peut être un appel de fonctions ou une relation entre deux classes. Cette étape est réalisée grâce à des outils automatiques (analyseurs du code source [Chen et al., 1997, Holt et al., 2000b, Mancoridis et al., 1994]) qui prennent comme input le code source et fournissent comme output une représentation graphique des dépendances des modules (MDG). Les nœuds représentent les modules et les arcs représentent les dépendances entre eux.

La deuxième étape consiste à exécuter un algorithme de partitionnement supporté par l'outil Bunch comme l'algorithme de Hill Climbing ou l'algorithme génétique. L'outil cherche une partition des nœuds du graphe MDG où dans chaque partie de la partition il y a une forte cohésion et entre les parties un couplage faible. Par exemple, la figure 2.3 est un graphe MDG qui représente 5 modules et leurs relations, et la figure 2.4 montre le résultat du partitionnement du graphe ; les modules sont groupés en deux parties ayant un faible couplage entre leurs modules et une forte cohésion à l'intérieur de chacune.

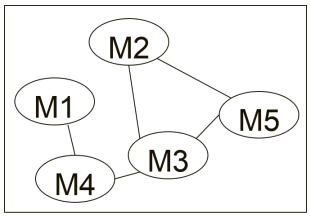

FIGURE 2.3 : Un graphe MDG représentant 5 modules d'un système.

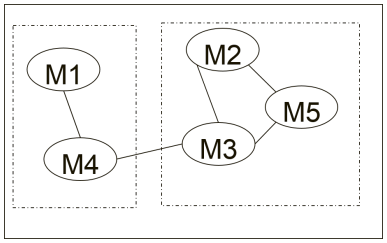

FIGURE 2.4 : Un partitionnement du graphe MDG.

Le processus proposé par Mahdavi

Mahdavi et al. [Mahdavi et al., 2003a] proposent un processus d'extraction basé sur un groupement multiple des entités du code source. Le processus est basé sur l'approche de Mancoridis [Mancoridis et al., 1999].

La première étape est l'exécution multiple de l'algorithme de partitionnement Hill climbing en se servant de l'outil Bunch. Chaque exécution fournit une architecture en termes de groupes d'entités. Le résultat de ces multiples exécutions est un ensemble de solutions (architecture). De cet ensemble, les meilleures solutions sont choisies.

La deuxième étape sert à créer des *building blocks*. Ces derniers sont des groupes d'entités présentes dans la plupart des solutions de l'étape précédente. Donc, cette étape prend comme input l'ensemble des solutions de la première étape, puis en comparant les parties des partitions, les *Building blocks* sont identifiés. En d'autres termes, les parties des partitions semblables représentent les 'building blocks' vu qu'elles ont vérifié après plusieurs exécutions de l'algorithme Hill climbing que les entités appartiennent à une même partie.

La troisième étape consiste à exécuter l'algorithme de Hill climbing de nouveaux avec les *building blocks*. Dans cette exécution, chaque building block identifié dans l'étape précédente fournit un nœud. De plus, les autres entités, n'appartenant à un *building block* précis, sont représentés chacun par un nœud. Dans cette exécution basée sur les *building blocks*, l'appartenance de certaines entités à une même partie est assurée.

La méthode IGA

Dans cette approche [Rajalakshmi, M, 2014] nommée IGA, certaines connaissances des développeurs sont intégrées dans le processus d'extraction. IGA est une variante de GA (algorithme génétique) qui utilise l'évaluation humaine. L'idée d'IGA est d'ajouter itérativement des contraintes sur l'algorithme génétique, favorisant ainsi l'approchement à une solution de bonne qualité. Cette idée est accomplie en exécutant l'algorithme génétique à plusieurs reprises tout en ajoutant à chaque itération de nouvelles contraintes.

Le processus a pour objectif de faire un partitionnement pour les modules du système tout en prenant en considération les propriétés de la cohésion forte et le couplage faible. Vu que les outils automatiques ne sont pas capables de donner une solution qui satisfait totalement les développeurs, une intégration d'un feedback donné par les utilisateurs paraît une étape importante durant le partitionnement. Pour cela Rajalakshmi propose 2 stratégies qui peuvent être appliquées pour aider les développeurs à intégrer leurs connaissances : La première stratégie consiste à sélectionner 2 composants(modules) et de les montrer aux développeurs. Ces derniers décident s'ils doivent être regroupés dans une même partie ou non. La deuxième stratégie prend en considération les parties isolés, c.à.d. les parties qui sont composées d'un seul module. Ces parties isolées sont montrées aux développeurs pour décider dans quelle partie doivent être placées.

Ainsi, les étapes du processus sont itératives. A chaque itération, l'algorithme génétique supporté par l'outil Bunch est exécuté. Puis, une intégration des développeurs est établie en fournissant un feedback selon l'une des stratégies proposées. A chaque itération, l'algorithme génétique est exécuté tout en prenant en considération le feedback des développeurs. Le processus s'arrête quand la solution satisfait les développeurs.

2.6.2 Les méthodes basées sur le principe des architectures conceptuelles

Les méthodes basées sur les architectures conceptuelles proposées font correspondre les entités du code source à une architecture conceptuelle proposée par l'architecte. L'architecture conceptuelle est une representation abstraite du système ; elle le représente en termes des blocs et les relations entre eux.

Le processus proposé par Bowman et al.

Cette approche [Bowman et al., 1999] est proposée en prenant Linux comme un cas d'étude parce qu'il est un système complexe et représente la plupart des systèmes existants d'une part, et parce qu'il existe plusieurs documentations sur l'architecture conceptuelle de ce système d'autre part. Ces documentations existantes donnent des détails sur les sous-systèmes du Linux mais pas sur les relations entre ces différents sous-systèmes. On présente ci-dessous le processus de l'extraction qui nécessite quelques documentations pour l'étape initiale.

La consultation des documents existants est nécessaire pour que l'architecte puisse proposer une architecture conceptuelle. Cette architecture contient les sous-systèmes (sans les entités qui les composent) et les relations entre eux. Or le système pris comme cas d'étude est un système complexe, l'architecte propose des sous-architectures conceptuelles pour chaque sous-système en se basant aussi sur les documents.

Afin d'avoir une architecture concrète, il faut examiner l'artefact définitif qui est le code source. L'outil *cfx* est utilisé pour extraire les fonctions et les relations entre eux et quels fichiers définissent les différentes fonctions. Donc grâce à cet outil, les dépendances entre les données et les fonctions sont extraites. De même, l'outil *grok* est utilisé pour déterminer les

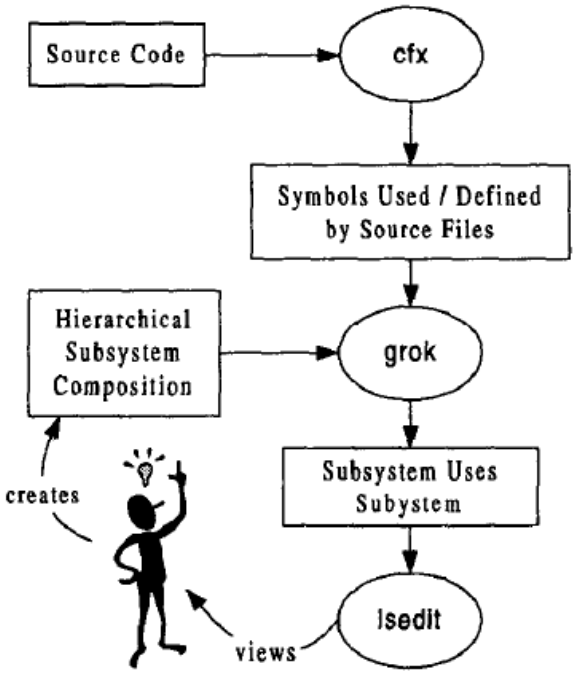

FIGURE 2.5 : Le processus de l'extraction proposé par Bowman : extrait du [Bowman et al., 1999].

relations entre les fichiers sources. Ces relations sont identifiées selon les fonctions et les variables définies dans les fichiers sources. En fait, le but est de définir les relations entre les sous-systèmes et pas entre les fichiers. Pour achever ce résultat, l'architecture conceptuelle est utilisée comme une structure initiale et les fichiers du code source sont assignés dans les sous-systèmes en se basant sur plusieurs critères : La structure des répertoires, les noms des fichiers, les commentaires du code source, etc. De nouveau *grok* est utilisé pour identifier les relations entre les sous-systèmes selon les relations entre les fichiers sources. Finalement, la décomposition hiérarchique du système, qui comporte des sous-systèmes, leurs relations et les fichiers qui les composent, est raffinée en utilisant l'outil *lsedit* pour la visualisation et tout en déplaçant les fichiers sources d'un sous-système à un autre.

La méthode de reconstruction ARM

ARM [Guo et al., 1999] est une méthode d'analyse semi-automatique pour reconstruire l'architecture des systèmes à base des patrons. La méthode a pour but de reconnaitre les patrons architecturaux du système (comme MVC, médiateur, etc. [Gamma, 1995]) afin de passer des informations au niveau du code source à un niveau d'abstraction plus élevée qui représente les patrons existants dans l'implémentation. ARM est un processus itératif qui nécessite (1) l'intégration humaine pour analyser les résultats et diriger le processus ; (2) la présence de quelques documents pour aider l'architecte à mettre une première description des patrons ; et (3) l'utilisation de l'infrastructure Dali [Kazman et Carrière, 1999] qui offre une variété d'outils d'extraction, de manipulation, d'analyse et de visualisation. Le processus RAM est formé de 4 phases qui ont pour but (1) d'utiliser les compétences de l'architecte pour développer un plan de reconnaissance des patrons, (2) d'extraire un modèle source qui représente le code, (3) d'utiliser des outils pour appliquer le plan sur le modèle source, et (4) d'analyser les résultats en utilisant des outils de visualisation.

1. Le développement d'un plan pour la reconnaissance des patrons. Cette phase contient 3 étapes : La première est l'instanciation d'une description d'un patron. L'architecte détermine une description abstraite des patrons utilisés et extrait les règles abstraites des patrons, en se basant sur les documents existants. Cette description est comme une architecture conceptuelle, elle définit quels sont les composants (blocs) à extraire et leurs relations. Afin de rendre la description concrète, l'architecte consulte le code source. La reconnaissance concrète des règles des patrons peut être faite via des signaux synthétiques comme les conventions de nommage, les mots-clés des langages de programmation, ou une analyse de l'accès aux données et du flux de contrôle. Le but de cette étape est d'avoir comme output une description concrète du patron. Cette dernière est une spécification des règles concrètes dans RSF(Rigi Standard Format). L'architecte décrit alors les règles concrètes du patron en utilisant RSF [Müller et Klashinsky, 1988] ; une clause dans RSF est un triplet (relation, entité 1, entité 2). L'output de cette étape est la liste des triplets qui expriment la description concrète des patrons. Après cette étape, la description de l'étape précédente est transformée en des requêtes. Dernièrement, l'architecte met un plan de reconnaissance des composants. C.à.d. il identifie quel composant doit être reconnu au premier, et la séquence des autres composants.

2. L'extraction d'un modèle source qui représente les éléments et les relations entre eux. L'output de cette phase est un modèle source contenant des informations nécessaires à l'étape suivante comme les différentes relations entre les classes et les fonctions... Cette étape est faite grâce aux outils de l'infrastructure *Dali*.

3. La détection et l'évaluation des instances des patrons. La détection des instances est un processus automatique dans l'infrastructure *Dali* où les requêtes sont exécutées selon le plan (output de l'étape 1) sur le modèle source extrait (output de l'étape 2). Alors tous les candidats des instances des patrons sont fournis par *Dali*. L'architecte doit évaluer les candidats afin de déterminer quel patron est détecté mais n'est pas désigné comme une instance et quel candidat est désigné comme une instance mais il n'est pas détecté. Pour améliorer le résultat, l'architecte peut modifier le plan en réordonnant les requêtes ou l'ordre des composants à détecter. Cette étape se termine quand *Dali* détecte un nombre maximal des instances de patrons vraies et avec la capacité de l'architecte à expliquer les instances fausses.

4. La quatrième phase consiste à analyser et reconstruire l'architecture. L'architecte organise les autres éléments du modèle source autour des instances détectées et fait une comparaison entre l'architecture et l'implémentation.

Le processus Reflexion Model

Le processus ReflexionModel [Murphy et al., 1995, Murphy et Notkin, 1997, Murphy et al., 2001] consiste à correspondre les entités du code source à un modèle de haut niveau proposé par l'architecte. Cette correspondance est établie grâce à un plan (*map*) défini par l'architecte. Ce plan n'est qu'une définition de certains règles concernant les noms des entités du code source et les blocs du modèle à haut niveau. Par exemple, une règle peut être : tous les fichiers sources contenant le nom 'Pager' sont correspondu au bloc ayant le nom 'Page' du modèle de haut niveau. Plus précisément, le processus est composé des étapes suivantes : la définition d'une architecture de haut niveau, l'extraction d'un modèle source à partir du code source, la définition d'un map entre les deux modèles (l'architecture de haut niveau et le modèle source), un calcul d'un modèle (appelé le modèle de réflexion) en utilisant l'outil ReflexionModel et la dernière étape est l'interprétation du modèle de réflexion par l'ingénieur. La sortie du processus est un

modèle, appelé modèle de reflexion, qui met en évidence les convergences, les divergences et les absences entre les deux modèles. Koschke et Simon ont étendu le travail de Murphy et al. pour supporter une architecture de haut niveau hiérarchique [Koschke et Simon, 2003]. Leur extension permet à l'ingénieur de décomposer le modèle de haut niveau (appelé la vue hypothétique dans l'extension) à différents niveaux de détails. La méthode complète pour appliquer le modèle de réflexion hiérarchique est la suivante : (1) à partir des informations disponibles, l'utilisateur chargé de l'extraction crée une première version de la vue hypothétique ; (2) l'utilisateur identifie les modules initiaux pour la vue concrète. Ces modules initiaux implémentent des abstractions clés du système et peuvent eux-mêmes contenir des modules ; (3) l'utilisateur trace les modules initiaux sur les entités conceptuelles de la vue hypothétique ; (4) le calcul du modèle de réflexion ; (5) selon les divergences et les absences, l'utilisateur ajuste la cartographie et la vue hypothétique. Le raffinement est établi itérativement jusqu'à ce que les vues concrètes et hypothétiques soient raisonnablement similaires.

Le processus de reconstruction proposé par Kazman et al.

Autre que les niveaux du modèle de Horseshoe présentés dans la section des modèles conceptuels 2.5.1, Kazman et al. [Kazman et Carrière, 1999] ont décrit d'une manière générale les étapes d'un processus d'extraction d'une représentation architecturale à partir du code source. Le processus implique tout d'abord l'extraction de plusieurs modèles concrets décrivant le code source ; chaque modèle représente un type d'information avec leurs relations. Par exemple, un modèle décrit les fonctions et leurs appels, un autre décrit les fichiers et leurs relations d'*import*, ensuite les relations entre les entités des modèles sont dérivées ; par exemple, les relations entre les fonctions et les fichiers sont extraites. Finalement, les entités des modèles sources sont correspondus à une abstraction conceptuelle (comme une architecture conceptuelle) en utilisant l'outil Dali [Kazman et Carrière, 1999].

2.6.3 Les méthodes basées sur la conciliation d'une architecture extraite avec une architecture conceptuelle

Les méthodes appartenant à cette catégorie sont basées sur les deux stratégies : le groupement des entités du code source et la correspondance des groupes d'entités à une architecture conceptuelle.

Le processus de l'ingénierie inverse proposé par Riva

Claudio Riva [Riva, 2000] propose un processus composé de 6 phases qui commence par l'analyse du système représenté au plus bas niveau pour arriver au niveau de l'architecture. La première phase englobe 2 étapes : La première consiste à définir les concepts architecturaux qui représentent les éléments du système. Ces concepts sont représentés sous forme de composants, des modules, des sous-systèmes. Et la deuxième étape est d'identifier comment ces concepts sont figurés dans le code source afin de faciliter l'extraction des informations au niveau du code source. Par exemple le concept d'un sous-système est représenté dans le code source par un répertoire.

La deuxième phase consiste à utiliser des outils automatiques pour extraire un modèle du code source ; ce modèle contient des entités qui caractérisent l'implémentation comme classes, fonctions, fichiers... A ces 2 phases l'implémentation est présentée par le modèle source et des indicateurs qui aident à faire une correspondance entre le modèle et l'architecture à haut niveau sont identifiés.

La troisième phase est l'abstraction où les entités du modèle source sont groupées pour créer une abstraction architecturale. Dans cette phase les outputs des 2 phases précédentes sont utilisés pour créer une abstraction au niveau de l'architecture à l'aide des documentations, de l'expertise humaine et des outils ; cette abstraction représente la vue étendue du système présentée dans le modèle source et respecte les concepts de correspondance entre le modèle source et l'architecture. Les dernières phases consistent à améliorer les documents concernant l'architecture dans le but d'augmenter la compréhension de l'architecture correcte du système et à déterminer les défauts de l'architecture en analysant les relations entre les composants.

L'approche Focus

Le but de cette approche [Ding et Medvidovic, 2001] est de faciliter l'évolution effective des systèmes Orientés Objets en utilisant une méthode de reconstruction de l'architecture du système. Le point fort de cette approche est qu'elle est appliquée d'une manière itérative permettant ainsi de reconstruire l'architecture physique et logique du système d'une part, et d'autre part de mettre à jour l'architecture pour être compatible avec les nouveaux changements du système. Chaque itération contient 2 grandes étapes : la récupération de l'architecture et l'évolution du système. Ces 2 étapes sont interdépendantes et appliquées à plusieurs reprises.

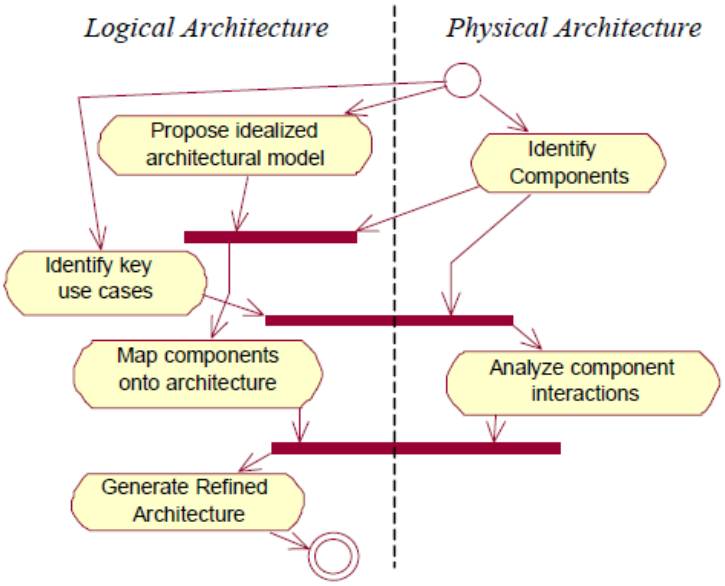

FIGURE 2.6 : Les différentes activités de l'approche Focus pour la reconstruction d'une l'architecture : extrait du [Ding et Medvidovic, 2001]

.

Première étape : La récupération de l'architecture. Cette étape est composée de 6 activités qui sont divisées en 2 catégories et dont la plupart sont accomplies manuellement : la reconstruction de l'architecture physique et la reconstruction de l'architecture logique comme montre la figure 2.6. En général, ces activités identifient les composants d'une part, et intègrent la proposition de l'architecte d'un modèle architectural d'autre part ; puis la correspondance entre les 2 outputs est établie. Afin de raffiner l'architecture intermédiaire obtenue par la correspondance, il est proposé d'étudier les interactions et le flux de données entre les composants. Pour cela Focus utilise la liste des cas d'utilisation et l'expertise humaine afin d'analyser les interactions et obtenir une architecture de bonne qualité.

1. L'identification de composants. Cette activité a pour but de déduire une architecture abstraite du haut niveau du système. Afin d'achever ce but, il faut générer en premier le diagramme de classe à partir du code source ; un des outils qui génère automatiquement le diagramme de classes à partir du code source est alors utilisé. Puis, les classes seront regroupées en des paquets selon les relations entre eux ; la technique de partitionnement *(clustering)* est utilisée pour faire un groupement selon les relations fortes entre les classes.
2. La proposition d'un modèle d'architecture idéalisée. L'architecte propose au début un style d'architecture selon l'aspect du système ; par exemple le style C2 est prévu pour les applications ayant un aspect de GUI. En se basant sur ce style, il propose un modèle d'architecture qui capte les propriétés de l'architecture. Donc à cette étape, un modèle d'architecture est proposée par l'architecte selon le style choisi.
3. La correspondance entre les composants identifiés et l'architecture idéalisée. Cette étape joint les compétences de l'architecte avec l'artefact concret qui est en relation avec le code source. Cette activité produit une architecture intermédiaire contenant les composants de l'activité 1 qui sont bien claires à inclure dans le modèle de l'activité 2. Les composants restants seront intégrés dans une autre étape.
4. L'identification des cas d'utilisation. Dans cette étape, il y a identification des U.C (cas d'utilisation) qui sont de 3 types : ceux qui ne sont pas en relation avec les nouveaux besoins, ceux qui doivent être ajoutés et ceux qui doivent être modifiés. Ces U.C sont déduits par l'observation du comportement de l'application du point de vue utilisateur.
5. Analyse de l'interaction des composants. Afin d'intégrer tous les composants dans l'architecture, il est proposé d'analyser les relations entre les composants. Cette analyse demande une analyse du flux de données entre les composants. Donc, c'est proposé de dresser les diagrammes de séquences pour les U.C de l'activité 4 en mettant les entités comme composants. A chaque itération, les U.C seront choisis selon leur priorité.
6. La génération d'une architecture raffinée de l'architecture intermédiaire proposée dans l'activité 3. C'est proposé d'utiliser le résultat de l'analyse des diagrammes de séquences afin de raffiner l'architecture.

Deuxième étape : l'évolution du système. Cette étape est composée de 5 activités qui ont pour but de mettre à jour l'architecture obtenue de l'étape précédente pour supporter le plan d'évolution du système. L'activité 1, consiste à proposer une nouvelle architecture du système du haut niveau qui inclut les nouvelles besoins. L'activité 2 consiste à ajouter, supprimer ou modifier des composants. L'activité 3 a pour but de mettre à jour les interactions entre les composants. Comme le flux de données a changé, il faut modifier les diagrammes de séquences pour analyser les interactions. La quatrième activité a pour but de générer la nouvelle architecture en intégrant tous les changements des étapes précédentes. Et la dernière activité, consiste à mettre un nouveau Focus : Si l'architecture générée dans l'étape précédente n'est pas suffisante, alors une nouvelle itération est faite.

L'approche proposée par Medvidovic

Dans le but d'arrêter l'érosion architecturale, [Medvidovic et al., 2003] Medvidovic et al. proposent d'utiliser des inputs fiables contenant des informations qui révèlent sur l'architecture du système. Parmi ces inputs, on trouve (1) les besoins du système (*Requirements*) qui aident d'une

manière explicite à découvrir des informations sur l'architecture, (2) l'implémentation qui représente le code source et (3) le style architectural qui sert à identifier le type des blocs et les relations entre eux. L'idée de l'approche consiste à combiner des techniques qui servent à découvrir l'architecture à partir des *requirements* avec des techniques d'extraction d'un modèle de l'architecture à partir de l'implémentation. Ces 2 techniques produisent comme résultat 2 modèles de l'architecture du système, l'un découvert et l'autre extrait. Vu que le style est une clé pour le design car il englobe la composition des éléments architecturaux et les décisions prises par l'architecte pour l'implémentation, l'approche s'appuie sur les styles pour être un moyen de réconciliation entre les 2 modèles. L'approche compte d'une part sur ces 2 modèles provenant de 2 sources fiables et d'autre part sur l'utilisation des styles architecturaux pour réconcilier les 2 modèles.

L'approche comporte 3 activités. La première activité est l'utilisation des techniques pour découvrir une architecture à l'aide des *requirements*. Ces derniers, qui sont déduits du comportement du système, peuvent révéler d'une manière explicite des informations sur l'architecture ; par exemple si un *requirement* nécessite un payement en ligne alors l'architecture doit contenir un composant de payement. Ces informations déduites seront organisées selon un modèle appelé DAM où chaque information est classifiée dans une catégorie des concernes architecturaux. Le modèle comporte 7 catégories : les composants de données, les composants de traitements, les connecteurs, les configurations, les propriétés des composants, les propriétés des connecteurs et les propriétés de sous-système. Donc, comme résultat cette activité produit un modèle DAM contenant des informations architecturales et organisées selon leurs concernes architecturaux.

La deuxième activité est la récupération d'une l'architecture à partir de l'implémentation. Le résultat de cette activité est un modèle, nommé RAM, qui est déduit grâce à une série d'activité d'extraction : commençant par la génération des digrammes de classes, puis le groupement de ces classes selon les techniques de partitionnement pour terminer avec l'assemblage des groupes en se basant sur plusieurs critères. Ce modèle ne présente pas en fait toute l'architecture, il englobe des éléments architecturaux, qui sont clairs d'être des éléments et présente quelques relations entre eux.

Après que les *requirements* sont raffinés dans des éléments de DAM selon les concernes architecturaux, et que l'implémentation est abstraite en un modèle RAM, les 2 modèles doivent être confrontés pour donner une architecture complète en se basant sur les informations communes. Vu que les styles précisent le vocabulaire de l'architecture, il est proposé de choisir un style architectural pour intégrer les 2 modèles. Cette intégration est appliquée soit en appliquant le style sur l'un des modèles pour obtenir une architecture, puis faire une correspondance entre cette dernière et l'autre modèle, soit en intégrant les 2 modèles pour avoir une architecture puis un style lui sera appliqué.

En conclusion, en se servant des *Requirements* et de l'implémentation, qui sont successivement des connaissances à un niveau élevé et à un niveau bas, une architecture conceptuelle du système est découverte et une architecture concrète est extraite. L'utilisation des styles a fortifié la création d'une architecture intermédiaire, presque complète, qui prend en considération toutes les informations présentées dans les 2 modèles.

2.7 Les outils d'extraction d'architectures logicielles

Les informations architecturales ne sont pas exprimées évidemment dans les artefacts existants (comme le code source, les documents ...). Ainsi, de nombreux outils ont été proposés pour soutenir l'extraction. Nous classifions ces outils, selon leur rôle, sous 3 catégories :

- Les extracteurs de faits *(facts extractors)*.
- Les outils de partitionnement (*clustering*).
- Les outils de correspondance.

2.7.1 Les extracteurs de faits

Les extracteurs de faits sont les outils les plus connus qui supportent l'extraction de l'architecture logicielle. Leur rôle est de fournir une représentation du code source (connue sous le nom de modèle source) indépendante du langage de programmation.

L'outil Columbus

L'extracteur de faits Columbus [Ferenc et al., 2002, Ferenc et al., 2004, Beszédes et al., 2005] analyse des systèmes C++ et représente la structure statique du code source selon le schéma Columbus [Ferenc et Beszedes, 2002]. Les opérations essentiels de l'outil sont : (1) l'analyse du code source qui vise à créer plusieurs instances du schéma de Columbus ; Une instance contient des entités et des relations représentées selon le schéma de Columbus [Ferenc et Beszedes, 2002] ; (2) La liaison des instances créées et leur filtrage qui visent à fusionner les différentes instances de la première étape dans un même schéma ; Et (3) la transformation du schéma en un de ces formats : les formats XML (CPPML, GXL [Holt et al., 2000a] et FAMIX XMI [Tichelaar et al., 2000]), RSF [Müller et Klashinsky, 1988], HTML et Maisa. Ces opérations sont effectuées via trois types de plug-ins : Les *plug-ins Extractor* qui analysent les fichiers sources et créent un fichier contenant les informations extraites, les *plug-ins Linker* qui fusionnent et filtrent les informations extraites des premiers plug-ins et les *plug-ins Exporter* qui exportent les fichiers dans de nombreux formats.

Les outils d'ingénierie inverse des modèles UML

Plusieurs outils fournissent un moyen d'extraction des faits en élaborant des modèles UML comme un digramme de classes en notation UML. En effet, presque tous les outils CASE *(Computer-aided Software Engineering)* [Muller et al., 2012] offrent la fonctionnalité d'ingénierie inverse en des modèles UML.

Autre que les outils CASE, Rational Rose [Quatrani, 2002], qui a été utilisé par l'approche Focus [Ding et Medvidovic, 2001] et Medvidovic et al. [Medvidovic et al., 2003] prend en charge l'ingénierie inverse des systèmes C++ et Java et donne comme résultat une vue arborescente comprenant des classes, des interfaces, leurs associations et fournit également un diagramme de classes UML.

L'outil Ptidej

Alors que les outils mentionnés donnent la structure statique du code source, plusieurs outils extraient la structure dynamique. Ptidej *(Pattern Trace Identification, Detection, and Enhancement in Java)* [Guéhéneuc, 2004] construit des diagrammes de classes à partir du code source Java. Plus précisément, Ptidej englobe deux outils d'analyse. Le premier outil, *PADL ClassFile creator*, analyse la structure statique du système. Le deuxième outil, Caféine, donne la structure dynamique des programmes Java en analysant le comportement d'exécution des programmes. Les auteurs ont développé un méta-modèle appelé PADL [Albin-Amiot et Guéhéneuc, 2001] qui décrit un programme comme diagramme de classes en termes de classes, de méthodes et de relations.

Les extracteurs d'un graphe MDG

Il existe de nombreux outils qui extraient les informations du code source sous forme d'un graphe de dépendance des modules, dénoté MDG *(Module Dependency Graph)*. Ce dernier représente un code source tel que les nœuds représentent les modules et les arcs représentent les dépendances entre eux. Par exemple, l'outil Chava [Korn et al., 1999] pour l'analyse des programmes Java. Les entités extraites sont : classes, interfaces, paquets, fichiers, méthodes, et variables ; Et les relations sont les relations d'héritage et de réalisation, des relations de déclaration et d'appel des méthodes. Autres outils qui génèrent un graphe MDG : CIA [Chen, 1995] pour C, Acacia [Chen et al., 1998] pour C et C++.

En outre, l'outil Gadget [Gargiulo et Mancoridis, 2001] extrait la structure dynamique à partir des programmes Java, il rassemble les données lors de l'exécution des programmes. La sortie est un graphe DDG *(Dynamic Dependency Graph)* où les nœuds représentent des classes ou des objets et les arcs représentent des relations telles que les invocations des méthodes.

2.7.2 Les outils de partitionnement

Les outils de partitionnement fournissent un moyen de groupement des entités de code source telles que les fichiers système ou les classes. Généralement, ces outils commencent par une partition (groupement) initiale et essaient de l'améliorer selon certains criteres.

L'outil Bunch

L'outil Bunch supporte plusieurs algorithmes à base de recherche. Mancoridis et al. [Mancoridis et al., 1999, Mitchell et Mancoridis, 2006] définissent d'une part une fonction d'évaluation *(fitness function)* qui guide les algorithmes vers la meilleure partition et d'autre part plusieurs algorithmes de partitionnement.

- La fonction d'évaluation
  La fonction d'évaluation, dénotée MQ 2.1, est celle qui dirige l'algorithme ; elle évalue une partition selon les critères de cohésion forte et de couplage faible. Mathématiquement, la fonction est en relation avec 2 mesures importantes mesurant les propriétés de connectivité : A_i mesurant l'intra-connectivité dans un cluster i et $E_i j$ qui mesure la connectivité entre 2 clusters distincts i et j. La fonction, qui a été considérée comme base pour d'autres travaux, [Praditwong et al., 2011, Qiu et al., 2015, Chardigny, 2009], est la suivante, avec k le nombre de clusters d'une partition.

$$MQ = \begin{cases} \frac{\sum_{i=1}^{k} A_i}{k} - \frac{\sum_{i,j=1}^{k} E_{i,j}}{\frac{k(k-1)}{2}} & \forall k > 1 \\ A_i & k = 1 \end{cases} \tag{2.1}$$

- Algorithmes
  Dans cette section, on présente les trois algorithmes adaptés par l'outil Bunch. Principalement, ces algorithmes cherchent à minimiser le couplage entre deux clusters différents, c.à.d la valeur $E_i j$ et à maximiser la cohésion forte à l'intérieur de chaque cluster, c.à.d la valeur A_i.

  L'algorithme optimal de clustering : Cet algorithme génère toutes les partitions possibles, puis une évaluation de toutes les partitions est établie grâce à la fonction fitness. Une partition des modules est un ensemble de parties non vides qui sont

deux à deux disjointes et qui recouvrent tous les modules du système. Une génération de toutes les partitions donne comme résultat un grand nombre interdisant ainsi sa réalisation sur les systèmes complexes.

L'algorithme sous optimale : Cet algorithme commence avec une partition aléatoire et essaie de trouver une partition voisine plus meilleure. L'algorithme s'arrête quand une partition n'a pas de voisines ayant une fitness plus élevée. Donc au lieu de générer toutes les partitions comme l'algorithme précédent, une seule partition voisine est générée, si elle est meilleure alors elle devient la partition courante. Cette instruction sera répétée jusqu'à arriver à la condition d'arrêt de l'algorithme. Cet algorithme est celui de Hill Climbing et il présente comme inconvénient le maximum local.

L'algorithme génétique : Cet algorithme consiste à commencer avec une population N qui contient plusieurs partitions initiales arbitraires et à les améliorer jusqu'à ce que les partitions convergent. L'amélioration se passe comme suit : sélectionner un pourcentage de N et trouver un voisin meilleur pour chaque individu de la population. Une nouvelle population sera alors générée en sélectionnant N partitions avec remplacement de la population existante. L'algorithme s'arrête quand les partitions convergent vers leur fitness maximale ou quand le nombre de génération dépasse un nombre maximal de génération. A la fin la meilleure partition est choisie de la population finale.

En résumé, Mancordis et al. ont adapté les algorithmes à base de recherche sur la modularisation logicielle tout en développant un outil de clustering nommé Bunch. De plus, Mancordis a mentionné dans [Mancoridis et al., 1998] que pour un système complexe, le nombre de clusters peut être très large. Pour cela, il faut faire un clustering pour les clusters, ce qui crée une hiérarchie des sous-systèmes ayant le système comme racine de la hiérarchie et les modules au dernier niveau.

2.7.3 Les outils de correspondance

Les outils de correspondance prennent en charge l'extraction en faisant correspondre les entités du code source du système à des connaissances de haut niveau. Une connaissance de haut niveau pourrait être considérée comme une architecture conceptuelle qui reflète le point de vue de l'architecte.

L'outil Reflexion Model

L'outil ReflexionModel [Murphy et al., 1995, Murphy et Notkin, 1997, Murphy et al., 2001] est basé sur l'élaboration d'un modèle qui calcule les différences entre le modèle de haut niveau des développeurs et le modèle source. L'outil aide l'ingénieur à valider son modèle mental structurel d'un système. L'outil ReflexionModel prend trois inputs : un modèle de haut niveau proposé par l'ingénieur, un modèle source qui représente le code source et une carte définie par l'architecte. La carte spécifie comment les deux modèles (le modèle de haut niveau et le modèle source) peuvent être correspondus. L'outil calcule un modèle (appelé modèle de réflexion) qui met en évidence les convergences (où le modèle source est conforme au modèle de haut niveau), les divergences (où le modèle source comprend des relations non prévues par le modèle de haut niveau) et les absences (où le modèle source ne comprend pas des relations prédites par le modèle de haut niveau).

L'outil Dali

Kazman et al. proposent un outil appelé Dali [Kazman et Carrière, 1999] qui intègre plusieurs techniques comme l'outil Rigi [Tilley et al., 1994] et l'outil de requête POSTGREDSQL. Ces outils permettent l'extraction d'un modèle source à partir d'un code C ou C++, l'enregistrement des résultats dans une base de données relationnelle, l'exécution des requêtes données par l'ingénieur et la visualisation des résultats. Dans cet outil, la correspondance entre le code source et le modèle conceptuel proposé par l'architecte est effectuée à travers le mécanisme de requête fourni par l'outil de requête.

2.8 Conclusion

Dans ce chapitre, nous avons introduit les terminologies liées aux architectures logicielles et l'extraction d'une architecture logicielle. Vu que notre problématique porte sur l'absence d'un méta-modèle aidant l'architecte à choisir un processus d'extraction convenable, nous avons présenté les travaux liés à la spécification des concepts concernant l'extraction. De plus, nous avons focalisé sur les processus d'extraction tout en les catégorisant selon la stratégie utilisée. Ces différents processus seront analysés dans le chapitre suivant ; nous soulignons leurs limites surtout celles concernant l'habilité de l'architecte à intégrer ses connaissances et interagir avec le processus. Finalement, les outils supportant les processus d'extraction sont présentés.

Ainsi, dans le chapitre suivant, nous menons une analyse qui permet d'évaluer les différentes approches d'extraction étudiées (méta-modèles, processus et outils) afin de souligner leurs limites.

3

Analyse comparative et limitations

3.1 Introduction

En se basant sur l'étude bibliographique présentée dans le chapitre précédent, nous effectuons une analyse des différentes approches d'extraction d'une architecture logicielle. L'analyse a pour but d'évaluer la qualité de chaque approche d'extraction d'une architecture logicielle, de comparer les différentes approches d'extraction selon plusieurs critères et de souligner les limitations des approches existantes. Pour atteindre ce but, nous nous sommes basés sur le modèle de qualité McCall. Ainsi, nous présentons dans ce chapitre le modèle McCall, notre modèle de qualité basé sur McCall et l'évaluation des différentes approches sur notre modèle de qualité.

3.1.1 Le modèle McCall

McCall est un modèle de qualité qui a été créé pour mesurer le niveau de qualité d'un logiciel pour l'US Air Force [Company et al., 1977]. Le modèle McCall est hiérarchique, il recense les principes de qualité en partant des exigences globales et des principes les plus généraux pour descendre vers les métriques qui permettent de les mesurer. Plus spécifiquement, le modèle McCall est composé de trois couches : les métriques, les critères et les facteurs.

- **Les facteurs de qualité**
  Le premier niveau contient des facteurs de qualité. Ce sont des attributs du haut-niveau qui constituent les caractéristiques influant sur la qualité d'un logiciel. Par exemple, l'efficacité d'un logiciel est un facteur de qualité.

- **Les critères de qualité**
  Le deuxième niveau décrit les critères de qualité. Ces critères permettent de diviser les facteurs de qualité du premier niveau et d'obtenir des attributs internes à un niveau plus bas. Par exemple, l'efficacité est décomposée en deux critères : l'efficacité de l'exécution du logiciel et l'efficacité du stockage.

- **Les métriques de qualité**
  Le troisième niveau décrit les métriques de qualité. Ces métriques sont le dernier niveau de décomposition et évaluent les critères de qualité du deuxième niveau. Les paramètres

ne sont pas définis dans le modèle McCall. Il est adapté à chaque utilisateur du modèle de définir toutes les métriques les plus appropriées dans sa zone, son système et ses objectifs.

3.2 Notre modèle de qualité

En nous basant sur McCall, nous avons élaboré un modèle qui suit 3 niveaux principaux pour mesurer la qualité des approches d'extraction d'une architecture logicielle. Ainsi, au lieu d'utiliser le modèle McCall pour la mesure de la qualité d'un logiciel, nous l'avons utilisé pour mesurer la qualité des approches d'extraction. Par la suite, notre modèle suit les trois niveaux qui sont les facteurs, les critères et les métriques. Dans ce qui suit, les trois niveaux sont détaillés.

3.2.1 Les facteurs de qualité

A notre avis, une approche d'extraction doit définir un méta-modèle qui détaille les concepts généraux du principe d'extraction, définir un processus d'extraction et être supportée par un outil d'extraction. Ainsi, nous distinguons trois facteurs de qualité pour mesurer la qualité des approches d'extraction d'une architecture logicielle.

- La définition d'un méta-modèle d'extraction d'une architecture logicielle.
- La proposition d'une méthode d'extraction d'une architecture logicielle.
- L'existence d'un outil d'extraction d'une architecture logicielle.

3.2.2 Les critères de qualité et leurs métriques

En partant des facteurs, qui sont définis dans le paragraphe précédent, nous avons défini les critères qui sont au niveau 2 du modèle McCall ; chacun des trois facteurs définis est décomposé en plusieurs critères. En addition, chaque critère est décomposé en des métriques indiquant la qualité (Bonne, Moyenne, Mauvaise) de chaque critère. Ainsi, dans cette section nous présentons les critères de qualité (au niveau 2 du modèle McCall) pour chaque facteur et les métriques de qualité (au niveau 3 du modèle McCall) de chaque critère.

Les critères de qualité de la définition d'un méta-modèle

Un méta-modèle des processus d'extraction d'architectures logicielles spécifie les concepts des processus d'extraction. Ainsi, les critères, présentés ci-dessous, sont pris en considération pour étudier si les méta-modèles définis par chaque approche détaillent suffisamment les concepts des processus d'extraction.

1. **La spécification des niveaux**
   Ce critère indique si le méta-modèle détaille les niveaux sur lesquels opèrent les processus d'extraction. Par exemple, un niveau sur lequel opère les processus d'extraction peut être le niveau du code source, le niveau du modèle source, le niveau architectural...
   Les métriques qui évaluent ce critère sont :

   - Le méta-modèle ne définit aucun niveau : Mauvaise qualité (MV).
   - Le méta-modèle définit deux niveaux, niveau inférieur et niveau supérieur : Qualité moyenne (MY).

- Le méta-modèle définit plus que deux niveaux et les détaille : Bonne qualité (B).

2. **La séparation du niveau architectural**
   Ce critère indique si le méta-modèle définit le niveau architectural parmi les niveaux sur lesquels opèrent les processus d'extraction.
   Les métriques qui évaluent ce critère sont :

   - Le méta-modèle ne définit pas le niveau architectural : Mauvaise qualité (MV).
   - Le méta-modèle définit le niveau architectural : Bonne qualité (B).

3. **La spécification des activités générales**
   Ce critère indique si le méta-modèle définit les activités générales des processus d'extraction. En fait, un méta-modèle dédié aux processus ayant une certaine fin doit définir les activités générales de ces processus.
   Ainsi les métriques de ce critère sont :

   - Le méta-modèle ne définit aucune activité des processus d'extraction : Mauvaise qualité (MV).
   - Le méta-modèle définit au moins les activités de passage d'un niveau à l'autre : Qualité moyenne (MY).
   - Le méta-modèle définit toutes les activités générales des processus d'extraction : Bonne qualité (B).

4. **La spécification des artefacts essentiels**
   Ce critère indique si le méta-modèle détaille les artefacts essentiels des processus d'extraction. Un méta-modèle d'extraction d'une architecture logicielle doit définir les artefacts essentiels des processus d'extraction.
   Ainsi les métriques de ce critère sont :

   - Le méta-modèle ne définit aucun artefact essentiel des processus d'extraction : Mauvaise qualité (MV).
   - Le méta-modèle définit les artefacts initiaux et finaux des processus d'extraction : Qualité moyenne (MY).
   - Le méta-modèle définit tous les artefacts initiaux, finaux et intermédiaires essentiels des processus d'extraction : Bonne qualité (B).

5. **La spécification des agents essentiels**
   Ce critère indique si le méta-modèle spécifie les agents essentiels des processus d'extraction. Un méta-modèle d'extraction d'une architecture logicielle doit identifier les agents essentiels des processus d'extraction. Par rôle, on entend une entité physique telle qu'un architecte, un développeur, un programmeur... ou une entité abstraite telle qu'un outil, une technique, un algorithme...
   Ainsi les métriques de ce critère sont :

   - Le méta-modèle ne définit aucun agent essentiel des processus d'extraction : Mauvaise qualité (MV).
   - Le méta-modèle sépare entre les agents physiques et les agents abstraits des processus d'extraction : Qualité moyenne (MY).
   - Le méta-modèle détaille tous les agents essentiels des processus d'extraction : Bonne qualité (B).

Les critères de qualité d'une méthode

Selon la problématique étudiée sur les méthodes d'extraction, nous avons analysé les méthodes d'extraction selon les critères suivants :

1. **L'intégration des connaissances de l'architecte**
   Ce critère indique si la méthode d'extraction prend en considération les connaissances de l'architecte ; ces connaissances sont des informations à propos du système acquis par l'architecte grâce à son expertise et les documentations existantes du système.
   Les métriques de ce critère sont :

   - La méthode d'extraction n'intègre pas les connaissances de l'architecte : Mauvaise qualité (MV).
   - La méthode d'extraction intègre les connaissances de l'architecte en lui permettant de proposer seulement une architecture conceptuelle : Qualité moyenne (MY).
   - La méthode d'extraction intègre les connaissances de l'architecte en lui permettant de proposer une architecture conceptuelle et d'ajouter d'autres informations durant le processus : Bonne qualité (B).

2. **L'interaction de l'architecte**
   Ce critère indique si la méthode d'extraction permet l'interaction de l'architecte avec le processus d'extraction. Plus spécifiquement, l'interaction de l'architecte signifie que l'architecte puisse interagir avec les artefacts manipulés durant le processus d'extraction et configurer le processus selon ses besoins.
   Les métriques de ce critère sont :

   - La méthode d'extraction ne permet pas l'interaction de l'architecte avec le processus d'extraction : Mauvaise qualité (MV).
   - La méthode d'extraction permet l'interaction de l'architecte en lui permettant de raffiner les artefacts : Qualité moyenne (MY).
   - La méthode d'extraction permet l'interaction de l'architecte en lui permettant de raffiner les artefacts et configurer le processus selon ses besoins : Bonne qualité (B).

3. **Les types des informations**
   Ce critère indique si la méthode d'extraction extrait une architecture logicielle à partir des informations qui représente efficacement le code source. Ainsi, ce critère signifie les types des informations extraites à partir du code source. Par exemple une méthode qui extrait l'architecture à partir des fichiers et la structuration des répertoires donne un résultat moins valorisant que celle qui extrait l'architecture à partir des fonctions et leurs appels, les classes et leurs relations.
   Par la suite, les métriques de ce critère sont :

   - La méthode d'extraction extrait des informations pas trop significatives comme les fichiers physiques et leurs relations : Mauvaise qualité (MV).
   - La méthode d'extraction extrait des informations ayant une signification moyenne comme les fonctions, les classes, fichiers physiques et leurs relations : Qualité moyenne (MY).
   - La méthode d'extraction extrait des informations de différentes types comme la structure des fichiers physiques, les classes, les fonctions, le flux de données durant l'exécution : Bonne qualité (B).

4. **Le groupement des entités logicielles**
   Ce critère indique si la méthode d'extraction regroupe les entités logicielles (comme les classes et les fonctions) pour obtenir des éléments architecturaux de haut niveau. En d'autre termes, ce critère indique si la méthode d'extraction est basée sur le principe de groupement.
   Les métriques de ce critère sont :

   - La méthode d'extraction n'est pas basée sur le principe de groupement : Mauvaise qualité (MV).
   - La méthode d'extraction groupe une seule fois les entités logicielles : Qualité moyenne (MY).
   - La méthode d'extraction groupe plusieurs fois les entités logicielles, c.à.d la méthode suit un groupement hiérarchique pour passer des entités logicielles de bas niveau jusqu'à l'architecture de haut-niveau : Bonne qualité (B).

5. **La correspondance des entités logicielles à l'architecture conceptuelle**
   Ce critère indique si la méthode d'extraction prend en considération une architecture conceptuelle proposée par l'architecte et inclut la correspondance des entités logicielles à l'architecture conceptuelle. En d'autres termes, ce critère indique si la méthode d'extraction est basée sur le principe des architectures conceptuelles.
   Les métriques de ce critère sont :

   - La méthode d'extraction n'est pas basée sur le principe des architectures conceptuelles : Mauvaise qualité (MV).
   - La méthode d'extraction est basée sur le principe des architectures conceptuelles mais n'a pas identifié la méthode de correspondance des entités à l'architecture conceptuelle : Qualité moyenne (MY).
   - La méthode d'extraction est basée sur le principe des architectures conceptuelles avec identification de la méthode de correspondance : Bonne qualité (B).

6. **L'identification d'une architecture logicielle complète**
   Ce critère indique si la méthode d'extraction a donné comme résultat une architecture logicielle selon la définition d'architecture logicielle la plus admise. Celle-ci est la définition de Len et al. [Len et al., 2003] (voir 2.2.5). Ainsi, ce critère indique si l'architecture identifiée est composée des éléments, des relations entre les éléments et des propriétés des deux.
   Par la suite les métriques de ce critère sont :

   - La méthode d'extraction identifie une architecture logicielle en termes d'éléments architecturaux, leurs relations et propriétés : Bonne qualité (B).
   - La méthode d'extraction identifie une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent : Qualité moyenne (MY).
   - Autres : Mauvaise qualité (MV).

Les critères de qualité d'un outil

Un outil d'extraction supporte le processus d'extraction. Ainsi, nous avons analysé les outils d'extraction en étudiant s'il supporte suffisamment les méthodes d'extraction. Ainsi, les critères de qualité d'un outil sont les suivants :

1. **L'intégration des connaissances l'architecte**
   Ce critère indique si l'outil d'extraction permet à l'architecture d'intégrer ses connaissances dans le processus comme définir une architecture conceptuelle.
   Les métriques de ce critère sont :

   - L'outil ne supporte pas l'intégration des connaissances de l'architecte : Qualité Mauvaise (MV).
   - L'outil supporte l'intégration des connaissances de l'architecte : Bonne qualité (B).

2. **L'interaction de l'architecte**
   Ce critère indique si l'outil d'extraction permet à l'architecte de raffiner les artefacts et d'interagir avec le processus, par exemple configurer certains paramètres.
   Les métriques de ce critère sont :

   - L'outil ne supporte pas l'interaction de l'architecte : Mauvaise qualité (MV).
   - L'outil ne supporte le raffinement des artefacts : Qualité moyenne (MY).
   - L'outil supporte le raffinement des artefacts et la configuration de certains paramètres : Bonne qualité (B).

3. **La structure extraite**
   Ce critère indique si l'outil extrait la structure statique et la structure dynamique à partir du code source. La structure dynamique définit les éléments du système et leurs relations lors de l'exécution du système et la structure statique définit les éléments du système et leurs relations sans exécution.
   Par la suite les métriques de ce critère sont :

   - L'outil extrait la structure statique : Qualité moyenne (MY).
   - L'outil extrait la structure statique et dynamique : Bonne qualité (B).

4. **La représentation des informations**
   Ce critère indique si l'outil d'extraction représente les informations extraites selon un modèle ou un standard. Par exemple, un modèle peut être un diagramme de classes selon la notation UML.
   Les métriques de ce critère sont :

   - L'outil n'extrait pas les informations selon un modèle ou un standard : Qualité moyenne (MY).
   - L'outil extrait les informations selon un modèle ou un standard : Bonne qualité (B).

5. **Les langages de programmation supportés**
   Ce critère indique si l'outil d'extraction opère sur plusieurs langages de programmation.
   Les métriques de ce critère sont :

   - L'outil n'opère que sur un seul langage de programmation : Qualité moyenne (MY).
   - L'outil opère sur plusieurs langages de programmation : Bonne qualité (B).

6. **Les types de programmation supportés**
   Ce critère indique si l'outil d'extraction opère sur plusieurs types de programmation comme programmation procédural ou orienté objet.
   Les métriques de ce critère sont :

- L'outil n'opère que sur le type de programmation procédural : Mauvaise qualité (MV).
- L'outil n'opère que sur le type de programmation orienté objet : Moyenne qualité (MY).
- L'outil opère sur plusieurs types de programmation : Bonne qualité (B).

7. **Le groupement des entités logicielles**
   Ce critère indique si l'outil d'extraction permet le groupement (partitionnement) des entités logicielles.
   Les métriques de ce critère sont :

   - L'outil ne supporte pas le groupement des entités logicielles : Qualité Mauvaise (MV).
   - L'outil supporte un seul groupement des entités logicielles : Qualité Moyenne (MY).
   - L'outil supporte plusieurs groupement des entités logicielles : Bonne qualité (B).

8. La correspondance des entités logicielles à une architecture conceptuelle
   Ce critère indique si l'outil d'extraction prend en considération une architecture conceptuelle proposée par l'architecte et permet la correspondance entre les entités logicielles et cette architecture.
   Les métriques de ce critère sont :

   - L'outil ne supporte pas la correspondance : Qualité Mauvaise (MV).
   - L'outil supporte la correspondance : Bonne qualité (B).

3.3 L'évaluation des approches selon notre modèle de qualité

Dans cette section, nous évaluons les différentes approches détaillées dans le chapitre 2 selon notre modèle de comparaison.

3.3.1 L'évaluation des travaux de Kazman et al. selon notre modèle de qualité

Kazman et al. [Kazman et al., 1998] ont défini un modèle conceptuel pour la spécification des processus d'extraction, un processus de reconstruction d'une architecture logicielle, ainsi qu'un outil [Kazman et Carrière, 1999]. La table 3.1 résume l'évaluation des travaux de Kazman et al. selon notre modèle de qualité. Le modèle conceptuel définit 4 niveaux sur lesquels opèrent les processus d'extraction dont un niveau est le niveau architectural (nommé la représentation architecturale par Kazman et al.). Cependant le modèle ne fournit aucun détail sur les activités, les artefacts et les rôles.

De plus, Kazman et al. ont proposé une méthode de reconstruction d'une architecture logicielle [Kazman et Carrière, 1999]. Leur méthode permet seulement l'intégration de l'architecte en lui permettant de proposer une abstraction conceptuelle. Concernant les types des informations, les entités logicielles peuvent être des fichiers, des classes, des fonctions, des variables, des structures de données avec leurs relations. Le résultat final est une architecture en termes d'éléments architecturaux et les éléments qui les composent.

En ce qui concerne les outils, Kazman et al. ont proposé l'outil Dali. Ce dernier fournit une unité de stockage, d'exécution des requêtes et de visualisation à l'architecte. Ainsi, l'outil est

utilisé pour (1) extraire un modèle source qui représente le système au niveau des fonctions ; la structure extraite du code source est une structure statique, et (2) exécuter la correspondance du modèle source à une architecture conceptuelle.

3.3.2 L'évaluation des travaux de Koschke selon notre modèle de qualité

Koschke [Koschke, 2000] a défini seulement un modèle conceptuel (voir table 3.2). Selon Koschke, il n'y a pas de différence entre le niveau de la structure source et le niveau du code puisque ce dernier est plus ou moins une représentation exacte du code source. Pour cela, le modèle de Koschke fusionne les deux premiers niveaux du modèle de Kazman et al. De même que le modèle de Kazman et al., le méta-modèle de Koschke ne fournit aucun détail sur les activités, les artefacts et les rôles. La table 3.2 résume la projection des travaux de Koschke selon notre modèle de comparaison.

3.3.3 L'évaluation de l'approche de Chardigny selon notre modèle de qualité

Chardigny [Chardigny, 2009] a défini un modèle conceptuel pour l'extraction d'une architecture logicielle. Son modèle spécifie les niveaux en tenant compte des activités d'extraction. En effet, son modèle est inspiré des stratégies des processus ; ainsi, il a précisé quelques activités d'extraction. Ces dernières sont : l'extraction d'un modèle source, le regroupement au sein des entités du modèle source, l'abstraction des groupes en des éléments architecturaux et la mise en correspondance. Néanmoins, Chardigny n'a pas défini un processus complet pour l'extraction, plutôt, son travail s'est focalisé sur la définition d'une fonction objective guidant les algorithmes de recherche. Ainsi, nous résumons dans la table 3.3 les travaux de Chardigny seulement en termes de méta-modèle.

3.3.4 L'évaluation de l'approche de Riva selon notre modèle de qualité

Riva a défini des niveaux d'abstraction et une méthode d'extraction qui suit les niveaux [Riva, 2000]. La table 3.4 résume la projection des travaux de Riva selon notre modèle de comparaison. Cinq niveaux d'abstraction sont détaillés allant du niveau de l'implémentation vers le niveau des besoins fonctionnels tout en passant par le niveau architectural. Bien que Riva a spécifié les niveaux des processus d'extraction, Riva n'a pas fourni des détails concernant les activités, les artefacts et les rôles.

Dans sa méthode d'extraction, Riva a mentionné le but de chaque phase sans détailler le processus. Selon [Riva, 2000], l'intégration des connaissances de l'architecte consiste à lui permettre de proposer les concepts architecturaux tels que l'architecture conceptuelle. Cependant, l'architecte ne peut pas remettre en cause les artefacts générés. Concernant les types des informations, la méthode d'extraction peut être appliquée sur n'importe quel type d'informations. En addition, la méthode de Riva est basée sur les deux principes : le groupement des entités logicielles (groupement plusieurs fois) et la proposition d'une architecture conceptuelle (sans identification de la manière de correspondance). Le résultat obtenu à la fin du processus est une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent.

En ce qui concerne un outil qui supporte la méthode d'extraction, Riva a proposé d'utiliser (1) un analyseur de code source pour extraire un modèle source qui représente la structure des fichiers et les appels des fonctions, et (2) des outils de visualisation et de groupement pour faciliter à l'architecte le groupement des entités et le raffinement des groupes. Cependant l'approche ne précise pas quels sont les outils et/ou techniques de groupement à utiliser.

TABLE 3.1 : L'évaluation des travaux de Kazman et al. selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	Spécification des niveaux	Plusieurs niveaux sont définis et détaillés	B
	Séparation du niveau architectural	Niveau architectural séparé	B
	Spécification des activités générales	Activités générales non définies	MV
	Spécification des artefacts essentiels	Artefacts essentiels non définis	MV
	Spécification des agents essentiels	Agents essentiels non définis	MV
Méthode	Intégration des connaissances de l'architecte	Seulement proposition d'une architecture conceptuelle	MY
	Interaction de l'architecte	Pas d'interaction de l'architecte	MV
	Types des informations	Des informations ayant une signification moyenne	MY
	Groupement des entités logicielles	Pas de groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance sans identification de la méthode	MY
	Identification d'une architecture logicielle complète	Une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Seulement proposition d'une architecture conceptuelle	MY
	Interaction de l'architecte	Ne permet pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Supporte la correspondance	B

TABLE 3.2 : L'évaluation des travaux de Koschke selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	Spécification des niveaux	Plusieurs niveaux sont définis et détaillés	B
	Séparation du niveau architectural	Niveau architectural séparé	B
	Spécification des activités générales	Activités générales non définies	MV
	Spécification des artefacts essentiels	Artefacts essentiels non définis	MV
	Spécification des agents essentiels	Agents essentiels non définis	MV
Méthode	-	-	-
Outil	-	-	-

TABLE 3.3 : L'évaluation de l'approche de Chardigny selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	Spécification des niveaux	Plusieurs niveaux sont définis et détaillés	B
	Séparation du niveau architectural	Niveau architectural séparé	B
	Spécification des activités générales	Définission des activités de passage d'un niveau à l'autre	MY
	Spécification des artefacts essentiels	Artefacts essentiels non définis	MV
	Spécification des agents essentiels	Agents essentiels non définis	MV
Méthode	-	-	-
Outil	-	-	-

3.3.5 L'évaluation des états de l'art selon notre modèle de qualité

Ducasse et Pollet [Pollet et al., 2007, Ducasse et Pollet, 2009] ont présenté une comparaison de certaines approches d'extraction. Bien que cette étude aide un architecte à choisir parmi les approches, elle reste loin d'être considérée comme un méta-modèle pour la spécification des processus d'extraction. Premièrement, l'étude n'a pas détaillé les niveaux sur lesquels opèrent les processus d'extraction. Deuxièmement, concernant les activités générales d'extraction, les travaux de Ducasse et Pollet n'ont pas spécifié ces activités. Par contre, le cadre de comparaison a souligné les inputs initiaux et finaux des processus et a détaillé le niveau d'automaticité de chaque processus. Ainsi, les travaux de Ducasse et Pollet ont spécifié certains artefacts et agents qui doivent exister dans les processus d'extraction (voir 3.5).

De même, De Silva et Balasubramaniam [De Silva et Balasubramaniam, 2012] ont présenté une comparaison entre certaines approches pour lutter contre l'érosion. A l'instar des travaux de Ducasse et Pollet, le modèle de comparaison n'a pas détaillé les niveaux sur lesquels opèrent les processus d'extraction. En effet, le but de leur travail est de catégoriser certaines approches qui aident à contrôler l'érosion selon la stratégie adoptée. Ainsi, ce travail n'a pas fourni des

TABLE 3.4 : L'évaluation de l'approche de Riva selon notre modèle de qualité.

Facteur	**Critère**	**Métrique**	Q
Méta-modèle	Spécification des niveaux	Plusieurs niveaux sont définis et détaillés	B
	Séparation du niveau architectural	Niveau architectural séparé	B
	Spécification des activités générales	Activités générales non définies	MV
	Spécification des artefacts essentiels	Artefacts essentiels non définis	MV
	Spécification des agents essentiels	Agents essentiels non définis	MV
Méthode	Intégration des connaissances de l'architecte	Seulement proposition d'une architecture conceptuelle	MY
	Interaction de l'architecte	Seulement raffinement des artefacts	MY
	Types des informations	Des informations de différents types	B
	Groupement des entités logicielles	Groupement plusieurs fois des entités logicielles	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance sans identification de la méthode	MY
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Supporte le raffinement des artefacts	MY
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Supporte le groupement des entités logicielles (non bien détaillé)	MY
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

TABLE 3.5 : L'évaluation des travaux de Ducasse et Pollet selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	Spécification des niveaux	Aucun niveau n'est défini	MV
	Séparation du niveau architectural	Pas de séparation du niveau architectural	MV
	Spécification des activités générales	Activités générales non définies	MV
	Spécification des artefacts essentiels	Artefacts initiaux et finaux définis	MY
	Spécification des agents essentiels	Séparation entre performant automatique ou manuel	MY
Méthode	-	-	-
Outil	-	-	-

concepts génériques concernant l'extraction. Du point de vue de l'architecte, ce travail l'aide à distinguer entre les approches qui tentent à récupérer, découvrir ou à réconcilier l'architecture.

3.3.6 L'évaluation de l'approche de Mancoridis et al. selon notre modèle de qualité

Mancoridis et al. [Mancoridis et al., 1998, Mancoridis et al., 1999, Mitchell et al., 2001, Mitchell et Mancoridis, 2006, Mitchell et Mancoridis, 2008, Doval et al., 1999] ont proposé l'extraction d'une architecture en utilisant les algorithmes de recherche. Ainsi, leur travail s'est focalisé sur l'élaboration d'une fonction objective et l'amélioration des algorithmes. Donc, comme méthode d'extraction, ils ont fourni un processus de deux étapes : La première étape est l'extraction d'un graphe MDG, supportée par les extracteurs d'un graphe MDG et la deuxième étape est l'exécution de l'algorithme de recherche en utilisant l'outil Bunch. Ainsi, leur travail néglige tout type d'interaction ou d'intégration de connaissances de l'architecte. La table 3.6 résume la projection des travaux de Mancoridis el al. selon notre modèle de comparaison.

3.3.7 L'évaluation de l'approche de Mahdavi et al. selon notre modèle de qualité

Mahdavi et al. [Mahdavi et al., 2003a, Mahdavi et al., 2003b] ont proposé un processus d'extraction pour obtenir une décomposition du système (qui est une architecture) ayant un couplage faible entre les éléments et une cohésion forte entre les entités d'un même élément. Le processus, supporté par les extracteurs des graphes MDG et l'outil Bunch, se focalise sur l'idée de groupement plusieurs fois, tout en prenant en compte certains informations de la part de l'architecte. En effet, l'idée de Mahdavi et al. consiste à élaborer des *Building Blocks*. Ces entités logicielles sont des groupes d'entités obtenus après une exécution multiple de l'algorithme Hill Climbing. Ainsi, l'architecte choisit ces *Building Blocks* tout en comparant les résultats des exécutions. Par la suite, nous pouvons dire que l'architecte interagit et intègre ses connaissances avec un faible degré dans le processus d'extraction. La table 3.7 résume l'évaluation de l'approche de Mahdavi et al. selon notre modèle de qualité.

TABLE 3.6 : L'évaluation de l'approche de Mancoridis et al. selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Pas d'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Pas d'interaction de l'architecte	MV
	Types des informations	Différents types	B
	Groupement des entités logicielles	Groupement plusieurs fois	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Pas de correspondance	MV
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne supporte pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Ne supporte pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Supporte plusieurs groupement des entités logicielles	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

3.3.8 L'évaluation de l'approche de Rajalakshmi selon notre modèle de qualité

Rajalakshmi [Rajalakshmi, M, 2014] a proposé un processus d'extraction pour obtenir une décomposition du système ayant un couplage faible entre les éléments et une cohésion forte entre les entités d'un même élément tout en prenant en compte l'avis de l'architecte. La table 3.8 résume la projection des travaux de Mahdavi selon notre modèle de comparaison. Son processus se focalise sur l'exécution du processus de groupement plusieurs fois mais en ajoutant à chaque fois des contraintes données par l'architecte. Ainsi, la méthode d'extraction de Rajalakshmi est basée sur l'interaction de l'architecte après chaque groupement dans le but de raffiner les résultats des groupements.

Comme Mahdavi, Rajalakshmi a proposé d'utiliser un outil pour extraire un graphe MDG représentant les modules et leurs relations et l'outil Bunch pour grouper les entités logicielles.

TABLE 3.7 : L'évaluation de l'approche de Mahdavi et al. selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Intégration moyenne, en choisissant les *Building Blocks*	MY
	Interaction de l'architecte	Interaction moyenne, en choisissant les *Building Blocks*	MY
	Types des informations	Différents types	B
	Groupement des entités logicielles	Groupement plusieurs fois	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Pas de correspondance	MV
	Identification d'une architecture logicielle complète	Une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Ne permet pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Supporte plusieurs groupement des entités logicielles	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

3.3.9 L'évaluation de l'approche de Bowman et al. selon notre modèle de qualité

Bowman et al. [Bowman et al., 1999] ont proposé une méthode d'extraction qui est quasi-manuelle. La table 3.9 résume la projection des travaux de Bowman et al. selon notre modèle de comparaison. Cette approche, qui vise à extraire l'architecture du système Linux, repose sur la correspondance (d'une manière manuelle) des fichiers du système Linux à une architecture conceptuelle proposée par l'architecte. Ainsi, l'architecte intègre ses connaissances et interagit avec le processus tout en proposant une architecture conceptuelle, en faisant la mise en correspondance et en raffinant les résultats. Le résultat final est en termes des éléments architecturaux et les éléments (les fichiers) qui les composent.

Concernant un outil qui supporte le processus d'extraction, Bowman et al. proposent d'utiliser (1) l'outil cfx pour extraire les fonctions et leurs appels, et (2) l'outil grok pour extraire les relations entre les fichiers selon les appels des fonctions.

TABLE 3.8 : L'évaluation de l'approche de Rajalakshmi selon notre modèle de qualité.

Facteur	**Critère**	**Métrique**	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Intégration moyenne, en suivant les deux stratégies	MY
	Interaction de l'architecte	Permet seulement le raffinement des artefacts	MY
	Types des informations	Différents types	B
	Groupement des entités logicielles	Groupement plusieurs fois	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Pas de correspondance	MV
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Ne permet pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Supporte plusieurs groupement des entités logicielles	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

3.3.10 L'évaluation de l'approche ARM selon notre modèle de qualité

Guo et al. [Guo et al., 1999] ont proposé la méthode d'extraction ARM basée sur l'outil Dali pour reconstruire l'architecture du système en termes de patrons *(design pattern)*. La table 3.10 résume la projection des travaux de Guo selon notre modèle de comparaison. La méthode intègre les connaissances de l'architecte dans le processus en lui permettant de proposer une architecture conceptuelle. De même, elle lui permet de raffiner cette architecture et le résultat obtenu en modifiant l'architecture conceptuelle saisie sous forme de requêtes. Concernant les types des informations, les types sont les fichiers, les classes et fonctions avec les relations entre eux. Le résultat final est en termes des éléments architecturaux (qui respectent un patron) et les éléments qui les composent.

TABLE 3.9 : L'évaluation de l'approche de Bowman et al. selon notre modèle de qualité.

Facteur	**Critère**	**Métrique**	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Proposition d'une architecture conceptuelle et d'autres informations	B
	Interaction de l'architecte	Seulement raffienemt des artefacts	MY
	Types des informations	pas trop significatives (fichiers physiques)	MV
	Groupement des entités logicielles	Pas de groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance avec identification de la méthode	B
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Ne permet pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	un seul langage de programmation	MY
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

TABLE 3.10 : L'évaluation de l'approche ARM selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Seulement proposition d'une architecture conceptuelle	MY
	Interaction de l'architecte	Seulement raffinement des artefacts	MV
	Types des informations	signification moyenne	MY
	Groupement des entités logicielles	Pas de groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance avec identification de la méthode	B
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Seulement proposition d'une architecture conceptuelle	MY
	Interaction de l'architecte	Supporte le raffinement des artefacts	MY
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Supporte la correspondance	B

3.3.11 L'évaluation de l'approche de ReflexionModel selon notre modèle de qualité

Cette approche, comprenant un processus d'extraction supporté par un outil, reconstruit l'architecture en établissant une correspondance entre une architecture de haut-niveau et un modèle source extrait du code source. La table 3.11 résume la projection de cette approche selon notre modèle de comparaison. La méthode [Murphy et al., 2001] intègre les connaissances de l'architecte dans le processus en lui permettant de proposer un modèle de haut niveau et définir un plan (map) entre les deux modèles. Le processus est exécuté d'une manière statique ne supportant pas l'interaction de l'architecte avec les artefacts manipulés. Concernant les types des informations, les types sont les fichiers du code source. Le résultat final est en termes des éléments architecturaux et les éléments qui les composent.

Le processus de RelexionModel est supporté par un outil [Murphy et al., 2001] chargé de l'extraction d'un plan et l'exécution du plan sur le modèle source. Cet outil est limité par l'exécution des deux étapes qui sont l'extraction d'un modèle source et l'extraction d'une architecture selon le plan. Alors, il supporte l'intégration des connaissances de l'architecte sans lui

permettre d'interagir avec le processus et raffiner les artefacts.

TABLE 3.11 : L'évaluation de l'approche de ReflexionModel selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Proposition d'une architecture conceptuelle et d'autres infirmations	B
	Interaction de l'architecte	Pas d'interaction de l'architecte	MV
	Types des informations	signification moyenne	MY
	Groupement des entités logicielles	Pas de groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance avec identification de la méthode	B
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Supporte la proposition d'une architecture conceptuelle et d'autres infirmations	B
	Interaction de l'architecte	Supporte le raffinement des artefacts	MY
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle ou standard	MY
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	Plusieurs types de programmation	B
	Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Supporte la correspondance	B

3.3.12 L'évaluation de l'approche Focus selon notre modèle de qualité

Ding et al. [Ding et Medvidovic, 2001] ont proposé une méthode d'extraction qui est Focus. Cette méthode est basée sur le principe d'un groupement des entités logicielles et la correspondance des groupes à une architecture conceptuelle proposée par l'architecte. La table 3.12 résume la projection des travaux de Ding et al. selon notre modèle de comparaison. L'approche Focus intègre les connaissances de l'architecte en lui permettant de proposer une architecture conceptuelle, choisir un style architectural et établir la correspondance des groupes d'entités à l'architecture conceptuelle. Concernant l'interaction, la méthode est quasi-manuelle ; donc il y a une forte interaction avec le processus. Les informations extraites ont une signification moyenne ; elles sont des classes et leurs relations. Le résultat final est une architecture en termes d'éléments architecturaux et les éléments qui les composent.

TABLE 3.12 : L'évaluation de l'approche Focus selon notre modèle de qualité

Facteur	Critère	Métrique	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Proposition d'une architecture conceptuelle et d'autres informations	B
	Interaction de l'architecte	Interaction de l'architecte	B
	Types des informations	Signification moyenne	MY
	Groupement des entités logicielles	Un seul groupement	MY
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance sans identification de la méthode	MY
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Ne permet pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Selon un modèle	B
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	type orienté objet	MY
	Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

L'approche Focus est quasi-manuelle, il y a utilisation d'un seul outil qui est Rational Rose pour extraire les classes et leurs relations selon la nutation UML. Cet outil supporte seulement les systèmes en programmation orienté objet.

3.3.13 L'évaluation de l'approche de Medvidovic et al. selon notre modèle de qualité

Medvidovic et al. [Medvidovic et al., 2003] ont proposé une méthode d'extraction qui est basée sur le principe d'un groupement des entités logicielles et la correspondance des groupes à une architecture conceptuelle proposée par l'architecte. La table 3.13 résume la projection des travaux de Ding et al. selon notre modèle de comparaison. Cette approche est très semblable à Focus, ce qui la distingue est que la correspondance des groupes d'entités à l'architecture conceptuelle est en utilisant un style architectural. Ainsi, cette approche intègre les connaissances de l'architecte en lui permettant de proposer une architecture conceptuelle, choisir un style architectural et établir la correspondance des groupes d'entités à l'architecture conceptuelle. Concernant l'in-

TABLE 3.13 : L'évaluation de l'approche de Medvidovic et al. selon notre modèle de qualité.

Facteur	Critère	Métrique	Q
Méta-modèle	-	-	-
Méthode	Intégration des connaissances de l'architecte	Proposition d'une architecture conceptuelle et d'autres informations	B
	Interaction de l'architecte	Interaction de l'architecte	B
	Types des informations	Signification moyenne	MY
	Groupement des entités logicielles	Un seul groupement	MY
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance sans identification de la méthode	MY
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et les éléments qui les composent	MY
Outil	Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte	MV
	Interaction de l'architecte	Ne permet pas l'interaction de l'architecte	MV
	Structure extraite	Structure statique	MY
	Représentation des informations	Selon un modèle	B
	Langages de programmation supportés	Plusieurs langages de programmation	B
	Types de programmation supportés	type orienté objet	MY
	Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles	MV
	Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance	MV

teraction, la méthode est quasi-manuelle ; donc il y a une forte interaction avec le processus. Les informations extraites ont une signification moyenne ; elles sont des classes et leurs relations. Le résultat final est une architecture en termes d'éléments architecturaux et les éléments qui les composent.

L'approche de Medvidovic et al. est quasi-manuelle, il y a utilisation d'un seul outil qui est Rational Rose pour extraire les classes et leurs relations selon la nutation UML. Cet outil supporte seulement les systèmes en programmation orienté objet.

3.3.14 L'évaluation des outils selon notre modèle de qualité

Concernant les outils d'extraction, leurs projections sur le modèle de comparaison sont mentionnées pour la plupart d'entre eux dans les analyses présentées dans les deux sections précédentes. Ainsi, parmi les outils qui n'ont pas été mentionnés, on trouve les extracteurs de faits : l'outil Columbus [Ferenc et al., 2002, Ferenc et al., 2004, Beszédes et al., 2005], l'outil Gaget [Gargiulo et Mancoridis, 2001] et Ptidej [Guéhéneuc, 2004]. Etant donné que le rôle de ces outils

TABLE 3.14 : L'évaluation des outils Columbus, Gadget et Ptidej selon notre modèle de comparaison.

Critères	Columbus	Gadget	Ptidej
Intégration des connaissances de l'architecte	Ne permet pas l'intégration des connaissances de l'architecte (MV)	Ne permet pas l'intégration des connaissances de l'architecte (MV)	Ne permet pas l'intégration des connaissances de l'architecte (MV)
Interaction de l'architecte	Ne permet pas l'interaction de l'architecte (MV)	Ne permet pas l'interaction de l'architecte (MV)	Ne permet pas l'interaction de l'architecte (MV)
Structure extraite	Structure statique (MY)	Structure dynamique (B)	Structure dynamique (B)
Représentation des informations	Selon le schéma Columbus (B)	Pas selon un modèle (MY)	Selon le modèle PADL (B)
Langages de programmation supportés	Un seul langage de programmation (MY)	Un seul langage de programmation (MY)	Un seul langage de programmation (MY)
Types de programmation supportés	type orienté objet (MY)	type orienté objet (MY)	type orienté objet (MY)
Groupement des entités logicielles	Ne supporte pas le groupement des entités logicielles (MV)	Ne supporte pas le groupement des entités logicielles (MV)	Ne supporte pas le groupement des entités logicielles (MV)
Correspondance des entités logicielles à l'architecture conceptuelle	Ne supporte pas la correspondance (MV)	Ne supporte pas la correspondance (MV)	Ne supporte pas la correspondance (MV)

est limité par l'extraction d'un modèle source, nous les projetons brièvement dans la table 3.14 sur le facteur d'outil de notre modèle de comparaison.

3.4 La synthèse

Aucune approche ne permet de couvrir à la fois tous les facteurs de qualité que nous avons défini dans notre modèle de comparaison basé sur McCall. Par la suite, malgré les avantages apportés par les approches étudiées dans le domaine d'extraction d'une architecture logicielle, ces approches souffrent de plusieurs limitations dont les principales sont les suivantes :

1. Au niveau des méta-modèles d'extraction d'une architecture logicielle
   Un méta-modèle de bonne qualité spécifie les processus d'extraction en termes des niveaux, activités, artefacts et agents. Bien que les modèles conceptuels ont défini les nivaux sur lesquels opèrent un processus d'extraction, la plupart de ces modèles n'ont spécifié aucune activité, artefact ou rôle qui concernent le principe d'extraction. Comme exception, Chardigny [Chardigny, 2009], Ducasse et Pollet [Ducasse et Pollet, 2009] ont défini certains concepts mais qui sont très généraux pour détailler les processus d'extraction.

2. Au niveau des méthodes d'extraction

Dans les méthodes d'extraction, il n'y a aucune approche permettant à la fois :

- La conciliation des informations extraites avec une architecture conceptuelle d'une manière quasi-automatique
  Les travaux d'extraction existants ne permettent pas d'intégrer l'utilisation de l'architecture conceptuelle (qui est selon l'intention de l'architecte) avec un bon niveau d'automatisation. Alors, ces approches sont loin d'être appliquées sur des systèmes complexes.
- L'extraction d'une architecture complète
  Les approches existantes se focalisent sur l'extraction d'une architecture logicielle en termes d'éléments architecturaux et les entités qui les composent. Cependant, l'architecture logicielle doit être en termes d'éléments architecturaux, leurs interactions et les propriétés des éléments.
- L'intégration des connaissances de l'architecte durant l'exécution du processus.
  L'intégration des connaissances de l'architecte ultérieurement durant l'exécution du processus guide le processus vers une architecture conforme à leurs souhaits [Rajalakshmi, M, 2014]. Néanmoins, il n'y a aucune approche qui intègre effectivement les connaissances de l'architecte. Premièrement, les approches basées sur la stratégie de conciliation prennent seulement les connaissances de l'architecte sous forme d'une architecture conceptuelle. Deuxièmement, concernant les approches basées sur la stratégie de partitionnement, Rajalakshmi [Rajalakshmi, M, 2014] et [Mahdavi et al., 2003a] ont permis l'intégration des connaissances sous forme de décisions pris par l'architecte. Cependant, ces approches ne prennent pas en compte une architecture conceptuelle proposée par l'architecte et ne considèrent pas d'autres informations de sa part. Et finalement, les approches basées sur la stratégie de conciliation prennent en compte les connaissances de l'architecture mais sans support d'un outil .
- L'interaction d'un architecte avec le processus.
  Généralement, l'interaction est établie dans les processus existants en deux cas : le premier cas est lorsque le processus est quasi-manuel. Le deuxième cas est lorsque le processus est décomposé en une séquence d'étapes tout en permettant à l'architecte de raffiner les inputs et outputs de chaque étape. Cependant, cette interaction n'est pas supportée par des outils permettant à l'architecte de découvrir les artefacts manipulés, de les modifier et d'essayer de les comprendre.

3.5 Conclusion

Dans ce chapitre nous avons proposé un modèle de qualité qui recense trois facteurs de qualité : Méta-modèle, Méthode et Outil ; ces facteurs sont décomposés successivement en cinq critères de qualité d'un méta-modèle, six critères de qualité d'une méthode et huit critères de qualité d'un outil. En se basant sur ce modèle, nous avons évalué les différentes approches d'extraction d'une architecture logicielle. En addition, nous avons souligné à la fin du chapitre les limitations des approches d'extraction d'une architecture logicielle.

II

Notre contribution

4

Le méta-modèle SArEM

4.1 Introduction

Dans les chapitres précédents, nous avons effectué une synthèse bibliographique dont une partie porte sur les méta-modèles des processus d'extraction d'une architecture logicielle. L'objectif de ce chapitre est de présenter un nouveau méta-modèle qui spécifie les processus d'extraction d'une architecture logicielle et qui répond aux limitations des méta-modèles existants présentés au chapitre précédent. Au début, nous commençons par la définition des objectifs de notre travail. Ensuite, nous présentons le méta-modèle SPEM sur lequel notre approche est basée. Après, notre méta-modèle, nommé SArEM *(Software Architecture Extraction Meta-model)*, est détaillé. Plus spécifiquement, la méthode adoptée pour son élaboration et ses concepts sont présentés. Puis, nous fournissons la validation du méta-modèle SArEM tout en étudiant sa conformité avec les processus d'extraction existants. A la fin, nous synthétisons notre approche en évaluant SArEM selon le modèle de comparaison basé sur McCall et soulignant les apports de notre travail.

4.2 Positionnement et motivations de notre approche

Notre travail se situe dans le cadre des études concernant les processus d'extraction d'une architecture logicielle et la proposition d'un méta-modèle qui spécifie ces processus. Autrement dit, notre objectif est de proposer un méta-modèle qui détaille les concepts utilisés dans les processus d'extraction d'une architecture logicielle [Abboud et al., 2016a, Abboud et al., 2016b]. L'idée de proposer un tel méta-modèle a été inspiré de SPEM qui est un méta-modèle dédié à la spécification des processus de développement d'un logiciel [OMG et Notation, 2008]. En effet, comme le processus de développement est crucial dans le cycle de vie d'un logiciel, les processus de développement des systèmes ont gagné de plus en plus d'importance dans les industries des logiciels. Par conséquent, devant une pléthore des méthodes et processus de développement, les entreprises ont reconnu la nécessité d'une manière qui permet la compréhension, l'évaluation, l'amélioration et/ou l'exécution des procédés de développement d'un logiciel. Ainsi, pour répondre à ce besoin le groupe OMG et al. ont proposé le noyau SPEM.

D'une manière similaire aux processus de développement des systèmes, notre objectif est de capturer les processus d'extraction d'une architecture logicielle par un méta-modèle. Ainsi,

un tel méta-modèle forme un cadre formel pour les processus d'extraction d'une architecture logicielle et facilite à l'architecte la compréhension, l'évaluation, l'amélioration et/ou l'exécution des processus d'extraction d'une architecture logicielle. Pour atteindre un tel objectif, l'élaboration de notre méta-modèle a pris en considération ces deux points : Le premier point est la constitution d'un méta-modèle basé sur d'autres standards. En effet, parmi les éléments qui aident à avoir un cadre formel (standard) d'une chose est de se baser sur un autre standard. L'élaboration de notre méta-modèle est fondée sur le méta-modèle SPEM. Plus spécifiquement, notre méta-modèle est une extension du méta-modèle SPEM ; il hérite les définitions des éléments principaux de SPEM et les étend par des nouveaux éléments spécifiques à l'extraction qui seront présentés plus loin. Le deuxième point est la couverture des concepts d'extraction existants dans les processus d'extraction d'une architecture logicielle. En fait, un méta-modèle dédié à une certaine chose doit capturer tous les concepts pertinents à cette chose. Pour cette fin, l'élaboration de notre méta-modèle a été fondée sur l'étude des concepts existants dans les processus d'extraction d'une architecture logicielle les plus connus et l'unification de ces concepts. En fusionnant ces deux points, notre méta-modèle a alors hérité des concepts de SPEM d'une part, et d'autre part, il est étendu par d'autres concepts qui sont déduits à partir des concepts existants dans les processus d'extraction d'une architecture logicielle les plus connus.

Un méta-modèle élaboré d'une telle manière peut être considéré un outil conceptuel pour l'analyse, la comparaison, et l'évaluation des différentes approches d'extraction d'une architecture logicielle. En addition, à notre avis, le méta-modèle peut apporter des impacts réels sur les futurs travaux de recherche vu qu'il peut jouer les rôles suivants :

- Une référence pour la proposition de nouveaux processus d'extraction d'une architecture logicielle. En effet, dans la plupart des domaines, lors de la proposition d'une nouvelle approche, les auteurs essaient de suivre un certain standard afin d'atteindre un niveau souhaitable de leur approche. Un tel méta-modèle peut être considéré comme une référence qui précise ce que doit être mis dans une approche d'extraction d'architecture logicielle. Une telle référence guide alors les chercheurs dans la création des processus d'extraction tout en leur précisant les ingrédients qui doivent exister dans un processus d'extraction.

- Un cadre formel pour la description des processus d'extraction d'une architecture logicielle. Un tel méta-modèle qui couvre presque tous les concepts importants de l'extraction aide à décrire avec le même formalisme tous les processus d'extraction.

4.3 Le méta-modèle SPEM

Par définition, SPEM *(Software & Systems Process Engineering Meta-Model)* est un méta-modèle pour la spécification des processus de développement des systèmes [OMG et Notation, 2008] ; il est considéré un cadre formel (standard) pour les processus de développement des logiciels en fournissant les concepts nécessaires à la modélisation. Autrement dit, SPEM propose des concepts qui permettent de décrire tout type des processus de développement d'un logiciel.

4.3.1 Les entités définies par SPEM

Comme montre la figure 4.1, SPEM est composé de trois entités qui définissent les processus de l'ingénierie des systèmes.

- **L'entité «activité» :** Cette entité définit toute unité de travail qui doit être réalisée pour atteindre l'objectif du processus.

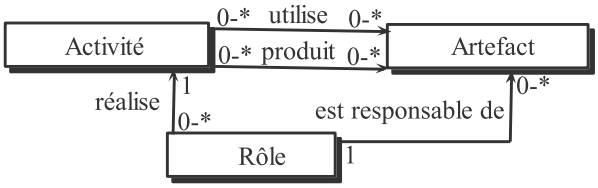

FIGURE 4.1 : Le méta-modèle du SPEM.

- **L'entité «artefact» :** Cette entité définit tout objet concret capable d'être manipulé. Un artefact peut être utilisé, modifié et/ou produit par les activités.
- **L'entité «rôle» :** Cette entité représente les compétences et les responsabilités requises pour la réalisation d'activité. Un rôle exécute l'activité lors de l'utilisation des artefacts.

4.3.2 Les relations entre les entités définies par SPEM

Selon SPEM, un processus de développement logiciel est une collaboration entre des rôles exécutant des activités sur des produits. Ainsi, SPEM est composé des relations qui sont définies comme suivant :

- **La relation «utilise» :** Cette relation indique qu'un artefact est utilisé par une activité.
- **La relation «produit» :** Cette relation indique qu'un artefact est produit par une activité.
- **La relation «exécute» :** Cette relation indique qu'une activité est effectuée par un rôle.
- **La relation «est responsable de» :** Cette relation indique qu'un rôle est responsable d'un artefact.

4.4 La méthode adoptée pour l'élaboration du méta-modèle SArEM

L'élaboration du SArEM a suivi un processus composé de plusieurs activités qui sont présentées à la figure 4.2 et résumées ci-dessous :

Activité 1 : L'extension du méta-modèle SPEM. Cette activité consiste à produire une extension par héritage du méta-modèle SPEM. Ainsi l'extension élaborée est composée de 3 entités et 4 relations. Cette extension, qui est une partie du méta-modèle SArEM, est présentée dans la section 4.5.

Activité 2 : L'élaboration des concepts d'extraction d'une architecture logicielle. Cette activité comporte deux étapes : La première est l'identification des approches d'extraction les plus citées. Et la deuxième, l'élaboration du modèle de processus de chaque approche d'extraction. Cette deuxième étape consiste à souligner les éléments importants de chaque approche et leurs relations et à les exprimer sous formes d'entités avec des relations entre elles. Ainsi, à la fin de cette étape, un méta-modèle en termes d'entités et des relations est élaboré pour chaque approche.

Activité 3 : L'organisation des méta-modèles des approches selon le modèle SPEM. Cette activité consiste à classifier les entités des modèles élaboré à l'activité 2 sous trois catégories : Activité, Artefact et Rôle. De plus, l'activité consiste à établir des relations («utilise», «produit», «exécute», «est responsable de») entre les entités.

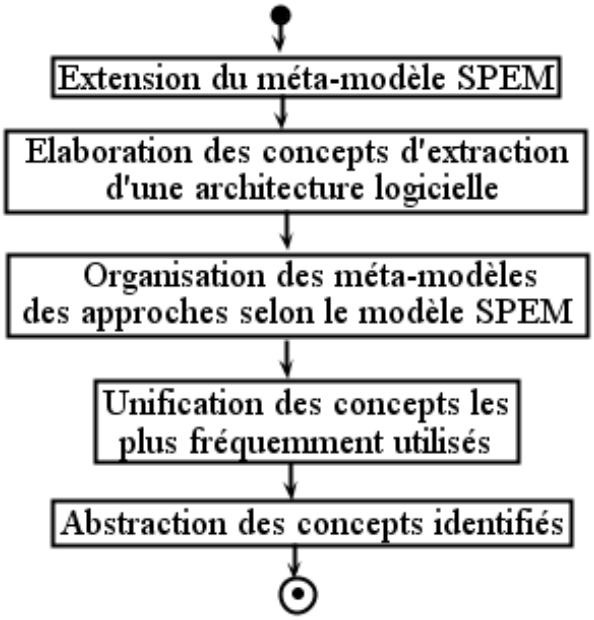

FIGURE 4.2 : Les activités réalisées pour l'élaboration du méta-modèle SArEM.

Activité 4 : L'unification des concepts les plus fréquemment utilisés. Cette activité consiste à souligner les activités, les rôles et les artefacts classifiés les plus fréquemment répétés, et l'unification de ces concepts. Cette activité a produit une partie du méta-modèle SArEM présenté dans la section 4.6.

Activité 5 : L'abstraction des concepts identifiés. Cette étape consiste à abstraire les concepts identifiés dans l'activité 4 en les généralisant. Cette abstraction a élaboré des nouvelles entités qui concilient la différence entre les concepts identifiés dans l'activité 4 (qui sont des concepts très concrets) et les concepts hérités du SPEM (qui sont des concepts très généraux). Cette activité a produit une partie du méta-modèle SArEM présenté dans la section 4.6.

4.5 Une extension du méta-modèle SPEM pour l'extraction des architectures logicielles

Cette section présente une extension du méta-modèle SPEM. Cette extension hérite les concepts du méta-modèle SPEM. Ainsi, trois entités et quatre relations sont définies dans l'extension. Ces entités avec leurs relations, illustrées à la figure 4.3, définissent les concepts de base de l'extraction d'une architecture logicielle. Ci-dessous, nous expliquons le besoin de ces concepts dans un méta-modèle spécifiant les processus d'extraction d'une architecture logicielle.

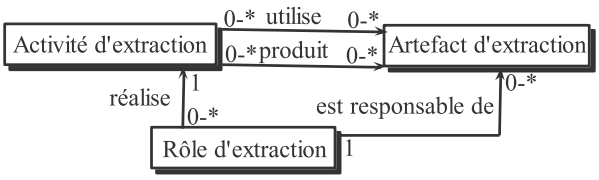

FIGURE 4.3 : Une extension du méta-modèle SPEM.

4.5.1 Les entités définies dans l'extension

L'extension hérite les entités du SPEM. Les entités de l'extension sont :

- **L'entité «activité d'extraction» :** La définition d'un processus s'intéresse principalement à l'identification de la séquence des activités qui doit être établie pour achever le but du processus. D'où, la présence du principe «activité d'extraction» dans une norme dédiée au processus d'extraction. Nous définissons une activité d'extraction comme le travail qui doit être effectué pour extraire une architecture.

- **L'entité «artefact d'extraction» :** L'architecture logicielle ne peut pas être produite sans éléments de base tels que le code source, les documents système et sans la manipulation de ces éléments. Ainsi, l'entité artefact d'extraction est requise pour modéliser les processus d'extraction. Nous considérons un artefact d'extraction comme un objet concret qui peut être manipulé par une activité d'extraction durant le processus d'extraction.

- **L'entité «rôle d'extraction» :** Une activité d'extraction doit être exécutée par un interprète tel qu'un outil ou un utilisateur. Ainsi, nous définissons dans notre méta-modèle l'entité «rôle d'extraction» qui représente une compétence capable d'exécuter une activité d'extraction.

4.5.2 Les relations définies dans l'extension

L'extension hérite les relations du SPEM. Les relations de l'extension sont : «utilise», «produit», «exécute», «est responsable de».

- **La relation «utilise» :** Cette relation indique qu'un artefact d'extraction est utilisé par une activité d'extraction.

- **La relation «produit» :** Cette relation indique qu'un artefact d'extraction est produit par une activité d'extraction.

- **La relation «exécute» :** Cette relation indique qu'une activité d'extraction est effectuée par un rôle d'extraction.

- **La relation «est responsable de» :** Cette relation indique qu'un rôle d'extraction est responsable d'un artefact d'extraction.

4.5.3 L'insuffisance de l'extension du SPEM

L'extension présentée dans les sections 4.5.1 et 4.5.2 décrit les concepts d'extraction de l'architecture logicielle d'une manière très générale. Elle ne définit que 3 éléments généraux avec leurs relations. Cette généralité ne permet pas à un architecte de comparer les approches existantes. En fait, un méta-modèle qui aide un architecte à comparer plusieurs approches d'extraction devrait contenir des éléments plus spécifiques dédiés à l'extraction. Pour atteindre cet objectif, plus d'éléments qui spécifient les concepts d'extraction doivent être ajoutés aux 3 entités et 4 relations existants dans l'extension. Ainsi, cette première extension du méta-modèle SPEM doit être encore étendue par des concepts spécifiques à l'extraction des architectures logicielles. La section suivante présente la nouvelle extension, nommée SArEM *(Software Architecture Extraction Meta-model)*, qui est enrichie par des concepts dédiés au processus d'extraction d'une architecture logicielle.

4.6 Le méta-modèle SArEM

L'enrichissement de l'extension du SPEM consiste à spécifier chaque concept en des sous-concepts pour donner plus de détails. Un concept est spécifié en sous-concepts par deux manières :

1. Par héritage : Dans ce cas, le sous-concept est une spécialisation du concept.
2. Par composition : Dans ce cas, le sous-concept entre dans la composition du concept.

De plus, l'enrichissement n'est pas limité à l'ajout des concepts dédiés à l'extraction, mais aussi il définit les relations qui existent entre eux. Trois enrichissements sont établis, formant ainsi trois extensions : l'extension de l'activité d'extraction, l'extension de l'artefact d'extraction et l'extension du rôle d'extraction. Ainsi, nous présentons dans ce qui suit ces trois extensions qui constituent SArEM. Puis, nous détaillons dans la section 4.7 comment les entités de l'extension sont reliées entre eux.

4.6.1 Une extension de l'entité artefact d'extraction

L'extension de l'entité artefact d'extraction, illustrée à la figure 4.4 est établie sur 2 niveaux : Le premier niveau, qui est le niveau 2 de la figure 4.4, classe les artefacts d'extraction sous 4 types : L'artefact initial, l'artefact intermédiaire, la pseudo-architecture et l'artefact final. Le deuxième niveau, qui est le niveau 3 de la figure 4.4, spécialise chaque type d'artefact à des sous-types. Ces deux niveaux sont les résultats des activités 4 et 5 mentionnées dans la section 4.4. Plus précisément, l'activité 4 (unification des concepts les plus fréquemment utilisés) a aidé l'élaboration du niveau 3 du méta-modèle SArEM et l'activité 5 (abstraction des concepts identifiés) a élaboré le niveau 2.

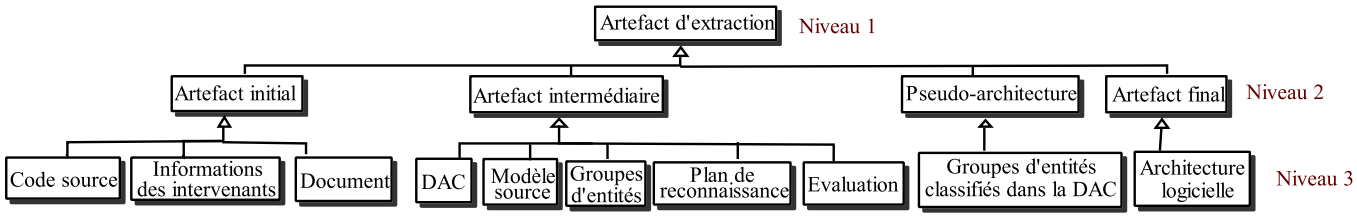

FIGURE 4.4 : Les artefacts d'extraction de SArEM

Les entités du niveau 2 : Les types de l'artefact d'extraction

Les types de l'artefact d'extraction sont des artefacts d'extraction. Ainsi, une relation d'héritage existe entre les types d'artefacts d'extraction et l'entité d'artefact d'extraction. Ces types d'artefacts d'extraction sont :

- **L'artefact initial :** Un artefact initial définit tout artefact qui existe avant l'exécution du processus d'extraction.
- **L'artefact intermédiaire :** Un artefact intermédiaire définit les artefacts manipulés avant la génération d'une architecture.
- **La pseudo-architecture :** Une pseudo-architecture définit la première architecture générée.
- **L'artefact final :** Un artefact final définit l'artefact résultant de tout le processus d'extraction, il est l'output définitif du processus d'extraction.

Les entités du niveau 3 : Les sous-types de l'artefact d'extraction

Les sous-types de l'artefact d'extraction sont des artefacts d'extraction. Ainsi, une relation d'héritage existe entre les entités du niveau 2 et celles du niveau 3. Ces sous-types d'artefacts d'extraction sont les suivants :

1. **Les sous-types de l'artefact initial**
   Le code source : Le code source est l'implémentation du système. Il est représenté par les fichiers, les lignes du code et même les commentaires du codage.
   Les informations des intervenants : Une information est une idée acquise par les intervenants grâce à leurs expériences, expertises et connaissances à propos du système.
   Le document : Un document représente toute description contenant des informations reliées au système.

2. **Les sous-types de l'artefact intermédiaire**
   La description de l'architecture conceptuelle (DAC) : Cet artefact est une description de l'architecture du système à haut-niveau en termes de blocs et des relations entre eux.
   Le modèle source : Un modèle source est une représentation du code source. Il représente le code source en termes d'entités et des relations entre ces entités. Par exemple, un modèle source pourrait être un diagramme de classes ou pourrait être un graphe qui représente les fichiers du système et leurs relations.
   Les groupes d'entités : Un groupe d'entités est un ensemble des entités du modèle source. Ainsi, les groupes d'entités représentent une partition des entités du modèle source.
   Le plan de reconnaissance : Cet artefact est un plan qui indique comment la DAC est réalisée dans les groupes d'entités du modèle source.
   L'évaluation : Cet artefact représente l'avis des intervenants à propos de la pseudo-architecture. L'évaluation exprime si la pseudo-architecture leur satisfait ou non.

3. **Les sous-types de la pseudo-architecture**
   Groupes d'entités classifiés dans la DAC : Cet artefact définit la première architecture générée ; il représente les groupes d'entités classifiés dans les blocs de la DAC selon le plan de reconnaissance.

4. **Les sous-types de l'artefact final**
   L'architecture logicielle : Cet artefact représente l'architecture du système en termes d'éléments et des relations entre ces éléments.

4.6.2 Une extension de l'entité activité d'extraction

L'extension de l'entité activité d'extraction, illustrée à la figure 4.5, est établie sur 3 niveaux : Le premier niveau, qui est niveau 2 de la figure 4.5, classe les activités d'extraction sous deux types : les activités qui étudient les artefacts d'extraction et les analysent, appelées activités d'analyse et d'autres qui recueillent les résultats d'analyse afin de générer une architecture logicielle, appelées activités de synthèse. Le deuxième niveau, qui est niveau 3 de la figure 4.5, étend chaque type d'activité en 3 sous-types. Le dernier niveau, qui est au niveau 4 de la figure 4.5, étend chaque sous-type d'activité en précisant les activités qui le composent. Ces niveaux sont les résultats des activités 4 et 5 mentionnées dans la section 4.4. Plus précisément, l'activité 4 (unification des concepts les plus fréquemment utilisés) a aidé l'élaboration du niveau 4 du méta-modèle SArEM et l'activité 5 (abstraction des concepts identifiés) a élaboré les niveaux 2 et 3.

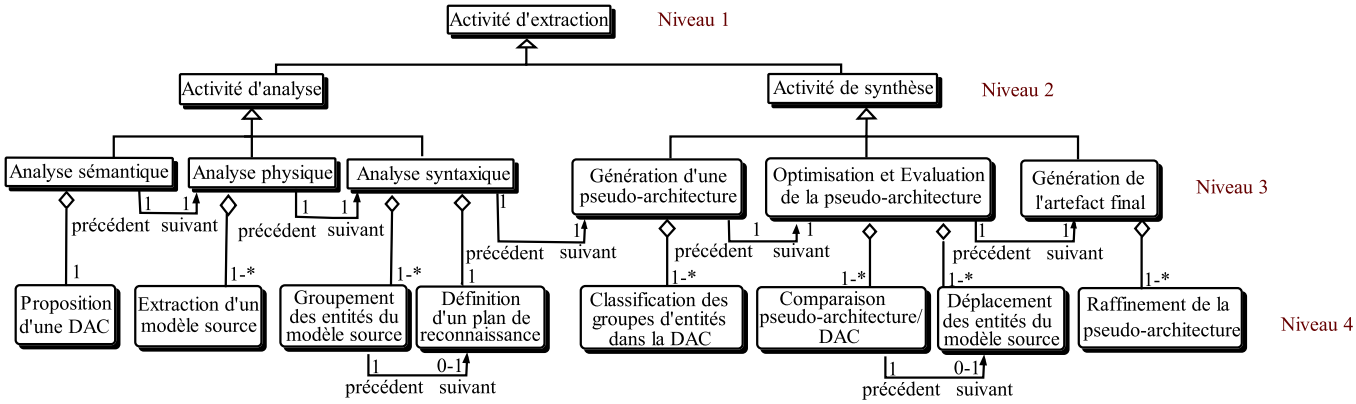

FIGURE 4.5 : Les activités d'extraction de SArEM

Les entités du niveau 3 : Les types de l'activité d'analyse et de l'activité de synthèse

1. **Les sous-types de l'activité d'analyse**

   Les types d'analyse consistent à étudier les artefacts afin de déduire des informations significatives nécessaires à l'élaboration d'une architecture logicielle. Ainsi, nous définissons dans notre méta-modèle trois types d'analyse :

   L'analyse sémantique : Cette analyse étudie le sens et les concepts présents dans les artefacts pris comme inputs.

   L'analyse physique : Cette analyse étudie les termes qui existent dans un artefact. Elle se base fondamentalement sur la récupération des informations des artefacts tout en consultant les termes qui existent dans ces artefacts.

   L'analyse syntaxique : Cette analyse étudie les relations entre les résultats de l'analyse sémantique et l'analyse physique. Elle analyse comment les concepts élaborés par l'analyse sémantique sont réalisés dans les termes élaborés par l'analyse physique.

2. **Les sous-types de l'activité de synthèse**

   Après la définition des activités d'analyse qui consistent à étudier les artefacts pris comme inputs d'une part et à fournir des artefacts utiles pour l'obtention de l'architecture d'autres part, nous avons remarqué la nécessité (1) des activités qui prennent ces résultats de l'analyse et génèrent une première architecture et (2) des activités qui essaient d'améliorer cette architecture pour obtenir à la fin l'architecture finale. Cette nécessité nous a conduit à définir des sous-types de l'activité de synthèse, qui sont les suivants :

   La génération d'une pseudo-architecture : Cette activité consiste à déduire une première architecture à partir des résultats de l'analyse.

   L'optimisation et l'évaluation de la pseudo-architecture : Cette activité désigne toute activité ayant pour but d'améliorer la pseudo-architecture. Elle prend la pseudo-architecture comme input et applique les opérations d'évaluation et d'amélioration.

   La génération de l'artefact final : Cette activité conclut par le dernier artefact souhaité. Elle prend la pseudo-architecture et ses évaluations et génère l'artefact final.

Les entités du niveau 4 : Les activités qui composent chaque sous-type d'activité

Afin de permettre une récupération efficace des processus d'extraction, on a proposé plus spécifiquement les activités qui composent chaque sous-type d'activité. Ainsi, une relation de composition existe entre les entités du niveau 3 et celles du niveau 4.

1. **Les activités définies sous les sous-types de l'activité d'analyse**

La proposition d'une description de l'architecture conceptuelle (DAC) : Cette activité compose l'analyse sémantique. Elle consiste à analyser les documents et les informations des intervenants du système afin de fournir une description de l'architecture du système en termes des blocs et des relations entre ces blocs, appelée description de l'architecture conceptuelle (DAC).
L'extraction d'un modèle source : Cette activité compose l'analyse physique. Elle consiste à analyser le code source dans le but d'extraire un modèle source qui représente les informations importantes du code source. Plusieurs modèles sources peuvent être extraits pour représenter les informations du code source.
Le groupement des entités du modèle source : Cette activité compose l'analyse syntaxique. Elle consiste à analyser les relations entre les entités du modèle source afin de grouper les entités. Ainsi, cette analyse utilise le modèle source et produit les groupes d'entités. Plusieurs groupements hiérarchiques peuvent être réalisés.
La définition d'un plan de reconnaissance : Cette activité compose aussi l'analyse syntaxique. Elle étudie les relations qui peuvent exister entre les groupes d'entités et la DAC. En d'autres termes, elle indique comment la DAC est réalisée dans les groupes d'entités du modèle source. Cette activité utilise les groupes d'entités et la DAC comme input pour générer le plan de reconnaissance qui sert comme manière de correspondance entre eux.

2. Les activités définies sous les sous-types de l'activité de synthèse
La classification des groupes d'entités dans la description de l'architecture conceptuelle (DAC) : Cette activité compose l'activité de génération d'une pseudo-architecture. Elle consiste à classifier les groupes d'entités dans les blocs de la DAC selon le plan de reconnaissance.
La comparaison de la pseudo-architecture avec la description de l'architecture conceptuelle (DAC) : Cette activité compose l'activité d'optimisation et d'évaluation de la pseudo-architecture. Elle consiste à comparer la pseudo-architecture avec la DAC et produire ainsi une évaluation indiquant si la pseudo-architecture répond à la DAC proposée par les intervenants.
Le déplacement des entités du modèle source : Cette activité compose aussi l'activité d'optimisation et d'évaluation de la pseudo-architecture. Elle consiste à déplacer les entités du modèle source d'un groupe d'entités à l'autre et/ou d'un bloc de la DAC à l'autre selon les résultats de l'évaluation.
Le raffinement de la pseudo-architecture : Cette activité compose l'activité de génération de l'artefact final. Elle consiste à raffiner la pseudo-architecture selon l'évaluation établie. Le raffinement de la pseudo-architecture peut être aussi établie en considérant un style architectural.

La séquence des activités

Un processus commence avec une activité, suit une séquence d'activités et se termine par une activité finale. Pour cela, nous voyons que l'indication de la suite qui doit être suivie par un processus d'extraction spécifie les processus d'extraction.

1. La séquence des activités d'analyse
Nous identifions la séquence des 3 types d'analyse en mettant la lumière sur l'objectif de chaque type d'analyse : L'analyse sémantique a un rôle d'analyser sémantiquement les artefacts, ce qui aide à prendre une idée sur l'architecture du système. Il sera alors important de commencer le processus par ce type d'analyse pour diriger le reste du processus. Les deux autres activités sont l'analyse physique et l'analyse syntaxique. Dans

l'analyse physique, le code est analysé pour extraire des informations de ce code. Tandis que, l'analyse syntaxique étudie ces informations et essaie de faire un lien entre les résultats de l'analyse sémantique et l'analyse physique. Ceci indique évidement que l'analyse physique doit être appliquée avant l'analyse syntaxique.

2. La séquence des activités de synthèse
La génération de la pseudo-architecture est établie tout en rassemblant les artefacts résultants de l'analyse, puis des opérations d'optimisation et d'évaluation sont réalisées pour arriver enfin à la génération de l'architecture finale.

4.6.3 Une extension des rôles d'extraction

L'extension des rôles d'extraction, illustré à la figure 4.6 est établie sur 2 niveaux : Le premier niveau, qui est niveau 2 de la figure 4.6, classe les rôles d'extraction sous 2 types : les intervenants du système (comme l'architecte et les développeurs) et les techniques. Ces dernières sont spécifiées au niveau 3 de la figure 4.6 ; il s'agit des outils ou des types de techniques.

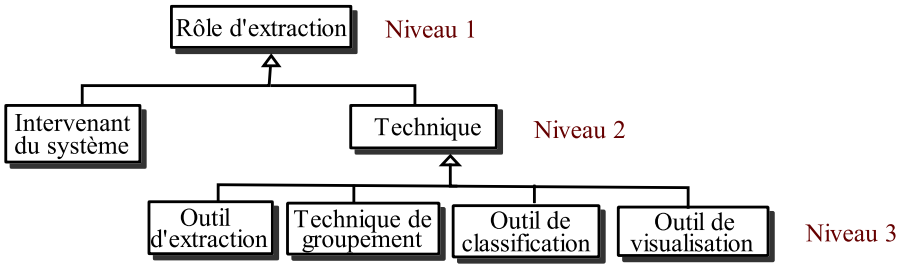

FIGURE 4.6 : Les rôles d'extraction de SArEM.

Les entités du niveau 3 : Les types du rôle technique

L'outil d'extraction : Cette entité définit tout outil qui effectue l'activité d'extraction du modèle source. Un outil d'extraction utilise le code source comme un input et fournit le modèle source en tant que sortie.
La technique de groupement : Cette entité définit toute technique capable de grouper des entités du modèle source selon certains critères. D'habitude, cette technique utilise une fonction objective qui conduit le groupement selon les critères souhaités.
L'outil de classification : Cette entité définit tout outil qui prend comme input les groupes d'entités, le plan de reconnaissance et la DAC et fournit comme output une classification des groupes d'entités selon le plan.
L'outil de visualisation : Cette entité définit tout outil qui aide à comprendre l'architecture logicielle grâce à une représentation graphique.

4.7 Les interactions entre les entités de SArEM

Les interactions spécifient comment les éléments se comportent entre eux dans un processus d'extraction. Nous présentons ces interactions en fonction des activités d'extraction. Nous présentons ainsi les activités de SArEM, les artefacts qu'elles utilisent et/ou produisent et les rôles qui les réalisent.

4.7.1 Le mécanisme opératoire des activités d'analyse de SArEM

L'analyse sémantique

L'analyse sémantique, composée de la proposition d'une DAC, utilise des artefacts initiaux et produit un artefact intermédiaire. La proposition d'une DAC est réalisée par les intervenants du système. Plus spécifiquement, grâce à son expertise et l'analyse des documents existants, l'architecte propose une DAC en termes de blocs et leurs relations (voir figure 4.7). Puis, chaque intervenant impliqué dans le développement du système examine les documentations qui couvrent son domaine d'expertise afin de vérifier la DAC proposée par l'architecte. Enfin, chaque perspective, une idée ou note proposée par les intervenants est discutée avec l'architecte. Le résultat final est une DAC qui réunit les idées des intervenants et forme une image globale de l'architecture du système.

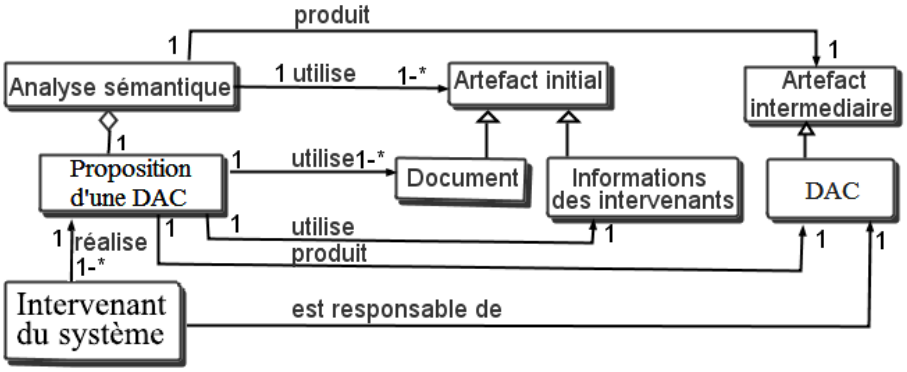

FIGURE 4.7 : Le mécanisme opératoire de l'analyse sémantique.

L'analyse physique

L'analyse physique, composée de plusieurs extractions d'un modèle source, utilise des artefacts initiaux et produit des artefacts de type artefact intermédiaire. Dans l'activité d'extraction d'un modèle source, un modèle source est élaboré à partir du code source en utilisant une technique d'extraction comme rôle d'extraction (voir figure 4.8). Cette extraction est effectuée en analysant automatiquement le code source ; elle est réalisée en vérifiant chaque mot présenté dans le code source pour former un modèle qui représente le code source.

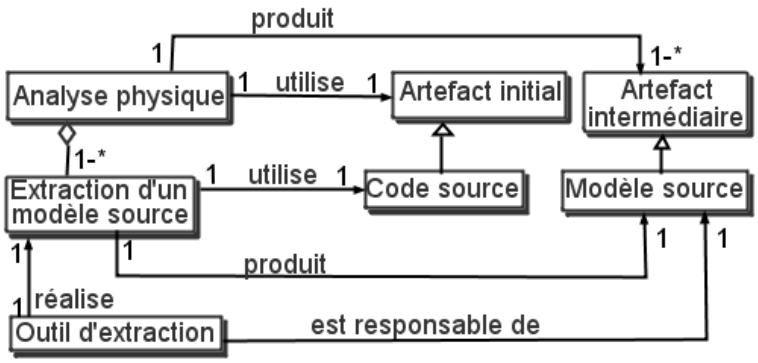

FIGURE 4.8 : Le mécanisme opératoire de l'analyse physique.

L'analyse syntaxique

L'analyse syntaxique, composée du groupement des entités du modèle source et de la définition d'un plan de reconnaissance, utilise et produit des artefacts intermédiaires (voir figure 4.9). L'activité du groupement des entités du modèle source utilise le modèle source comme input et produit des groupes d'entités comme output en utilisant un outil automatique. D'habitude, les algorithmes de recherche comme l'escalade de colline et l'algorithme génétique réalisent le groupement des entités en suivant une fonction qui dirige l'algorithme. Cette étape peut être réalisée d'une manière itérative où à chaque itération il y a passage à un niveau d'abstraction plus élevé. Concernant l'activité de définition d'un plan de reconnaissance, un plan qui relie les groupes d'entités à la DAC est produit comme output. Pour atteindre cet objectif, l'intervenant identifie quand un groupe d'entités est classifié dans un bloc de DAC. Ainsi, comme montre la figure 4.9, cette activité utilise les groupes des entités et la DAC afin de produire le plan de reconnaissance.

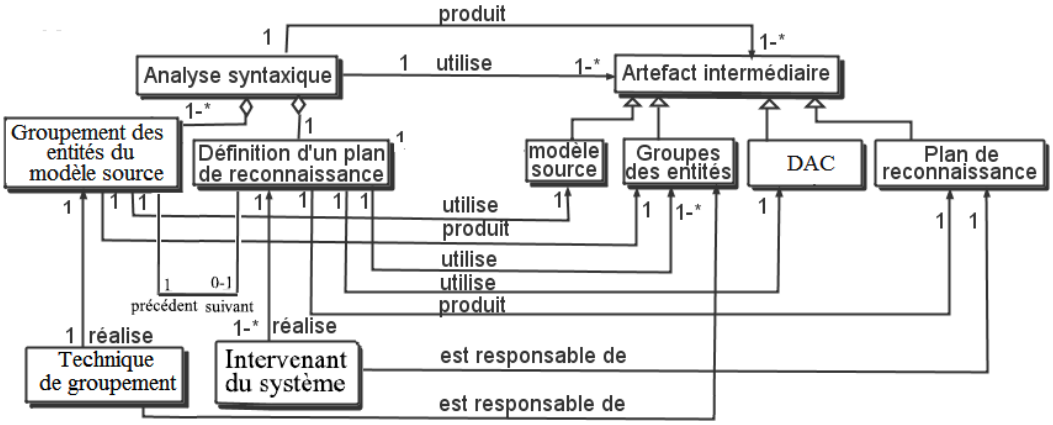

FIGURE 4.9 : Le mécanisme opératoire de l'analyse syntaxique.

4.7.2 Le mécanisme opératoire des activités de synthèse de SArEM

La génération d'une pseudo-architecture

La génération d'une pseudo-architecture, composée de la classification des groupes d'entités dans la DAC, utilise des artefacts intermédiaires et produit une première architecture nommée pseudo-architecture (voir figure 4.10). La classification des groupes d'entités dans la DAC classifie les groupes d'entités dans les blocs de la DAC selon le plan de reconnaissance. L'output de cette activité est une architecture qui présente les groupes d'entités classifiés dans la DAC. En outre, l'exécution est réalisée en utilisant un outil de classification.

L'optimisation et l'évaluation de la pseudo-architecture

L'optimisation et l'évaluation de la pseudo-architecture, composée de la comparaison de la pseudo-architecture avec la DAC et du déplacement des entités du modèle source, utilise les artefacts intermédiaires pour améliorer la pseudo-architecture (voir figure 4.11). Dans l'activité de comparaison, les intervenants comparent la pseudo-architecture avec la DAC pour produire une évaluation. Concernant le déplacement des entités du modèle source, les intervenants utilisent l'évaluation et modifient dans la classification des entités du modèle source.

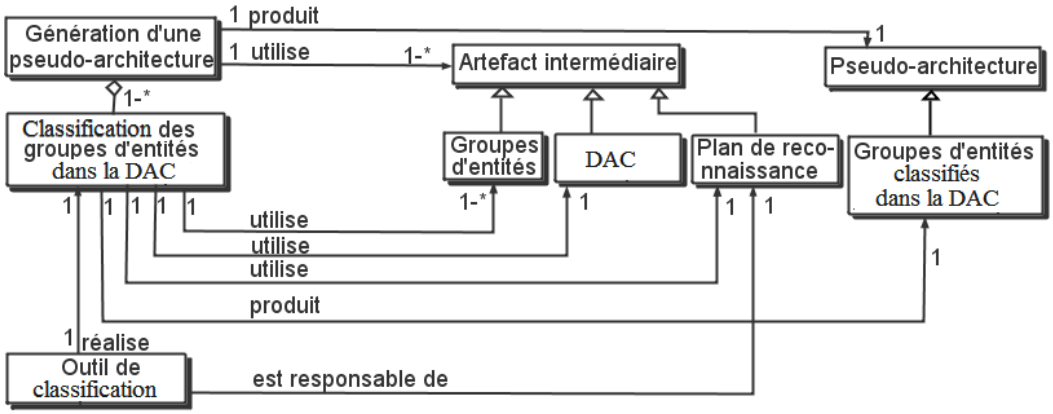

FIGURE 4.10 : Le mécanisme opératoire de la génération d'une pseudo-architecture.

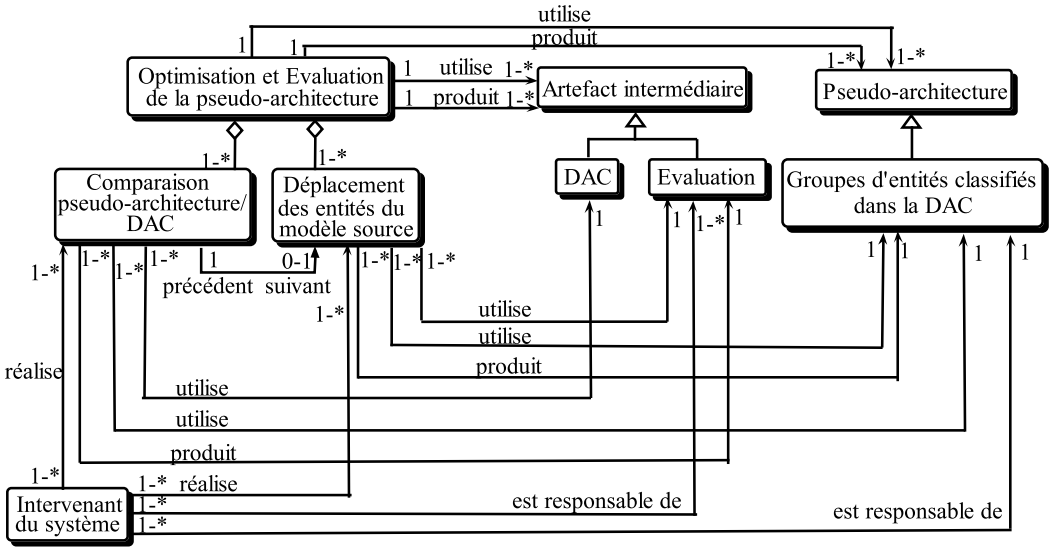

FIGURE 4.11 : Le mécanisme opératoire de l'optimisation et l'évaluation de la pseudo-architecture.

La génération de l'artefact final

La génération de l'artefact final, composée du raffinement de la pseudo-architecture, utilise l'évaluation et génère l'artefact final qui est l'architecture (voir figure 4.12). Les intervenants réalise l'activité à l'aide des outils de visualisation.

4.8 La validation de SArEM

Comme nous l'avons déjà mentionné, notre objectif est de développer un méta-modèle qui spécifie les processus d'extraction d'une architecture logicielle. Un tel méta-modèle doit respecter ou définir autant que possible les concepts d'extraction d'une architecture logicielle. Pour valider cette condition, nous devons valider que les éléments de SArEM en termes d'activités, artefacts et rôles, la séquence des activités de SArEM, les mécanismes opératoires des activités de SArEM sont définis dans la plupart des processus d'extraction des approches. Ainsi, nous présentons dans ce qui suit des études de conformité qui vérifient si SArEM a défini la plupart des concepts présents dans les processus d'extraction les plus connus dans la littérature.

Avant de réaliser l'étude de conformité de SArEM aux approches d'extraction, nous défi-

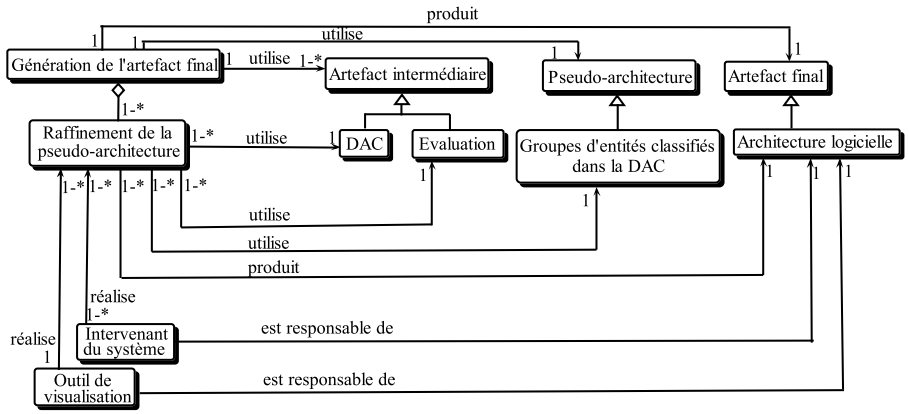

FIGURE 4.12 : Le mécanisme opératoire de la génération de l'artefact final.

nissons un ensemble de liens que nous appelons les liens de conformité. Ces liens sont utilisés pour exprimer si les éléments du SArEM sont conformes aux éléments d'une approche donnée ou non. Ainsi, la section 4.8.1 présente les liens de conformité définis, section 4.8.2 donne la méthodologie de conformité utilisée, et enfin la section 4.8.3 montre l'étude de conformité de SArEM aux approches d'extraction. Plus spécifiquement, nous détaillons l'étude de conformité de SArEM à deux approches d'extraction et nous fournissons un résumé de l'étude pour les approches les plus connues.

4.8.1 Les liens de conformité

L'étude de conformité de SArEM à une approche donnée revient principalement à l'étude de conformité des éléments de SArEM aux éléments de cette approche. Un élément SArEM est conforme à un élément d'une approche donnée si l'un des liens suivants peut être établi entre eux :

- **«équivaut à», noté $E_1 = E_2$:**
  Ce lien est déterminé entre un élément E_1 défini par SArEM et un élément E_2 de l'approche lorsque E_1 et E_2 représentent le même concept. Par exemple, l'activité «Proposition d'une DAC» de SArEM équivaut à l'activité «Création d'une architecture conceptuelle» de l'approche proposée par Bowman et al. (voir figure 4.16) : les deux activités ont le même but qui est la proposition d'une architecture à haut-niveau en termes de blocs et relations.

- **«composé de», noté $E_1 \blacklozenge\!\!- E_2$:**
  Ce lien est déterminé entre un élément E_1 défini par SArEM et un élément E_2 de l'approche lorsque E_2 réalise une partie de E_1. Par exemple, l'activité «Extraction d'un modèle source» défini par SArEM est composé de l'activité «Extraction des relations entre les fonctions et les variables» défini dans l'approche proposée par Bowman et al. (voir figure 4.16). En fait, l'activité «Extraction des relations entre les fonctions et les variables» consiste à extraire des informations du code source de types relations entre les fonctions et variables, ce qui est une partie de l'extraction d'un modèle qui représente le code source.

L'existence fréquente des liens de conformité entre les éléments de SArEM et les éléments d'une approche existante valide les éléments de notre méta-modèle SArEM.

4.8.2 La méthode d'étude de conformité

Pour étudier la conformité du méta-modèle SArEM à une approche donnée, nous avons établi les étapes suivantes :

Étape 1 : L'élaboration du méta-modèle de l'approche. Dans cette étape, un méta-modèle de l'approche est généré en soulignant les éléments importants de l'approche et leurs relations.

Étape 2 : L'organisation du méta-modèle de l'approche selon le modèle SPEM. Dans cette étape, les éléments élaborés à l'étape 1 sont classifiés sous forme d'entités en trois catégories : les artefacts d'extraction, les activités d'extraction et les rôles d'extraction. De plus, des relations («utilise», «produit», «exécute», «est responsable de») sont établies entre les entités.

Étape 3 : L'élaboration de la séquence des activités d'extraction de l'approche. Dans cette activité, des relations «précédent» et «suivant» sont établies entre les activités du méta-modèle. La figure 4.15 montre l'organisation des éléments de l'approche proposée par Bowman et al. selon le modèle SPEM et la séquence des activités en utilisant les relations «précédent» et «suivant».

Étape 4 : L'établissement des liens de conformité entre les éléments de SArEM et ceux de l'approche étudiée. Dans cette étape, les liens définis dans la section 4.8.1 sont établis. La figure 4.16 montre le résultat de cette étape pour l'approche proposée par Bowman et al.

Étape 5 : La classification des entités du méta-modèle de l'approche selon le mécanisme opératoire des activités de SArEM. Dans cette étape, les mécanismes opératoires des activités présents dans les approches sont étudiés. Ainsi, les entités du méta-modèle de l'approche sont classifiées selon les mécanismes opératoires des activités de SArEM. Par exemple, la ligne 1 de la table 4.1 montre la classification des entités de l'approche proposée par Bowman et al. selon le mécanisme opératoire de l'activité «Proposition d'une DAC».

4.8.3 La conformité de SArEM aux approches existantes

Dans ce qui suit, nous présentons en détail les études de conformité de SArEM à deux approches d'extraction. Nous avons choisi ces deux approches parmi la pléthore d'approches parce qu'elles sont les plus citées dans la littérature. Puis, nous présentons la conformité des éléments de SArEM et le mécanisme opératoire de ses activités à d'autres approches.

1. La conformité de SArEM à l'approche ARM

ARM [Guo et al., 1999] est une méthode semi-automatique qui reconstruit l'architecture des systèmes à base des patrons. La méthode a pour but de reconnaitre les patrons architecturaux du système (comme MVC, médiateur, etc.) afin de passer des informations au niveau du code source à un niveau d'abstraction plus élevé qui représente les patrons existants dans l'implémentation. Cette méthode est composée de 4 phases qui visent à (1) Développer un plan de reconnaissance, (2) Extraire un modèle qui représente le code source, (3) Détecter des instances de patrons tout en exécutant le plan développé sur le modèle source, et (4) Reconstruire et analyser les résultats(l'architecture). La figure 4.13 illustre les éléments de l'approche ARM classifiés sous trois catégories.

Premièrement, concernant la conformité des éléments de SArEM aux éléments de l'approche ARM, le résultat de l'étude de conformité est présenté à la figure 4.14. La proposition d'une DAC équivaut à l'extraction des règles abstraites des patrons. En fait, l'approche ARM consiste à extraire des règles abstraites des patrons où l'analyste analyse les documents et les informations existants pour élaborer les composants et le rôle de chacun. Donc, dans cette activité, l'architecte est en train d'analyser les inputs afin de proposer une architecture en termes de composants ; ce qui valide l'existence d'un lien «équivaut à» entre les deux activités. De plus, ARM consiste à extraire un modèle source à partir du code source, ce qui est défini clairement par SArEM. Concernant les liens de composition reliés à la définition d'un plan de reconnaissance, 4 étapes sont réalisées dans ARM pour préparer le plan. Ainsi, des liens «composé de» sont établis entre l'activité «Définir un plan de reconnaissance» de SArEM et les 4 activités de l'approche ARM. En outre, dans ARM, une première architecture a été construite par l'activité «Détection des instances des patrons». Cette dernière activité détecte une architecture candidate en termes des blocs et des entités assignées à ces blocs. Cependant dans SArEM l'assignation est entre des groupes d'entités et la DAC. Ainsi, on peut dire que l'activité «Classification des groupes d'entités dans la DAC» est composée de l'activité «Détection des instances des patrons». Enfin, l'activité de comparaison, et celle de reconstruction et d'analyse sont des activités définies clairement par SArEM. Alors, il existe une forte conformité des activités d'extraction de SArEM aux activités de l'approche ARM. Cette conformité dans les activités a été suivie d'une conformité dans les artefacts et les rôles comme montre la figure 4.14.

Deuxièmement, concernant la conformité de la séquence d'activités définie par SArEM à la séquence d'activité de l'approche ARM, la conformité est élevée ; presque toute la séquence définie par SArEM est conforme à la séquence des activités de l'approche. En fait, comme montre les liens «précédent» et «suivant» établis entre les activités à la figure 4.13, les activités commencent par l'extraction des règles abstraites des patrons, qui sont comme une architecture conceptuelle ; après, un plan de reconnaissance est défini et puis un modèle source est extrait. Alors que la séquence d'activité définie par SArEM ne respecte pas cet ordre, la séquence des activités de SArEM consiste à extraire un modèle source puis à définir un plan de reconnaissance. La suite des activités de SArEM, qui sont des activités de synthèse, sont conformes au reste de la séquence des activités de l'approche.

Troisièmement, concernant la conformité du mécanisme opératoire de SArEM au mécanisme de l'approche ARM, nous présentons la classification des entités selon le mécanisme opératoire des différentes activités dans le paragraphe 3 de cette section.

2. La conformité de SArEM à l'approche proposée par Bowman et al.

Cette approche [Bowman et al., 1999] est proposée en prenant Linux comme un cas d'étude. Le processus d'extraction consiste à extraire une architecture logicielle en consultant les documents existants. Cette architecture contient les sous-systèmes et les relations entre eux. Dans ce cas d'étude, pour chaque sous-système l'architecte propose des sous-architectures conceptuelles en se basant aussi sur les documents. Ainsi, le résultat est une architecture conceptuelle détaillée de tout le système Linux. D'autre part, une architecture concrète est extraite : un extracteur de code source est utilisé pour extraire les fonctions et les relations entre eux et un outil de structuration pour extraire les relations entre les fichiers selon les dépendances entre les données et les fonctions extraites. De plus, manuellement, les fichiers du code source sont assignées aux sous-systèmes de l'architecture conceptuelle en se basant sur plusieurs critères comme la structure des répertoires, les noms des fichiers, les commentaires du code source, etc. Puis, l'outil de structuration (grok) identifie les relations entre les sous-systèmes selon les relations entre les fichiers sources. Enfin, un outil de visualisation est utilisé pour raffiner l'assignation

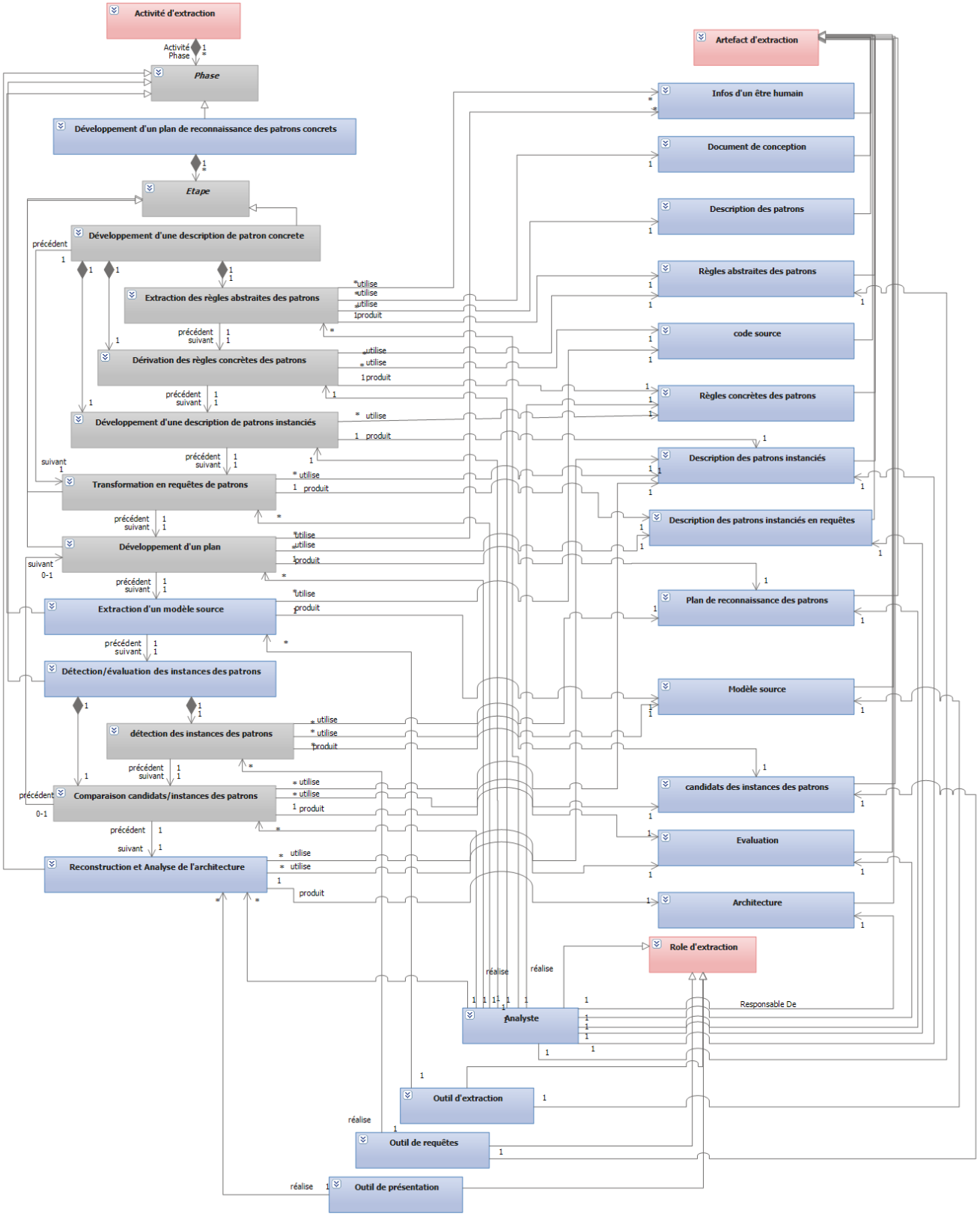

FIGURE 4.13 : Le méta-modèle élaboré de l'approche ARM.

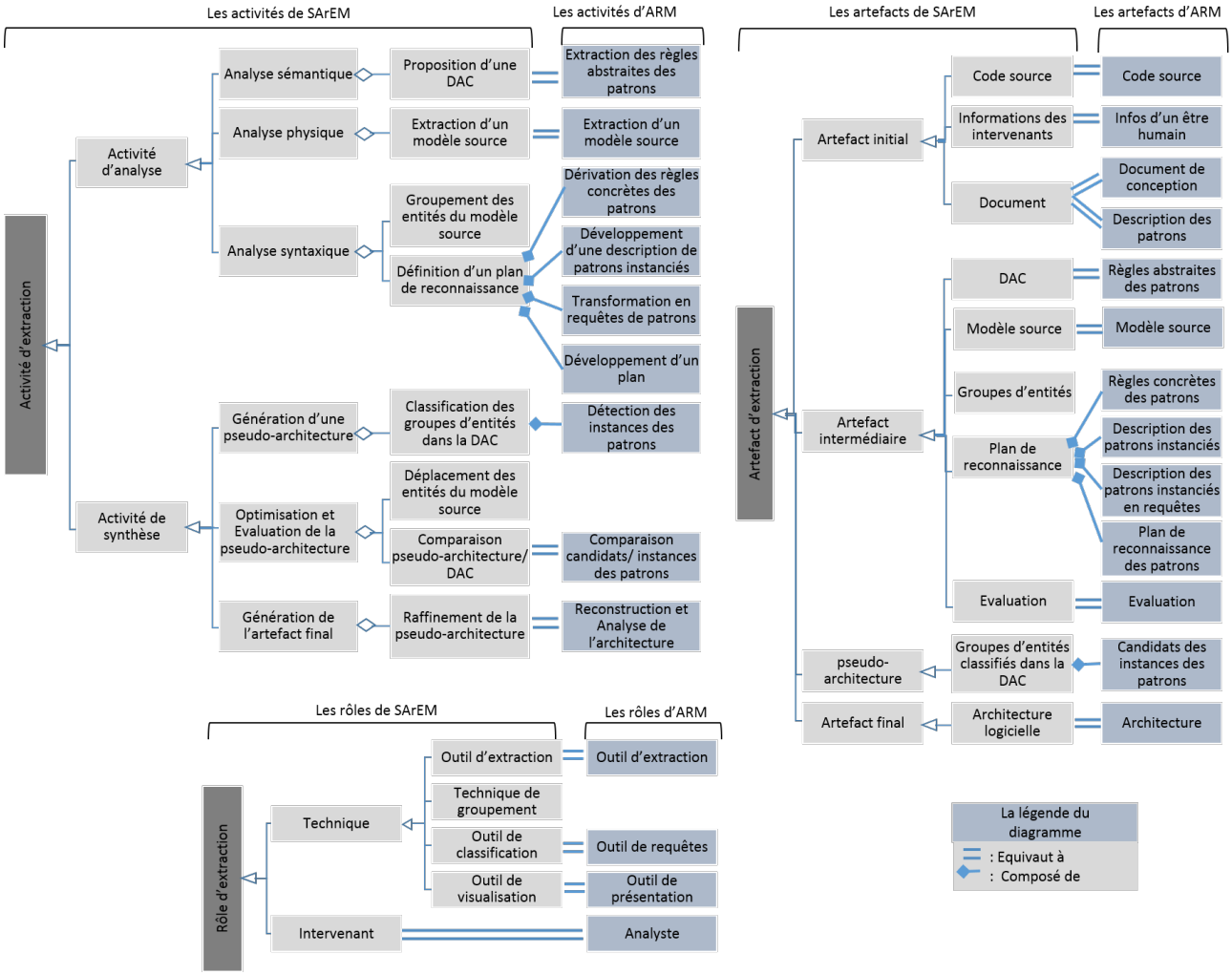

FIGURE 4.14 : La conformité de SArEM à l'approche ARM.

et produire la structure finale su système. La figure 4.15 illustre les éléments de l'approche proposée par Bowman et al. classifiés sous trois catégories.

Premièrement, concernant la conformité des éléments de SArEM aux éléments de l'approche de Bowman et al., le résultat de l'étude de conformité est présenté à la figure 4.16. La proposition d'une DAC définie par SArEM équivaut à la création d'une architecture logicielle. L'activité «Extraction d'un modèle source» est composée de deux activités définies par l'approche de Bowman et al. vu que l'approche consiste à extraire les relations entre les fonctions et celles des fichiers, ce qui est une partie du modèle source et pas un modèle source entier. Concernant les activités de synthèse, la conformité est élevée. Le seul défaut est que Bowman n'a pas évalué sa pseudo-architecture. Ainsi, comme montre la figure 4.16, SArEM a défini presque la majorité des éléments de l'approche de Bowman et al. en termes des activités, artefacts et rôles.

Deuxièmement, concernant la conformité de la séquence d'activité définie par SArEM à la séquence d'activité de l'approche proposée par Bowman et al., la conformité est élevée. La séquence définie par SArEM est entièrement conforme à la séquence des activités de l'approche. Comme montre les liens de «précédent» et «suivant» établis entre les activités de la figure 4.15, la séquence est entièrement comme définie par SArEM.

Troisièmement, concernant la conformité du mécanisme opératoire de SArEM au mécanisme de l'approche proposée par Bowman et al., nous présentons la classification des entités selon le mécanisme opératoire des différentes activités dans le paragraphe 3 de cette section.

3. La conformité de SArEM aux approches les plus connues

Dans ce qui suit, les études de conformité de SArEM aux approches d'extraction les plus connues sont présentées en bref. On présente si chaque mécanisme d'activité défini par SArEM est conforme aux mécanismes d'activités des approches les plus connues.

Le mécanisme de proposition d'une DAC

L'activité de proposition d'une DAC existe dans la plupart des approches (voir Table 4.1). L'approche de Bowman et al. [Bowman et al., 1999], tout comme l'approche de Riva [Riva, 2000], utilisent les documentations. Ainsi, l'utilisateur envisage l'architecture du système en utilisant les documents. L'approche Focus [Ding et Medvidovic, 2001], ReflexionModel [Murphy et Notkin, 1997], et celle de Kazman et al. [Kazman et Carrière, 1999] proposent l'utilisation de l'expertise humaine pour fournir un modèle architectural qui satisfait l'expert. L'approche de Medvidovic [Medvidovic et al., 2003] définit les composants principaux du système selon les besoins fonctionnels du système. Finalement, l'approche ARM [Guo et al., 1999] propose une méthode qui extrait une architecture logicielle en termes de patrons. Cette méthode prend en considération les documents de conception, les descriptions des patrons et les informations des expertises. D'autres approches, mentionnées dans le tableau, ne contiennent pas cette activité. Une forte existence de cette activité avec son mécanisme opératoire dans les approches existantes valide l'existence de l'activité 'Proposition d'une DAC' dans notre méta-modèle.

Le mécanisme d'extraction d'un modèle source

Comme il est mentionné dans le tableau 4.2, toutes les approches extraient à partir du code source des informations telles que les classes avec leurs relations, ou les noms des fichiers et les relations entre eux ... en utilisant des outils automatiques. Ce qui valide le mécanisme opératoire de l'activité d'extraction d'un modèle source défini par SArEM. Dans l'approche de Bowman et al. [Bowman et al., 1999], les outils, extracteur du code source et grok, réalisent

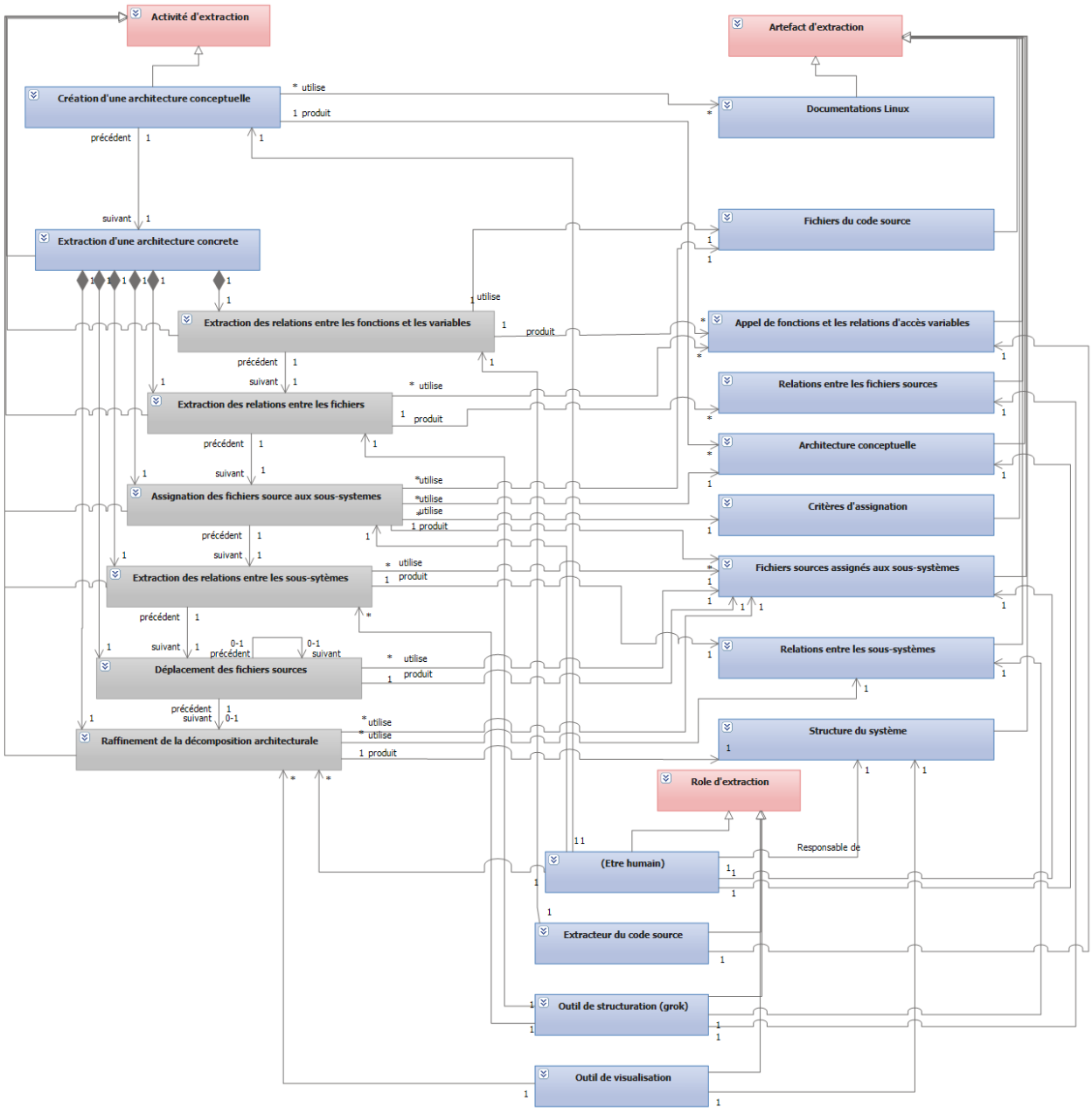

FIGURE 4.15 : Le méta-modèle élaboré de l'approche proposée par Bowman et al.

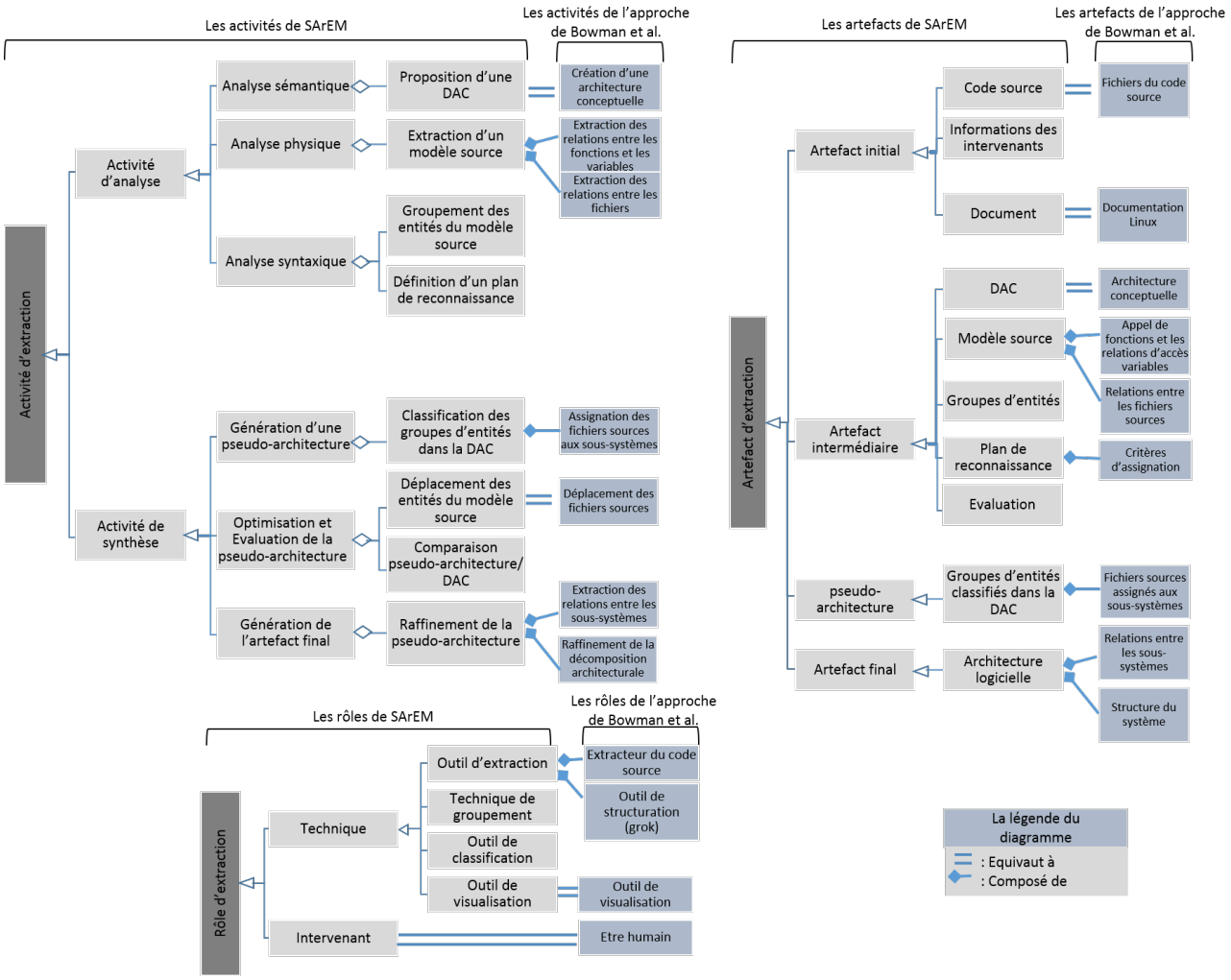

FIGURE 4.16 : La conformité de SArEM à l'approche proposée par Bowman et al.

successivement une extraction des relations entre les fonctions et les variables et une extraction des relations entre les fichiers. L'output, qui sont les relations entre les fonctions, variables et fichiers, est une partie du modèle source défini par SArEM. L'approche de Mahdavi [Mahdavi et al., 2003a], IGA [Rajalakshmi, M, 2014] et celle proposée par Mancoridis et al. [Mancoridis et al., 1998] génèrent un graphe de dépendances des modules formé des nœuds et des arcs. Les nœuds sont les modules et les arcs représentent les dépendances entre eux. L'outil utilisé est un outil d'analyse de code source. L'approche Focus [Ding et Medvidovic, 2001] et celle de Medvidovic [Medvidovic et al., 2003] génèrent les diagrammes de classes du système comme un modèle source en utilisant un outil d'extraction appelé Rational Rose. Kazman et al. [Kazman et Carrière, 1999] réalisent l'extraction de plusieurs modèles sources où chacun reflète un type d'information. De plus, les relations entre les modèles extraites sont dérivées.

Le mécanisme de groupement des entités du modèle source

Un nombre d'approches a proposé de grouper les entités du modèle source dont la plupart en utilisant des techniques de regroupement (voir Table 4.3). Ces techniques groupent les entités selon certains critères comme les relations entre les classes (composition, agrégation et d'association). Dans l'approche de Medvidovic [Medvidovic et al., 2003], même pour Focus [Ding et Medvidovic, 2001], l'architecte groupe manuellement les classes en des clusters. Puis, ces groupes sont paquetés en des composants. L'approche de Mahdavi [Mahdavi et al., 2003a], IGA [Rajalakshmi, M, 2014] et celle proposée par Mancoridis et al. [Mancoridis et al., 1998] utilisent l'outil Bunch pour regrouper les modules. La fonction appelée MQ 2.1 est utilisée comme fonction qui dirige les algorithmes. Riva [Riva, 2000] propose l'utilisation des outils de groupement afin de faire plusieurs groupements, nommés abstractions. Ainsi, une existence moyenne de cette activité avec son mécanisme a été observée dans les approches existantes.

Le mécanisme de définition d'un plan de reconnaissance

L'activité de définition d'un plan de reconnaissance indique comment la correspondance des groupes d'entités au modèle conceptuel doit être effectuée. Cependant, aucune des approches contient directement cette activité. Il existe certaines approches qui incluent la définition d'un plan entre les entités sources et l'architecture conceptuelle. Dans l'approche ARM [Guo et al., 1999], un utilisateur développe un plan concret de reconnaissance de patrons. Ce plan indique le composant «clé» qui doit d'abord être reconnu et l'ordre dans lequel le reste des composants doit être détecté. Le plan se compose de requêtes SQL où chaque requête vise à sélectionner pour un composant les entités du modèle source qui le composent. Même idée pour l'approche de Kazman et al. [Kazman et Carrière, 1999], pour chaque composant de l'abstraction conceptuelle, une requête SQL est définie ; l'ensemble des requêtes s'appelle Pattern. Dans le processus ReflexionModel [Murphy et al., 2001], le développeur définit un plan déclaratif qui indique quand une entité de modèle source correspond à un bloc du modèle de haut niveau. Par exemple, le plan déclaratif indique que tous les fichiers ayant le mot «pager» dans leur nom sont classés dans l'entité «Pager» du modèle de haut niveau. Pourtant, l'approche de Bowman et al. [Bowman et al., 1999] consiste à assigner manuellement les fichiers source à l'architecture conceptuelle ; un plan de reconnaissance a été implicitement défini par eux. Plus spécifiquement, Bowman et al. indiquent que la structure du répertoire, les conventions de dénomination des fichiers, les commentaires du code source, les documentations, le code source ... sont des critères pour réaliser la classification. Dans les approches restantes, cette activité n'est pas incluse. Ainsi, une existence moyenne de cette activité avec son mécanisme a été observée dans les approches existantes.

Le mécanisme de classification des groupes d'entités dans la DAC

Afin de créer une pseudo-architecture, Bowman et al. [Bowman et al., 1999] ont affecté manuellement chaque fichier source à un sous-système selon le plan de reconnaissance (les critères d'assignation). Dans le processus ReflexionModel [Murphy et al., 2001], un modèle de réflexion est calculé par l'outil ReflexionModel en utilisant le plan déclaratif, le modèle source et le modèle de haut niveau. Le modèle détermine où l'architecture de haut niveau convient et ne convient pas au modèle source. L'outil Dali [Kazman et Carrière, 1999] qui supporte l'approche de Kazman et al. exécute les patrons et donne les patrons reconnus comme output. Ce dernier dépend du modèle source extrait, dans les meilleurs cas, il montre les fichiers, les fonctions et les classes affectés aux composants définis dans les abstractions conceptuelles. Pareil pour ARM [Guo et al., 1999], qui effectue cette activité en utilisant Dali. Dans l'approche de Medvidovic et al. [Medvidovic et al., 2003], DAM et RAM sont également intégrés manuellement mais sans un plan de reconnaissance ; l'output est une architecture appelée 'comme architecture extraite'. Malheureusement, Medvidovic et al. n'ont pas défini un plan de reconnaissance ni spécifié comment les modèles sont intégrés. La même idée existe en Focus [Ding et Medvidovic, 2001] avec différentes terminologies. Une architecture intermédiaire est formée en cartographiant les composants identifiés sur l'architecture idéalisée. Selon le modèle source extrait, l'architecture intermédiaire est en termes de classes attribuées aux composants de l'architecture idéalisée. Une forte existence de cette activité avec son mécanisme opératoire dans les approches existantes valide l'existence de l'activité 'Classification des groupes d'entités dans la DAC' dans notre méta-modèle.

Le mécanisme de comparaison de la pseudo-architecture avec la DAC

Les approches qui accomplissent cette activité sont : ARM [Guo et al., 1999] et ReflexionModel [Murphy et al., 2001]. ARM propose la comparaison des candidats des instances des patrons avec les règles des patrons. La comparaison détermine quels candidats sont des faux positifs (un candidat non conçu comme une instance, mais il est détecté comme une instance) et de faux négatifs (un candidat conçu comme une instance, mais n'est pas détecté comme telle). Dans l'approche de Reflexion Model, le modèle généré donne directement la comparaison entre les deux modèles en mettant en évidence les convergences, les divergences et les absences. Les autres approches n'incluent pas cette activité. L'existence de l'activité de comparaison de la pseudo-architecture avec la DAC est faible dans les approches. Mais, la définition d'une telle activité dans un méta-modèle dédié à la spécification des processus d'extraction est nécessaire. Une telle activité exprime si le résultat satisfait les intervenants et nécessite par la suite des activités d'optimisation.

Le mécanisme du déplacement des entités du modèle source

Beaucoup des approches ont fourni l'architecture finale sans optimisation, comme l'approche de Mancoridis [Mancoridis et al., 1998], l'approche de Mahdavi [Mahdavi et al., 2003a], l'approche de Riva [Riva, 2000], l'approche de Medvidovic et al. [Medvidovic et al., 2003], l'approche de Kazman et al. [Kazman et Carrière, 1999] et l'approche ReflexionModel [Murphy et al., 2001]. En revanche, l'approche de Bowman et al. [Murphy et al., 2001] inclut cette activité dans le processus en transférant les fichiers du code source d'un sous-système vers un autre plus approprié. Dans ARM [Guo et al., 1999], il n'y a pas de déplacement des entités du modèle source, mais la modification du plan de reconnaissance ou du modèle source et la réitération de la méthode ARM sont proposées comme méthode d'optimisation. Cette méthode d'optimisation prend en compte le résultat de l'évaluation. Dans Focus [Ding et Medvidovic,

2001], l'optimisation est effectuée d'une autre manière pour améliorer les interactions entre les composants. Leur méthode consiste à identifier les cas clés d'utilisation puis à générer les diagrammes de séquences UML de ces cas clés d'utilisation. Les diagrammes de séquences contiennent les composants en tant qu'entités. L'approche IGA [Rajalakshmi, M, 2014] inclut le déplacement des entités selon l'évaluation tout en proposant deux stratégies aidant à la décision du déplacement. L'existence de l'activité du déplacement des entités du modèle source est faible dans les approches. Mais, à notre avis, les activités d'optimisation sont nécessaires dans les processus d'extraction pour améliorer la représentation de l'architecture.

Le mécanisme du raffinement de la pseudo-architecture

L'activité de raffinement n'a pas été exécutée par la plupart des approches. Il existe seulement deux approches qui ont raffiné l'architecture selon un style architectural. L'approche de Mevidovic [Medvidovic et al., 2003] qui inclut l'application d'un style au modèle intégré, et l'approche Focus [Ding et Medvidovic, 2001] qui comprend l'application d'un style mais pas à la fin du processus. Un style est choisi dans la proposition de l'architecture idéalisée.

4.9 La synthèse

Dans cette section, notre méta-modèle SArEM est évalué selon notre modèle de comparaison ; en particulier, selon le facteur méta-modèle. Tout d'abord, le méta-modèle a spécifié les concepts de base des processus d'extraction. Ces concepts sont en termes d'activités, d'artefacts et rôles et couvrent la plupart des principes des processus d'extraction. Cependant, notre méta-modèle n'a pas mentionné les niveaux sur lesquels opèrent les processus. Mais, en détaillant les artefacts, le méta-modèle a défini comment le système est représenté lors de l'extraction d'une architecture.

4.10 Conclusion

Dans ce chapitre, nous présentons un méta-modèle dénoté SArEM pour la spécification des processus d'extraction [Abboud et al., 2016a, Abboud et al., 2016b]. Ce méta-modèle a pour but d'offrir à l'architecte logiciel un outil conceptuel facilitant la tâche de selection, de comparaison et d'évaluation des différents processus d'extraction. En effet, cette approche répond aux limites des approches présentées dans le chapitre précédent. Principalement, elle se base autour de la spécification des activités, d'artefacts et des rôles d'extraction.

Notre méta-modèle est basé sur le méta-modèle SPEM [OMG et Notation, 2008] qui a été conçu au sein de la communauté OMG afin de standardiser les processus de développement des produits logiciels. Par conséquence, notre méta-modèle hérite les définitions des éléments principaux du SPEM et les étend par des concepts dédiés spécifiquement à l'extraction d'une architecture logicielle. Ces concepts ont été élaborés en étudiant les mécanismes d'extraction présents dans une multitude d'approches d'extraction. Nous avons présenté ici les approches les plus citées.

Dans le chapitre suivant, nous traitons la problématique au niveau des méthodes d'extraction. Cette deuxième contribution, basée sur SArEM, permet à un architecte de construire son propre processus d'extraction.

TABLE 4.1 : L'étude de conformité du mécanisme de proposition d'une DAC.

	<Activité d'extraction>	**<Artefact d'extraction>**			**<Rôle d'extraction>**
	Proposition d'une DAC	**Document**	**Informations des intervenants**	**DAC**	**Intervenants**
L'approche de Bowman et al.	= Création d'une architecture conceptuelle	= Documentions Linux	-	= Architecture conceptuelle	= Être humain
L'approche ARM	= Extraction des règles abstraites des patrons	= Document de conception = descriptions des patrons	= Infos d'un être humain	= Règles abstraites des patrons	= Analyste
L'approche de Mancoridis et al.	-	-	-	-	-
L'approche de Mahdavi	-	-	-	-	-
L'approche Focus	= Proposition d'un modèle architectural idéalisé	-	= Informations d'un ingénieur	= Modèle architectural idéalisé	= Ingénieur
L'approche de Riva	= Définition des concepts architecturaux	= Documentations	-	= Description des concepts architecturaux	= Experts du système
L'approche de Medvidovic	= Création du modèle DAM *(Discovered Architectural Model)*	-	-	= Modèle DAM	= Ingénieur
L'approche IGA	-	-	-	-	-
L'approche Reflexion model	= Définition d'un modèle conceptuel	-	= Expérience de l'ingénieur	= Modèle à haut-niveau	= Ingénieur
L'approche de Kazman et al.	= Identification des abstractions conceptuelles	-	= Connaissances d'un expert	Abstractions conceptuelles	= Expert

TABLE 4.2 : L'étude de conformité du mécanisme d'extraction d'un modèle source.

	<Activité d'extraction>	**<Artefact d'extraction>**		**<Rôle d'extraction>**
	Extraction d'un modèle source	**Code source**	**Modèle source**	**Outil d'extraction**
L'approche de Bowman et al.	◆— Extraction des relations entre les fonctions et les variables	= Fichiers du code source	◆— Relations entre les fonctions et les variables	◆— Extracteur du code source
	◆— Extraction des relations entre les fichiers	-	◆— Relations entre les fichiers sources	◆— Outil de structuration (grok)
L'approche ARM	= Extraction d'un modèle source	= Code source	= Modèle source	= Outil d'extraction
L'approche de Mancoridis et al.	= Représentation des modules du système et leurs relations	= Code source	= Graphe de dépendance des modules	= Outil d'analyse du code source
L'approche de Mahdavi	= Représentation des modules du système et leurs relations	= Code source	= Graphe de dépendance des modules	= Outil d'analyse du code source
L'approche Focus	= Génération des diagrammes de classes	= Code source	= Diagrammes de classes	= Rational Rose (outil)
L'approche de Riva	= Extraction d'un modèle source	= Code source	= Modèle du code source	= Outils automatiques
L'approche de Medvidovic	= Génération des diagrammes de classes	= Code source	= Diagrammes de classes	= Rational Rose (outil)
L'approche IGA	= Représentation du code	= Code source	= Graphe de dépendance des modules	= Outil d'analyse
L'approche Reflexion Model	= Extraction d'un modèle source	= Code source	= Modèle source	= Outil d'extraction
L'approche de Kazman et al.	= Extraction de plusieurs modèles concrets (vues)	= Code source	= Modèle concrets	= Outil d'extraction
	= Dérivation des relations entre les modèles	-	= Modèle concret	= Base de données SQL

TABLE 4.3 : L'étude de conformité du mécanisme de groupement des entités du modèle source.

	<Activité d'extraction>	**<Artefact d'extraction>**		**<Rôle d'extraction>**
	Groupement des entités sources	**Modèle source**	**Groupes des entités**	**Algorithme de recherche**
L'approche de Bowman et al.	-	-	-	-
L'approche ARM	-	-	-	-
L'approche de Mancoridis et al.	= Partitionnement	= Graphe de dépendances des modules	= Partition	= Outil Bunch
L'approche de Mahdavi	= Partitionnement	= Graphe de dépendances des modules	= Partition	= Outil Bunch
	= Partitionnement final (avec des building blocks)	= Building blocks	= Partition avec les building blocks	= Outil Bunch
L'approche Focus	= Groupement des classes reliées	= Diagrammes de classes	= Groupes des classes	Ingénieur
	= Paquetage des groupes des classes en composants	= Groupes des classes	= Composants	Ingénieur
L'approche de Riva	= Création des abstractions architecturales	= Modèle du code source	= Abstractions	= Outil de groupement
L'approche de Medvidovic	= Groupement des classes reliées	= Diagrammes de classes	= Groupes des classes	Ingénieur
	= Paquetage des groupes des classes en des éléments architecturaux	= Groupes des classes	= Éléments architecturaux	Ingénieur
L'approche IGA	= Partitionnement	= Graphe de dépendances des modules	= Partition	=Outil Bunch
L'approche Reflexion model	-	-	-	-
L'approche de Kazman et al.	-	-	-	-

TABLE 4.4 : L'étude de conformité du mécanisme de définition d'un plan de reconnaissance.

	<Activité>	**<Artefact>**			**<Role>**
	Définition d'un plan de reconnaissance	**Groupes d'entités**	**DAC**	**Plan de reconnaissance**	**Intervenant du système**
L'approche de Bowman et al.	-	◆— Fichiers du code source	= Architecture conceptuelle	◆— Critères d'assignation	-
L'approche ARM	◆— Développement d'un plan	◆— Modèle source	= Règles abstraites de patrons	◆— Plan de reconnaissance des patrons	= Analyste
L'approche de Mancoridis et al.	-	-	-	-	-
L'approche de Mahdavi	-	-	-	-	-
L'approche Focus	-	-	-	-	-
L'approche de Riva	-	-	-	-	-
L'approche de Medvidovic	-	-	-	-	-
L'approche IGA	-	-	-	-	-
L'approche de Reflexion model	◆— Définition d'un plan déclaratif entre les deux modèles	◆— Modèle source	= Modèle à haut-niveau	◆— Plan déclaratif	= Ingénieur
L'approche de Kazman et al.	◆— Définition des patrons	◆— Modèle concret	= Abstractions conceptuelles	◆— Patrons	= Expert

TABLE 4.5 : L'étude de conformité du mécanisme de classification des groupes d'entités dans la DAC.

	<Activité>	**<Artefact>**				**<Role>**
	classification des groupes d'entités dans la DAC	**Groupes d'entités**	**DAC**	**Plan de reconnaissance**	**Groupes d'entités classifiés dans la DAC**	**Outil de classification**
L'approche de Bowman et al.	◆— Assignation des fichiers sources aux sous-systèmes	◆— Fichiers du code source	= Architecture conceptuelle	◆— Critères d'assignation	◆— Fichiers sources assignés aux sous-systèmes	Être humain
L'approche ARM	◆— Détection des instances des patrons	◆— Modèle source	= Règles abstraites des patrons	◆— Plan de reconnaissance des patrons	◆— Candidats des instances des patrons	= Outil de requêtes
L'approche de Mancoridis et al.	-	-	-	-	-	-
L'approche de Mahdavi	-	-	-	-	-	-
L'approche Focus	= Correspondance des composants identifiés au modèle l'architectural idéalisé	= Composants	= Modèle architectural idéalisé	-	= Architecture intermédiaire	Ingénieur
L'approche de Riva	-	-	-	-	-	-
L'approche de Medvidovic	= Intégration du modèle DAM et RAM	◆— Modèle RAM	= Modèle DAM	-	= Architecture extraite	◆— Ingénieur
L'approche IGA	-	-	-	-	-	-
L'approche de Reflexion model	◆— Génération un modèle de réflexion	◆— Modèle source	= Modèle à haut-niveau	◆— Plan déclaratif	= Reflexion Model	= Outil Reflexion Model
L'approche de Kazman et al.	◆— Reconnaissance de patrons	◆— Modèle concret	= Abstractions conceptuelles	◆— Patrons	◆— Patrons identifiés	= Dali

5

SAD : Un modèle de processus d'extraction d'une architecture logicielle

5.1 Introduction

Dans le chapitre précédent, nous avons présenté un méta-modèle pour la spécification des processus d'extraction d'une architecture logicielle. Principalement, le méta-modèle comporte des entités définissant des activités, artefacts et rôles liés à la sémantique d'extraction d'une architecture logicielle et contient également des relations définissant les collaborations entre ces entités. Dans ce chapitre, nous présentons une nouvelle approche d'extraction d'une architecture logicielle, dénotée SAD *(Software Architecture Discovery)*, qui est basée sur le principe ECD (Extraction de Connaissances à partir de Données) et qui répond aux limitations des processus et outils d'extraction présentées au chapitre 3. Principalement, SAD aide un architecte à construire son propre processus d'extraction afin de découvrir une architecture du système conforme à ses souhaits. Notre approche d'extraction considère l'extraction d'une architecture logicielle comme un processus d'extraction de nouvelles connaissances à partir des données. Ainsi, notre approche SAD est basée sur deux contributions : La première contribution, présentée dans ce chapitre, est la proposition d'un modèle de processus d'extraction d'une architecture logicielle. Et la deuxième contribution, présentée dans le chapitre suivant, est l'élaboration d'une extension d'un outil ECD qui supporte l'exécution des processus d'extraction d'une architecture logicielle.

5.2 Positionnement et motivation

Notre étude des processus et outils d'extraction a mis en lumière certains manques dans les approches d'extraction existantes. Ces manques motivent nos travaux et permettent d'expliquer et de fixer le cadre et les hypothèses de notre approche. Nous avons ainsi développé une approche d'extraction [Abboud et al., 2017a, Abboud et al., 2017b], dénotée SAD, selon quatre particularités majeures. L'approche SAD permet à un architecte (1) d'intégrer ses connaissances durant le processus d'extraction, (2) d'interagir avec le processus d'extraction, (3) de construire le processus d'extraction d'une manière quasi-automatique selon la stratégie de conciliation,

(4) d'extraire une architecture logicielle en termes d'éléments, les éléments qui les composent et leurs interfaces.

Notre approche SAD est basée sur le principe d'ECD (Extraction des Connaissances à partir des données). ECD ou en anglais Knowledge Discovery in Databases (KDD) est apparue comme une solution pour découvrir l'information et les connaissances à partir des données. Selon Fayyad [Fayyad et al., 1996a, Fayyad et al., 1996b], KDD est un processus interactif et itératif impliquant de nombreuses étapes avec de nombreuses décisions prises par l'utilisateur. Il prend en compte toutes les données disponibles (base de données, images, vidéo ...) et fournit comme résultat une nouvelle connaissance utile (modèles, règles, clusters ...). En bref, KDD est un processus itératif et interactif qui (1) traite de grandes bases de données, (2) est supporté par les outils, et (3) permet l'intégration et l'interaction de l'utilisateur.

Ainsi, pour faire face aux limites présentes dans d'autres approches, notre travail est centré sur :

1. La proposition d'un modèle de processus d'extraction d'une architecture logicielle.
   En se basant sur le méta-modèle SArEM, qui est un méta-modèle contenant les concepts principaux d'extraction d'une architecture logicielle, nous proposons le modèle de processus d'extraction SAD *(Software Architecture Discovery)*. Le modèle d'extraction SAD peut être considéré comme un processus générique pour l'extraction d'une architecture logicielle. Ainsi, un architecte peut instancier ce modèle pour construire un processus particulier selon ses souhaits. Ce modèle est présenté dans ce chapitre.

2. L'élaboration d'une extension d'un outil ECD.
   Afin de permettre à l'architecte d'exécuter son processus d'extraction d'une manière automatique tout en permettant l'intégrité de ses connaissances et son interaction avec le processus, nous avons développé une extension d'un outil ECD. Ainsi, avec cette extension, un architecte peut profiter des propriétés d'ECD et exécuter d'une manière itérative et interactive son processus.

Par la suite, notre approche n'est pas limitée à la proposition d'un processus d'extraction. Elle permet à l'architecte d'instancier notre processus générique pour construire son processus particulier.

5.3 Les caractéristiques et propriétés principales du modèle SAD

Le modèle d'extraction d'une architecture logicielle SAD possède un ensemble de caractéristiques fondamentales qui lui permettent d'être suffisamment efficace :

- Le modèle SAD permet à l'architecte d'interagir avec les différents artefacts manipulés comme inputs et outputs durant son processus d'extraction. En addition, chaque activité peut être configurée par l'architecte selon ses souhaits.

- La plupart des activités définies dans le modèle prennent en compte des informations données par l'architecte et ont un bon niveau d'automatisation ; ceci aide à intégrer le point de vue de l'architecte d'une manière quasi-automatique.

- Le modèle suit la stratégie de conciliation entre une architecture prévue et des informations abstraites obtenues à partir du code source.

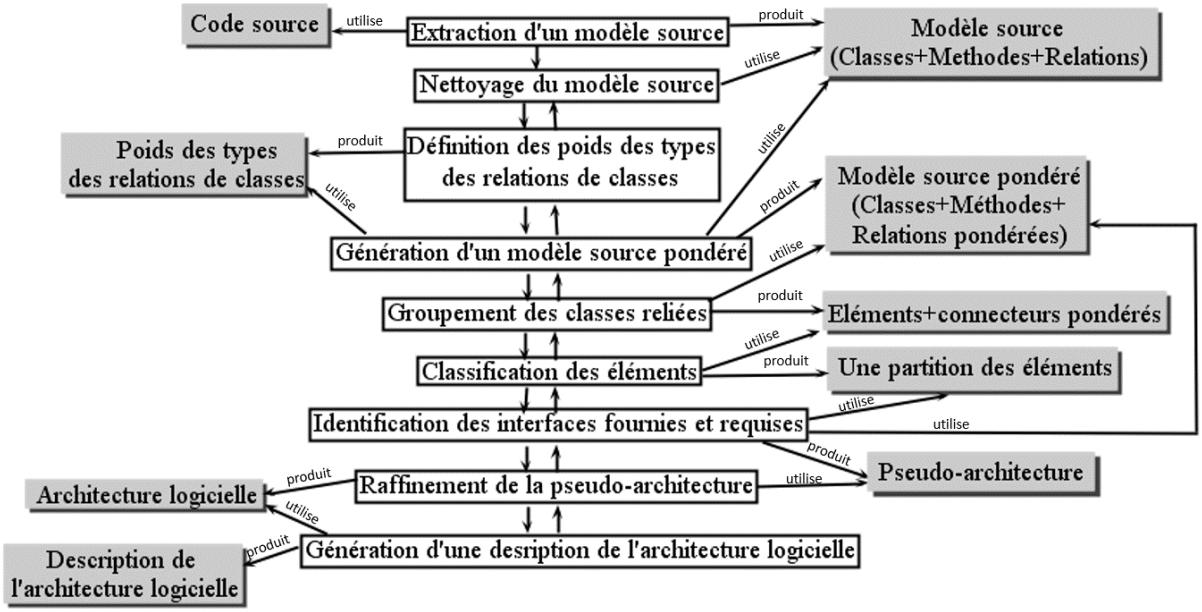

FIGURE 5.1 : Les activités et les artefacts du modèle SAD.

- Le modèle permet à l'architecte d'extraire une architecture complète à partir du code source. Cette architecture est en termes d'éléments architecturaux, de leurs interfaces fournies et requises, ainsi que de leur structure (les éléments qui les composent).

- Le modèle SAD, est supporté par un outil ECD que nous présentons dans le chapitre suivant. Cet outil permet à l'architecte de bénéficier des fonctionnalités classiques d'ECD comme les fonctionnalités de data mining et de visualisation ; ceci aide l'architecte à découvrir une architecture qui lui satisfait.

En général, le modèle SAD commence par l'analyse du code source et se termine par l'élaboration d'une architecture logicielle en termes d'éléments architecturaux du système, les éléments qui les composent et leurs interfaces. Précisément, des informations représentant le code source sont extraites à partir de l'implémentation. Ces informations sont les classes, leurs méthodes, fonctions... Puis, les classes sont regroupées en des éléments selon certains critères choisis par l'architecte. Le processus se poursuit avec l'élaboration d'une partition composée d'éléments aussi selon certains critères choisis par l'architecte. Enfin, les interfaces des parties de la partition sont identifiées. Le diagramme d'activités de la Figure 5.1 montre les activités du modèle du processus en mettant en évidence les entrées et sorties de chacune. Le modèle sera présenté dans les sections suivantes tout en détaillant ses artefacts, activités et rôles.

5.4 Les artefacts manipulés du modèle SAD

Un artefact est un objet manipulé durant le processus d'extraction ; il est utilisé et/ou produit par une activité. Dans ce qui suit, nous citons les artefacts manipulés durant le processus d'extraction SAD et nous les détaillons selon une représentation graphique et un formalisme mathématique. La figure 5.1 illustre les activités et les artefacts du modèle SAD en soulignant les inputs et outputs de chaque activité.

5.4.1 Le code source

Le code source est un texte qui représente les instructions d'un système telles qu'elles ont été écrites par un programmeur. Le code source se matérialise souvent sous la forme d'un ensemble de fichiers textes.

5.4.2 Le modèle source, $\mathbb{MS}$

Un modèle source est une représentation du code source du système. Cette représentation est indépendante du langage de programmation et décrit les entités logicielles du code source et leurs relations. Plus spécifiquement, un modèle source est composé des informations suivantes : des classes, leurs attributs et méthodes, des relations entre les classes et des appels entre les méthodes. Dans ce qui suit, nous présentons le modèle source selon une représentation graphique et un formalisme mathématique, puis, nous détaillons les constituants du modèle source.

Représentation graphique d'un modèle source :
Un modèle source est représenté graphiquement par :

- Un diagramme de classes selon la notation UML
  Ce diagramme symbolise les classes, leurs attributs et méthodes, et les relations entre les classes. La figure 5.2 présente un exemple d'un diagramme de classes selon la notation UML.

- Un graphe d'appels de méthodes
  Ce graphe est composé des nœuds et des arcs où les nœuds symbolisent les méthodes et les arcs symbolisent les appels entre les méthodes. La figure 5.3 présente un exemple d'un graphe d'appels de méthodes.

Formalisme mathématique d'un modèle source :
Un modèle source, $\mathbb{MS}$, est défini par le triplet :

$$\mathbb{MS} = (\mathbb{CL}, \mathbb{REL\_CL}, \mathbb{REL\_M})$$

avec :
$\mathbb{CL}$ l'ensemble des classes du modèle source.
$\mathbb{REL\_CL}$ l'ensemble des relations de classes du modèle source.
$\mathbb{REL\_M}$ l'ensemble des relations de méthodes du modèle source.

Les classes d'un modèle source, $\mathbb{CL}$

Une classe est un modèle de définition pour des objets ayant le même ensemble d'attributs (les caractères propres à un objet), et le même ensemble d'opérations ou méthodes (les actions applicables à un objet). Une classe peut être des types suivants :

- Une classe concrète
  C'est la classe par défaut, elle peut avoir directement des instances.

- Une classe abstraite
  C'est une classe dont l'implémentation n'est pas complète et qui n'est pas instanciable. Elle sert de base à d'autres classes dérivées (héritées).

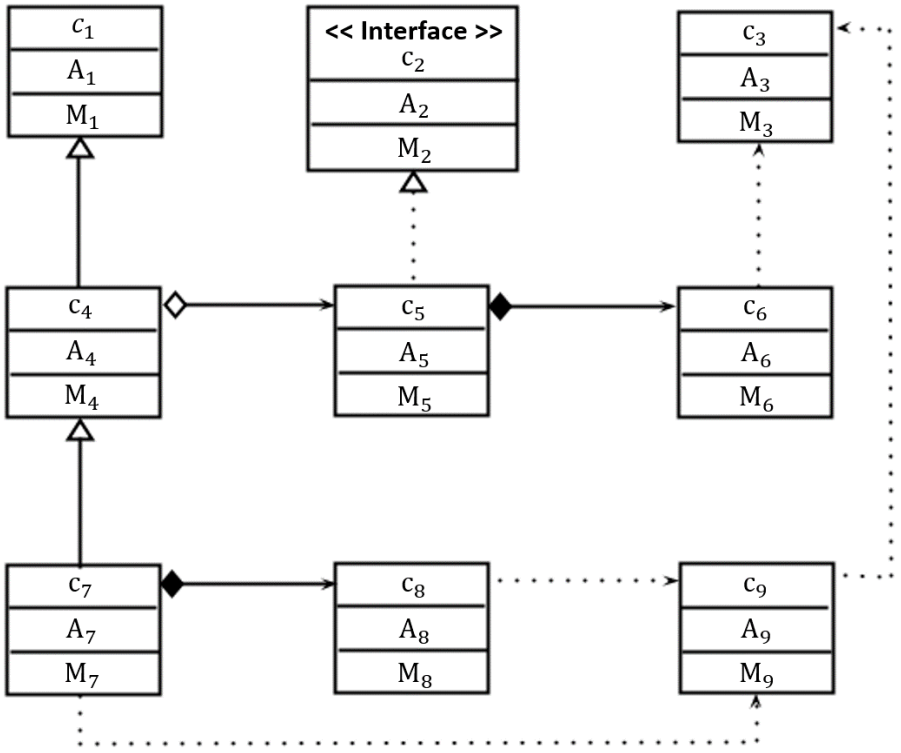

FIGURE 5.2 : Un diagramme de classes selon la notation UML.

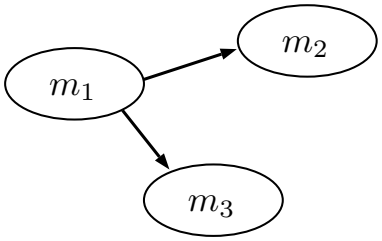

FIGURE 5.3 : Un graphe d'appels de méthodes.

- Une interface
  C'est une classe qui fournit un ensemble d'opérations sans implémentation.

Représentation graphique d'une classe :
Selon la notation UML, la classe est représentée par un rectangle divisé en trois compartiments contenant le nom de la classe, les attributs de la classe et ses méthodes. En addition, la classe est représentée graphiquement selon son type :

- La classe concrète
  Une classe de ce type n'a aucune spécificité dans sa représentation graphique. À la figure 5.2, les classes ayant respectivement les noms $c_3, c_4, c_5, c_6, c_7, c_8$ et c_9 sont des classes concrètes.

- La classe abstraite
  Une classe de ce type a son nom marqué en italique. À la figure 5.2, la classe ayant le nom c_1 est une classe abstraite.

- L'interface
  Une interface a son nom procédé par le stéréotype «Interface». À la figure 5.2, la classe ayant le nom c_2 est une interface.

Formalisme mathématique des classes :
Afin de définir les classes selon un formalisme mathématique, nous définissons les préambules suivants :
Soient :
$\mathbb{C}$ l'ensemble des noms des classes.
$\mathbb{A}$ l'ensemble des attributs des classes.
$\mathbb{M}$ l'ensemble des méthodes des classes.
$\mathbb{T}\_\mathbb{C} = \{c\_concrete, c\_abstraite, c\_interface\}$ l'ensemble des types des classes.

Une classe $CL_i \in \mathbb{CL}$ est définie par le quadruple $CL_i = (c_i, A_i, M_i, t_c)$ tel que :
$c_i \in \mathbb{C}$ est le nom de la classe CL_i.
$A_i \in \mathbb{A}$ est l'ensemble des attributs de la classe CL_i.
$M_i \in \mathbb{M}$ est l'ensemble des méthodes de la classe CL_i.
$t_c \in \mathbb{T}\_\mathbb{C}$ est le type de la classe CL_i.

Par la suite, les classes, $\mathbb{CL}$, qui constituent un modèle source,$\mathbb{MS}$, sont définies par l'ensemble suivant :

$$\mathbb{CL} = \{(c_i, A_i, M_i, t\_c) / i \in \mathbb{N} \wedge c_i \in \mathbb{C} \wedge A_i \in \mathbb{A} \wedge M_i \in \mathbb{M} \wedge t\_c \in \mathbb{T}\_\mathbb{C}\}$$

Ainsi, les classes de la figure 5.2 sont définies par l'ensemble suivant :
$\mathbb{CL} = \{CL_1, CL_2, CL_3, CL_4, CL_5, CL_6, CL_7, CL_8, CL_9\}$
tel que :
$C_1 = (c_1, A_1, M_1, c\_abstraite)$, $C_2 = (c_2, A_2, M_2, c\_interface)$, $C_3 = (c_3, A_3, M_3, c\_concrete)$,
$C_4 = (c_4, A_4, M_4, c\_concrete)$, $C_5 = (c_5, A_5, M_5, c\_concrete)$, $C_6 = (c_6, A_6, M_6, c\_concrete)$,
$C_7 = (c_7, A_7, M_7, c\_concrete)$, $C_8 = (c_8, A_8, M_8, c\_concrete)$, $C_9 = (c_9, A_9, M_9, c\_concrete)$

Les relations de classes d'un modèle source, $\mathbb{REL}\_\mathbb{CL}$

Une relation de classes est une relation entre deux classes, elle exprime une certaine dépendance entre elles. Une relation est des types suivants :

- Une relation de généralisation
  Cette relation signifie qu'une classe (classe enfant) hérite tous les membres d'une autre classe (qu'on nomme super classe).

- Une relation de réalisation
  Cette relation signifie qu'une classe (classe enfant) implémente les fonctions d'une autre classe de type interface (qu'on nomme super classe).

- Une relation de dépendance
  Cette relation signifie qu'une classe dépend d'une autre classe. Par conséquent, toute modification dans une des classes peut affecter la fonctionnalité de l'autre classe, qui dépend de la première.

- Une relation d'agrégation
  Cette relation signifie qu'une classe (classe agrégée) fait partie d'une autre classe ("classe agrégat"). Dans une relation d'agrégation, l'élément agrégé peut être partagé avec d'autre classes (agrégats) et peut exister sans agrégat : les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.

- Une relation de composition
  La composition est une agrégation forte ; à un même moment, une instance de composant ne peut être liée qu'à un seul agrégat. De plus, les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit, ses composants le sont aussi.

Représentation graphique des relations de classes :
Selon la notation UML, une relation de classes est représentée graphiquement selon son type :

- La relation de généralisation, symbolisée par $\leftarrow$
  Cette relation est représentée graphiquement par une ligne continue avec une flèche fermée et non remplie de la part de la super classe. À la figure 5.2, il existe deux relations de généralisation, qui sont les suivants : $CL_1 \leftarrow CL_4$, $CL_4 \leftarrow CL_7$.

- La relation de réalisation, symbolisée par $\dashleftarrow$
  Cette relation est représentée graphiquement par une ligne en pointillée avec une flèche fermée et non remplie de la part de la super classe. À la figure 5.2, il existe une relation de réalisation, qui est la suivante : $CL_2 \dashleftarrow CL_5$.

- La relation de dépendance, symbolisée par $\dashleftarrow$
  Cette relation est représentée graphiquement par une ligne en pointillée avec une flèche partant de la classe ayant une sorte de dépendance vers la deuxième classe. À la figure 5.2, il existe quatre relations de dépendance, qui sont les suivants : $CL_3 \dashleftarrow CL_6$, $CL_3 \dashleftarrow CL_9$, $CL_9 \dashleftarrow CL_7$, $CL_9 \dashleftarrow CL_8$.

- La relation d'agrégation, symbolisée par $\multimap$
  Cette relation est représentée graphiquement par une ligne continue de la classe agrégat à la classe de partie (classe agrégée) avec une forme de diamant non remplie de la part de la classe agrégat. À la figure 5.2, il existe une relation d'agrégation, qui est la suivante : $(c_4, A_4, M_4) \multimap (c_5, A_5, M_5)$.

- La relation de composition, symbolisée par $\blacklozenge\!\!\leftarrow$
  Cette relation est représentée graphiquement comme la relation d'agrégation, mais cette fois, la forme du diamant de la part de la classe agrégat est remplie. À la figure 5.2, il existe deux relations de composition, qui sont les suivantes : $CL_5 \blacklozenge\!\!\leftarrow CL_6$, $CL_7 \blacklozenge\!\!\leftarrow CL_8$

Formalisme mathématique des relations de classes :
Afin de définir les relations de classes selon un formalisme mathématique, nous définissons le préambule suivant :
Soit :
$\mathbb{T}\_\mathbb{R} = \{r\_generalisation, r\_realisation, r\_dependance, r\_agregation, r\_composition\}$ l'ensemble des types des relations de classes.

Une relation $rel\_cl \in \mathbb{REL}\_\mathbb{CL}$ entre deux classes $CL_i \in \mathbb{CL}$ et $CL_j \in \mathbb{CL}$, avec $i \neq j$, est défini selon son type :

- La relation de généralisation
  Une relation de ce type est définie par le triplet $rel\_cl = (CL_i, CL_j, r\_generalisation)$ tel que :
  $CL_i \in \mathbb{CL}$ la super classe et $CL_j \in \mathbb{CL}$ la classe enfant.
- La relation de réalisation
  Une relation de ce type est définie par le triplet $rel\_cl = (CL_i, CL_j, r\_realisation)$ tel que :
  $CL_i \in \mathbb{CL}$ la super classe et $CL_j \in \mathbb{CL}$ la classe enfant.
- La relation de dépendance
  Une relation de ce type est définie par le triplet $rel\_cl = (CL_i, CL_j, r\_dependance)$ tel que :
  $CL_i \in \mathbb{CL}$ la classe dont l'autre classe y dépend et $CL_j \in \mathbb{CL}$ la classe ayant une sorte de dépendance.
- La relation d'agrégation
  Une relation de ce type est définie par le triplet $rel\_cl = (CL_i, CL_j, r\_agregation)$ tel que :
  $CL_i \in \mathbb{CL}$ la classe agrégat et $CL_j \in \mathbb{CL}$ la classe agrégée.
- La relation de composition
  Une relation de ce type est définie par le triplet $rel\_cl = (CL_i, CL_j, r\_composition)$ tel que :
  $CL_i \in \mathbb{CL}$ la classe agrégat et $CL_j \in \mathbb{CL}$ l'élément composant.

Par la suite, les relations de classes, $\mathbb{REL}\_\mathbb{CL}$, qui constituent un modèle source, $\mathbb{MS}$, sont définies par l'ensemble suivant :

$$\mathbb{REL}\_\mathbb{CL} = \{(CL_i, CL_j, t\_r)/i, j \in \mathbb{N} \wedge CL_i, CL_j \in \mathbb{CL} \wedge CL_i \neq CL_j \wedge t\_r \in \mathbb{T}\_\mathbb{R}\}$$

Ainsi, les relations de classes de la figure 5.2 sont définies par l'ensemble suivant :

$$\begin{aligned}\mathbb{REL}\_\mathbb{CL} = \{&(CL_1, CL_4, r\_generalisation), (CL_4, CL_7, r\_generalisation), (CL_2, CL_5, r\_realisation),\\ &(CL_3, CL_6, r\_dependance), (CL_3, CL_9, r\_dependance), (CL_9, CL_7, r\_dependance),\\ &(CL_9, CL_8, r\_dependance), (CL_4, CL_5, r\_agregation), (CL_5, CL_6, r\_composition),\\ &(CL_7, CL_8, r\_composition)\}\end{aligned}$$

Les relations de méthodes d'un modèle source, $\mathbb{REL}\_\mathbb{M}$

Une relation de méthode est une relation entre deux méthodes, elle exprime un appel entre elles.

Représentation graphique d'une relation de méthodes :

TABLE 5.1 : Le poids des types des relations de classes.

Type de relations de classes	Poids
Généralisation	p_1
Réalisation	p_2
Dépendance	p_3
Agrégation	p_4
Composition	p_5

Une relation de méthodes est symbolisée par $\leftarrow$
Cette relation est représentée graphiquement par une ligne continue avec une flèche partant de la méthode qui appelle vers la méthode appelée. À la figure 5.3, il existe deux relations de méthodes, qui sont les suivants : $m_2 \leftarrow m_1$, $m_3 \leftarrow m_1$.

Formalisme mathématique des relations de méthodes :
Une relation de méthode $rel\_m \in REL\_M$ entre deux méthodes $m_i \in M_m$ et $m_j \in M_n$, avec $M_m, M_n \subset \mathbb{M}$ et $m \neq n$, est définie par le doublet $rel\_m = (m_i, m_j)$ tel que :
$m_i \in \mathbb{M}$ la méthode appelée et $m_j \in \mathbb{M}$ la méthode qui appelle.

Par la suite, les relations de méthodes REL_M sont définies par l'ensemble :

$$\mathbb{REL}\_\mathbb{M} = \{(m_i, m_j) / m_i \in M_m \ \wedge \ m_j \in M_n \ \wedge \ M_m, M_n \subset \mathbb{M} \ \wedge \ m \neq n\}$$

5.4.3 Les poids des types des relations de classes, $\mathbb{P}$

Cet artefact exprime les poids donnés aux différents types de relations de classes.

Représentation graphique des poids des types de relations de classes :
Cet artefact est représenté par un tableau montrant le poids de chaque type de relations de classes. La table 5.1 donne une représentation de cet artefact. Chaque type de relations de classes a un poids.
Formalisme mathématique des poids des types de relations de classes :
L'ensemble des poids donnés à chaque type de relation est défini par :

$$\mathbb{P} = \{(t\_r, p\_r) / t\_r \in \mathbb{T}\_\mathbb{R} \ \wedge \ p\_r \in \mathbb{N}\}$$

5.4.4 Le modèle source pondéré, $\mathbb{MS}\_\mathbb{P}$

Un modèle source pondéré est un modèle source avec les relations de classes pondérées ; autrement dit, les relations entre les classes ont un poids.

Représentation graphique d'un modèle source pondéré :
Un modèle source pondéré est représenté graphiquement par :

- Un diagramme de classes pondéré
  Ce diagramme est similaire au digramme de classes selon la notation UML, mais avec les relations de classes pondérées. Une relation de classes pondérée est représentée comme une relation de classes, selon son type, avec un poids sur le lien. La figure 5.4 présente un exemple d'un diagramme de classes pondéré.

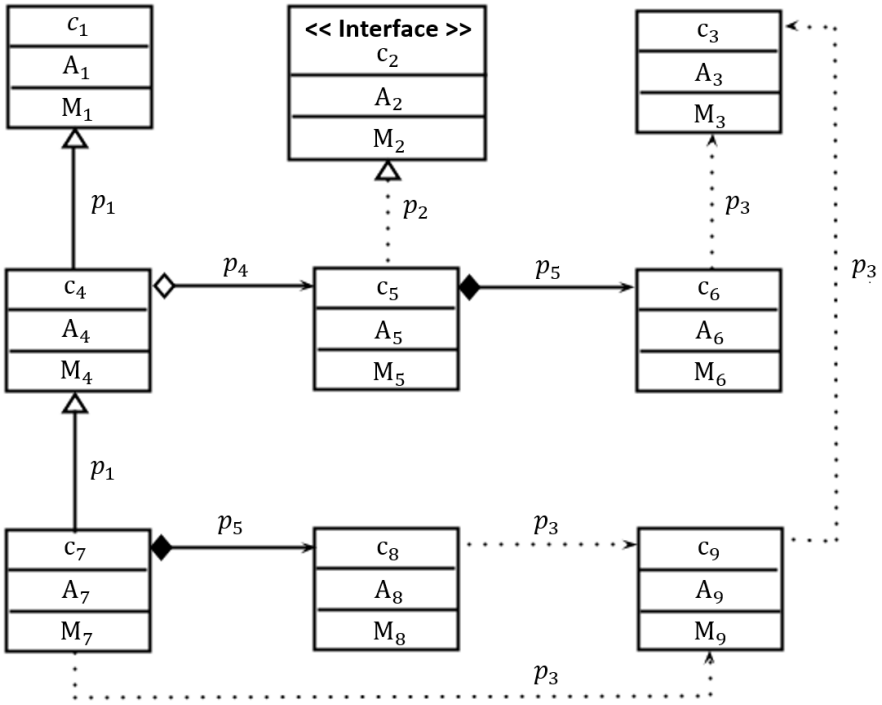

FIGURE 5.4 : Un diagramme de classes avec les relations de classes pondérées.

- Un graphe d'appels de méthodes
  Ce graphe est identique au graphe d'appels de méthodes d'un modèle source (voir figure 5.3).

Formalisme mathématique d'un modèle source pondéré :
Afin de définir un modèle source pondéré, nous définissons tout d'abord l'ensemble des relations de classes pondérées, $\mathbb{REL\_P\_CL}$.
Une relation pondérée $rel\_p\_cl \in \mathbb{REL\_P\_CL}$ entre deux classes $CL_i \in \mathbb{CL}$ et $CL_j \in \mathbb{CL}$, avec $i \neq j$, est définie par le quadruple $rel\_p\_cl = (CL_i, CL_j, r\_t, p_r)$ tel que :
$(CL_i, CL_j, r\_t) \in \mathbb{REL\_CL}$ et $p_r \in \mathbb{N}$.

Par la suite, les relations de classes pondérées, $\mathbb{REL\_P\_CL}$, qui constituent un modèle source pondéré, $\mathbb{MS\_P}$, sont définies par l'ensemble suivant :

$$\mathbb{REL\_P\_CL} = \{(CL_i, CL_j, t\_r, p_r) / (CL_i, CL_j, r\_t) \in \mathbb{REL\_CL} \wedge \ p\_r \in \mathbb{N}\}$$

Ainsi, un modèle source pondéré, $\mathbb{MS\_P}$, est défini par le triplet :

$$\mathbb{MS} = (\mathbb{CL}, \mathbb{REL\_P\_CL}, \mathbb{REL\_M})$$

avec :
$\mathbb{CL}$ l'ensemble des classes du modèle source pondéré.
$\mathbb{REL\_P\_CL}$ l'ensemble des relations de classes pondérées du modèle source pondéré.
$\mathbb{REL\_M}$ l'ensemble des relations de méthodes du modèle source pondéré.

5.4.5 Les éléments et leurs connecteurs pondérés, $\mathbb{ELT\_CON}$

Les éléments et leurs connecteurs pondérés sont des artefacts qui représentent le modèle source pondéré avec les classes groupées dans des éléments et des connecteurs pondérés entre les éléments. Dans ce qui suit, nous présentons les éléments et leurs connecteurs pondérés selon une

représentation graphique et un formalisme mathématique, puis, nous détaillons les constituants des éléments et leurs connecteurs pondérés.

Représentation graphique des éléments et leurs connecteurs pondérés :
L'artefact des éléments et leurs connecteurs pondérés est représenté graphiquement comme un diagramme de classes pondéré (voir figure 5.4) avec les classes groupées en des éléments et des liens entre les éléments. La figure 5.5 présente un exemple des éléments et leurs connecteurs pondérés.

Formalisme mathématique des éléments et leurs connecteurs pondérés :
Les éléments et leurs connecteurs pondérés sont définis par le doublet suivant :

$$\mathbb{ELT\_CON} = (\mathbb{ELT}, \mathbb{CON})$$

avec :
$\mathbb{ELT}$ l'ensemble des éléments.
$\mathbb{CON}$ l'ensemble des connecteurs pondérés.

Les éléments, $\mathbb{ELT}$

Un élément est un groupe de classes. Les éléments sont une partition de classes. Ainsi les éléments sont des groupes de classes ayant les propriétés suivantes :
- Les groupes de classes sont non vides.
- L'intersection des groupes de classes est vide.
- L'union des groupes de classes couvre toutes les classes.

Représentation graphique d'un élément :
Un élément est représenté graphiquement par un rectangle gris qui englobe les classes qui le composent. Par exemple, à la figure 5.5, il existe six éléments (rectangles gris) qui groupent des classes.

Formalisme mathématique des éléments :
Par la suite, les éléments, qui sont une partition des classes, sont définis par l'ensemble suivant :

$$\mathbb{ELT} = \{E_i / i \in \mathbb{N} \wedge \cap E_i = \emptyset \wedge \cup E_i = \mathbb{CL}\}$$

Ainsi, les éléments de la figure 5.5 sont définis par l'ensemble suivant :
$\mathbb{ELT} = \{E_1, E_2, E_3, E_4, E_5, E_6\} = \{CL_1, CL_4\} \cup \{CL_2\} \cup \{CL_3\} \cup \{CL_5, CL_6\} \cup \{CL_7, CL_8\} \cup \{CL_9\}$

Les connecteurs pondérés, $\mathbb{CON}$

Un connecteur pondéré est un lien, ayant un poids, établi entre deux éléments différents. Ainsi, les connecteurs pondérés sont des liens pondérés entre des éléments.

Représentation graphique d'un connecteur pondéré :
Un connecteur pondéré est représenté par un lien gris entre les éléments. Par exemple, à la figure 5.5, il existe six connecteurs pondérés (traits noirs épais) entre les éléments.

Formalisme mathématique des connecteurs pondérés :
Un connecteur pondéré $Con \in \mathbb{CON}$ est définit par le triplet $Con = (E_i, E_j, P_{i,j})$ tel que :

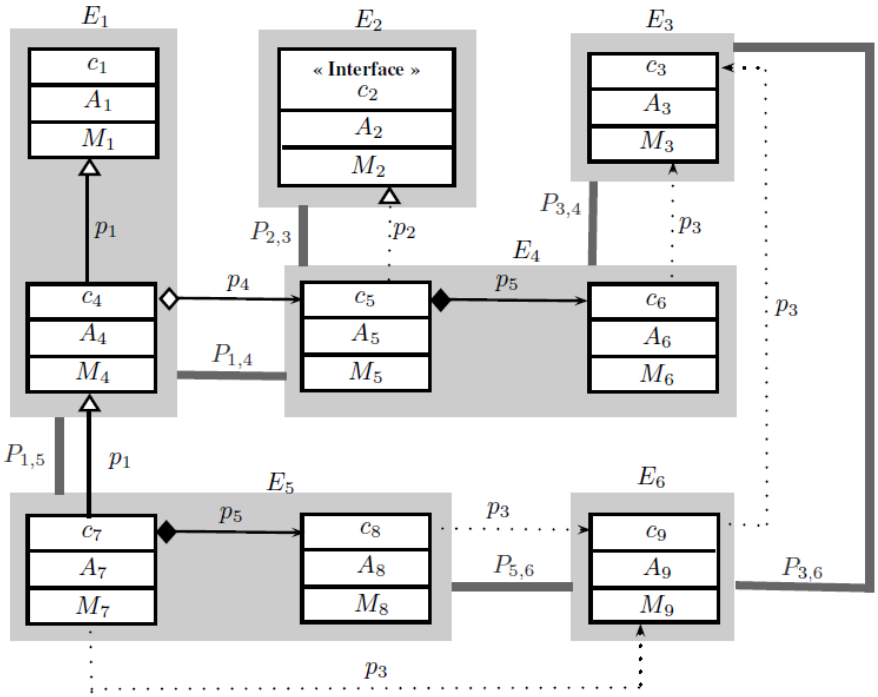

FIGURE 5.5 : Les éléments et leurs connecteurs.

$E_i, E_j \in \mathbb{ELT}$ les éléments extrémités.
$P_{i,j} \in \mathbb{N}$ le poids du connecteur pondéré.
Par la suite, les connecteurs pondérés, $\mathbb{CON}$ sont définis par l'ensemble suivant :

$$\mathbb{CON} = \{(E_i, E_j, P_{i,j})/i, j \in \mathbb{N} \wedge E_i,\ E_j \in \mathbb{ELT} \wedge P_{i,j} \in \mathbb{N}\}$$

Ainsi, les connecteurs de la figure 5.5 sont définis par l'ensemble suivant :
$\mathbb{CON} = \{(E_1, E_4, P_{1,4}), (E_1, E_5, P_{1,5}), (E_2, E_4, P_{2,4}), (E_3, E_4, P_{3,4}), (E_3, E_6, P_{3,6}), (E_5, E_6, P_{5,6})\}$

5.4.6 La partition des éléments, $\mathbb{PART}$

Une partition des éléments est un ensemble de parties non vides d'éléments tels que les parties sont deux à deux disjointes et recouvrent tous les éléments.

Représentation graphique d'une partie d'une partition des éléments :
Une partie d'une partition des éléments est représentée graphiquement par un cadre noir qui englobe les éléments qui le composent. La figure 5.6 présente un exemple d'une partition des éléments. Comme montre la figure, il existe deux parties qui groupent des éléments.

Formalisme mathématique de la partition des éléments :
Une partition des éléments est définie par l'ensemble suivant :

$$\mathbb{PART} = \{Part_i/i \in \mathbb{N} \wedge \cap Part_i = \emptyset \wedge \cup Part_i = \mathbb{ELT}\}$$

Ainsi, les parties de la figure 5.6 sont définies par l'ensemble suivant :
$\mathbb{PART} = \{Part_1, Part_2\} = \{E_1, E_2, E_3, E_4\} \cup \{E_5, E_6\}$

5.4.7 La pseudo-architecture, $\mathbb{PS\_A}$

La pseudo-architecture est une architecture du système. Elle est constituée d'une partition des éléments(voir le paragraphe 5.4.6), des interfaces fournies des parties et des interfaces requises

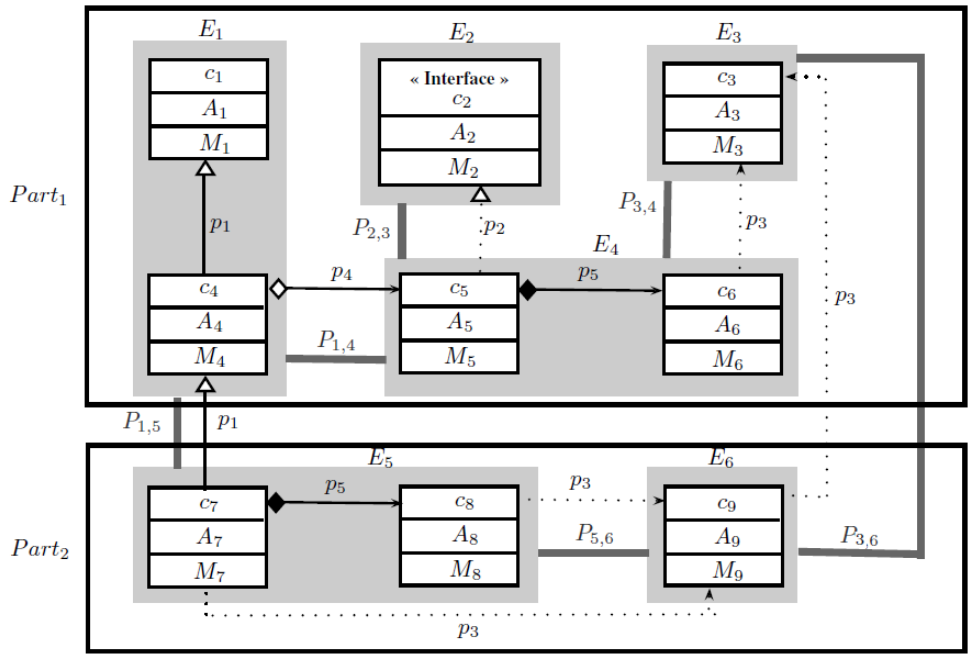

FIGURE 5.6 : La partition des éléments.

des parties.

Représentation graphique d'une pseudo-architecture :
Une pseudo-architecture est représentée graphiquement comme une partition avec des interfaces fournies et requises pour les parties. La figure 5.7 présente un exemple d'une pseudo-architecture.

Formalisme mathématique d'une pseudo-architecture :
Une pseudo-architecture est définie par l'ensemble suivant :

$$\mathbb{PS\_A} = (\mathbb{PART}, \mathbb{IF}, \mathbb{IR})$$

avec :
$\mathbb{PART}$ la partition de la pseudo-architecture.
$\mathbb{IF}$ l'ensemble des interfaces fournies des parties de la pseudo-architecture.
$\mathbb{IR}$ l'ensemble des interfaces requises des parties de la pseudo-architecture.

Les interfaces fournies, $\mathbb{IF}$

Une interface fournie d'une partie décrit un service fourni par la partie ; les autres parties interagissent avec la partie à travers son interface fournie.

Représentation graphique d'une interface fournie :
L'interface fournie d'un partie est symbolisée par ▫—●. Par exemple, à la figure 5.7, il existe deux interfaces fournies pour la partie $Part_1$ et une pour la partie $Part_2$.

Formalisme mathématique des interfaces fournies :
Les interfaces fournies sont définies par l'ensemble $\mathbb{IF} = \{IF_i(E_j)/i, j \in \mathbb{N} \wedge E_j \in \mathbb{ELT}\}$

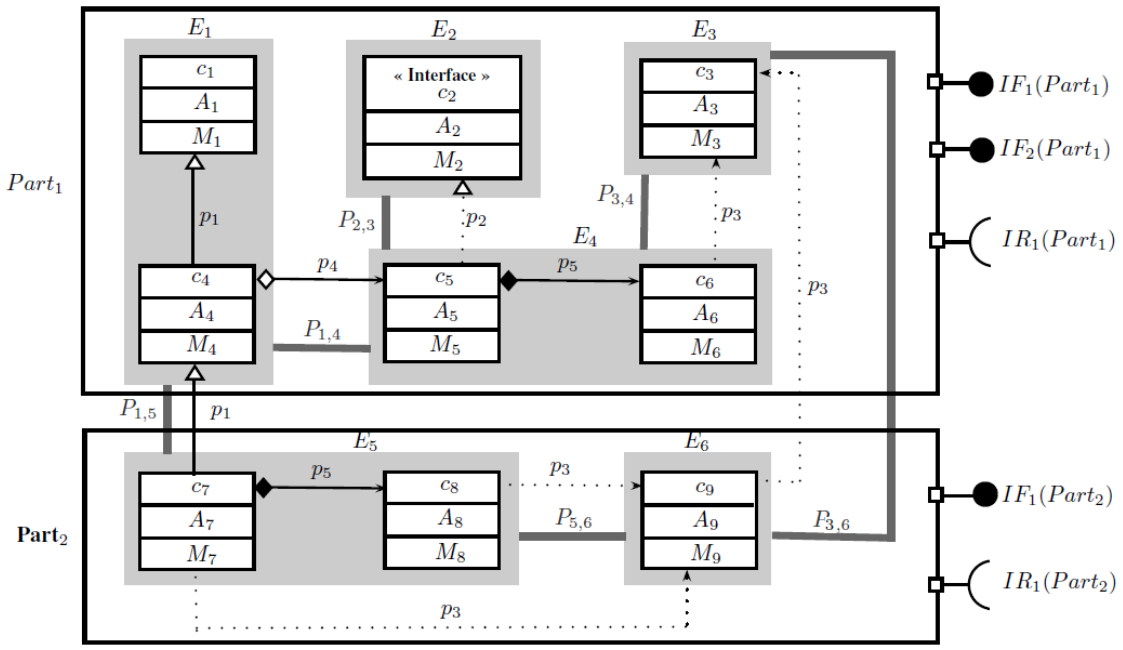

FIGURE 5.7 : Une pseudo-architecture.

Les interfaces requises, $\mathbb{IR}$

Une interface requise d'une partie définit un service dont la partie a besoin pour qu'elle fonctionne.

Représentation graphique d'une interface requise :
L'interface requise d'un partie est symbolisée par $\multimap($. Par exemple, à la figure 5.7, il existe une interface requise pour la partie $Part_1$ et une pour la partie $Part_2$.

Formalisme mathématique des interfaces requises :
Les interfaces requises sont définies par l'ensemble $\mathbb{IR} = \{IR_i(E_j)/i, j \in \mathbb{N} \wedge E_j \in \mathbb{ELT}\}$

5.4.8 L'architecture logicielle, $\mathbb{AL}$

Par définition, une architecture logicielle est la structure d'un système. Une structure comprend les éléments logiciels (tels que les classes, les composants,...), les relations entre eux (comme les relations de classes), et les propriétés des éléments et des relations. Ainsi, l'artefact architecture logicielle, $\mathbb{AL}$, est composé des informations suivantes :
- Une partition des éléments, qui sont les éléments logiciels selon la définition.
- Des relations entre les éléments, qui sont les relations selon la définition.
- Des interfaces fournies et requises, qui sont les propriétés selon la définition.

5.4.9 La description de l'architecture logicielle, $\mathbb{AD}$

Par définition, une description de l'architecture logicielle est un ensemble de produits qui documente une architecture d'une manière que les architectes peuvent comprendre et démontre que l'architecture a répondu à leurs préoccupations. Cette description fournit une image globale qui résume le système, et elle le décrit également dans suffisamment de détails. Ainsi, l'artefact

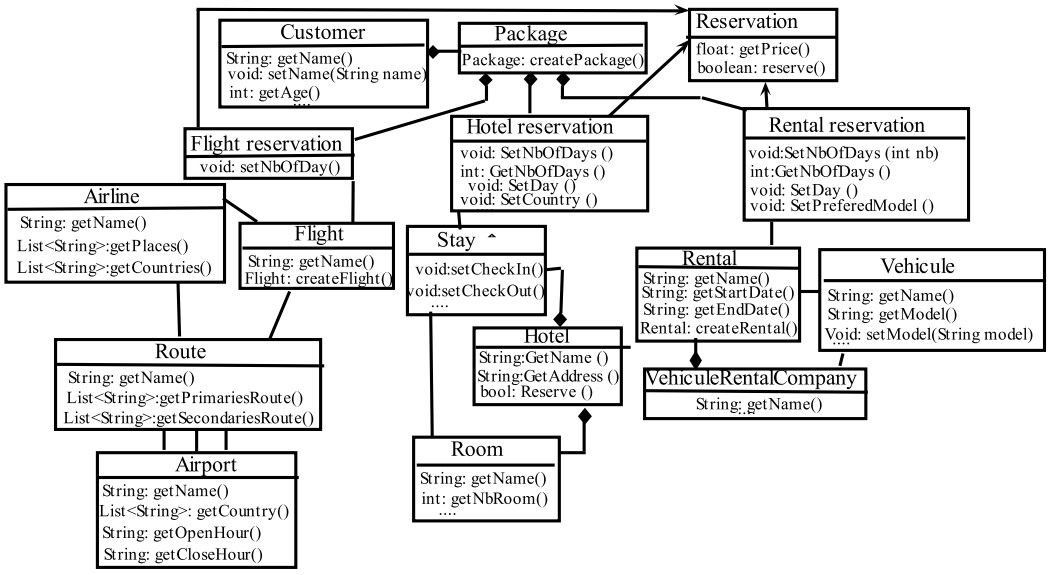

FIGURE 5.8 : Les classes, leurs opérations et relations du modèle source.

description de l'architecture logicielle, $\mathbb{AD}$, est un document détaillant l'architecture logicielle d'un système ; donc, elle détaille la partition des éléments, les relations entre les éléments, et les interfaces fournies et requises.

5.5 Les activités du modèle SAD

Une activité est une unité de travail ayant un objectif. Généralement, une activité utilise un artefact et/ou produit un autre et est exécutée par un rôle. Dans ce qui suit, nous détaillons les activités du modèle de processus SAD. Nous nous servons d'un cas d'étude qui est un système de réservation afin de clarifier les activités du modèle. Le système de réservation permet au client de réserver une location, un hôtel ou un vol. Les produits demandés par un client sont placés dans un paquet qu'il décide d'acheter à la fois. Chaque paquet est toujours acheté par un, et seulement client. Un client peut acheter plus d'un paquet.

5.5.1 L'extraction d'un modèle source

Input de l'activité : Un code source.
Output de l'activité : Un modèle source, $\mathbb{MS}$.

Cette activité consiste à analyser le code source afin de fournir un modèle source, $\mathbb{MS}$. L'analyse consiste à étudier les termes présents dans le code source selon le langage de programmation utilisé afin de déduire les constituants (les classes, leurs relation et les appels de méthodes) d'un modèle source. Pour notre cas d'étude, le modèle source produit à partir du code source du système de réservation est illustré dans deux figures : (1) La figure 5.8 qui représente les classes et les relations entre les classes et (2) la figure 5.9 qui représente les appels entre les méthodes. Pour arriver à ce résultat, cette activité a englobé les activités suivantes :

L'extraction des classes

L'extraction des classes, $\mathbb{CL}$, consiste à extraire les classes (leurs noms, attributs et méthodes) de différents types à partir du code source. Ainsi, cette activité inclut les activités suivantes :

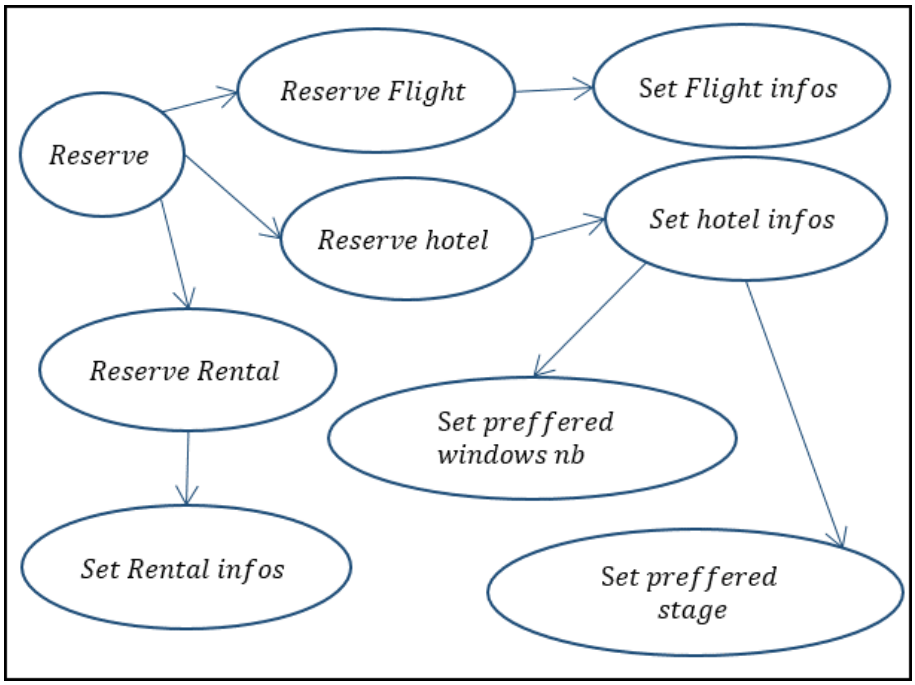

FIGURE 5.9 : Les appels des méthodes du modèle source représentés par un graphe d'appels.

- L'extraction des classes concrètes
  Une classe concrète $CL = (c_i, A_i, M_i, c\_concrete)$ est extraite à partir du code source si elle est déclarée dans le code source. Par exemple, si le code source est écrit en langage JAVA, la classe concrète est extraite du code si le nom de la classe est précédé par le mot-clé « class ».

- L'extraction des classes abstraites
  Une classe abstraite $CL = (c_i, A_i, M_i, c\_abstraite)$ est extraite à partir du code source si elle est déclarée dans le code source et marquée comme abstraite. Par exemple, si le code source est écrit en langage JAVA, la classe abstraite est extraite du code si le nom de la classe est précédé par le mot-clé « abstract class ».

- L'extraction des interfaces
  Une interface $CL = (c_i, A_i, M_i, c\_interface)$ est extraite à partir du code source si elle est déclarée dans le code source. Par exemple, si le code source est écrit en langage JAVA, l'interface est extraite du code si le nom de la classe est précédé par le mot-clé « interface ».

L'extraction des relations de classes

L'extraction des relations de classes, $\mathbb{REL}\_\mathbb{CL}$, consiste à identifier, à partir du code source, les classes ayant des relations entre elles et les types de relations. Ainsi, cette activité inclut les activités suivantes :

- L'extraction des relations de dépendance
  Une relation de dépendance, $r = (CL_1, CL2, r\_dependance)$, est extraite du code source entre une classe CL_1 et une classe CL_2 si une méthode de la classe CL_1 appelle une méthode de la classe CL_2.

- L'extraction des relations de généralisation
  Une relation de généralisation, $r = (CL_1, CL2, r\_generalisation)$, entre une classe CL_1 et

TABLE 5.2 : Le poids des relations selon leurs types.

Type	Weight
Composition	4
Agrégation	4
Généralisation	3
Association	2
Dépendance	1

une classe CL_2 est extraite du code source si la classe CL_2 contient dans sa déclaration le mot-clé « extends » suivi par le nom de la classe CL_1.

- L'extraction des relations de réalisation
  Une relation de réalisation, $r = (CL_1, CL2, r\_realisation)$, entre une classe CL_1 et une classe CL_2 est extraite du code source si la classe CL_2 contient dans sa déclaration le mot-clé « implements » suivi par le nom de la classe CL_1.

- L'extraction des relations d'agrégation
  Une relation d'agrégation, $r = (CL_1, CL2, r\_agregation)$, entre une classe CL_1 et une classe CL_2 est identifiée si la classe CL_1 contient dans sa déclaration un attribut de type CL_2.

- L'extraction des relations de composition
  Une relation de composition, $r = (CL_1, CL2, r\_composition)$, entre une classe CL_1 et une classe CL_2 est identifiée si la classe CL_1 contient dans sa déclaration un attribut de type CL_2 et cet attribut est instancié par la classe CL_1.

L'extraction des appels de méthodes

L'extraction des appels de méthodes consiste à identifier les relations de méthodes, $\mathbb{REL\_M}$. Une relation de méthodes, $rel\_m = (m_1, m_2)$, est identifiée entre une méthode m_1 et une méthode m_2 si la méthode m_1 appelle la méthode m_2.

5.5.2 Le Nettoyage du modèle source

Input de l'activité : Un modèle source, $\mathbb{MS}$
Output de l'activité : Un modèle source, $\mathbb{MS}$.

Cette activité consiste à nettoyer le modèle source des classes et des relations qui peuvent former un bruit lors de l'extraction.

5.5.3 La définition des poids des types des relations de classes

Input de l'activité : Pas d'input
Output de l'activité : Les poids des types de relation de classes, P.

Cette activité consiste à donner un poids pour chaque type de relation de classes. La table 5.2 donne un exemple de l'output de cette activité.

5.5.4 La génération d'un modèle source pondéré

Input de l'activité : un modèle source $\mathbb{MS}$ et des poids des types de relations de classes, $\mathbb{P}$
Output de l'activité : un modèle source pondéré, $\mathbb{MS\_P}$.

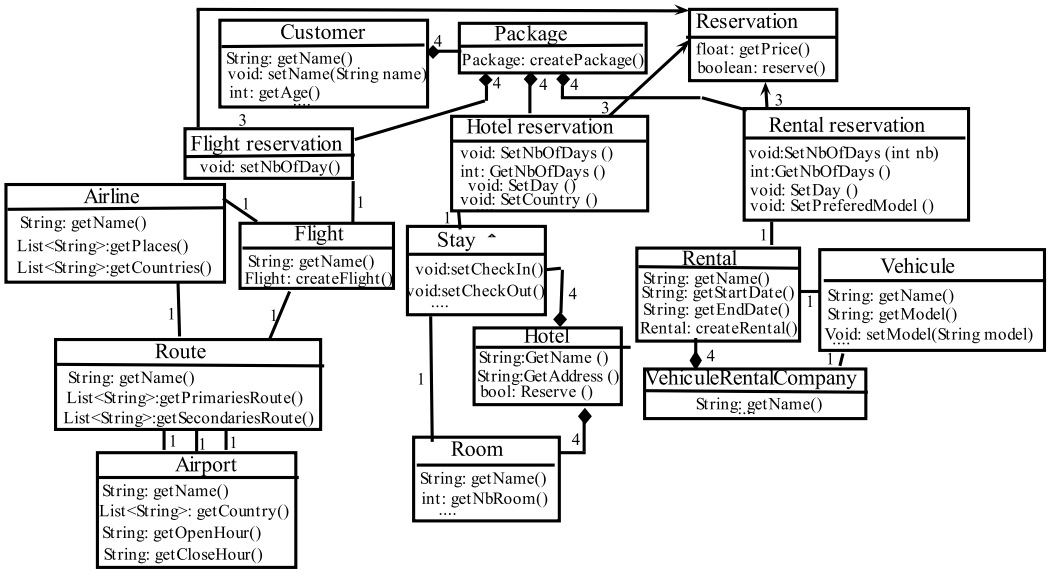

FIGURE 5.10 : Un modèle source pondéré extrait d'un système de réservation.

Cette activité consiste à élaborer un modèle source pondéré selon les poids des types de relations. Ainsi, pour chaque relation de classes dans le modèle source, un poids lui est donné selon son type. La figure 5.10 illustre le modèle source pondéré de notre cas d'étude ; les relations entre les classes sont pondérées selon la table 5.2.

5.5.5 Le groupement des classes reliées

Input de l'activité : un modèle source pondéré, $\mathbb{MS\_P}$, qui est l'output de l'activité 4.
Output de l'activité : les éléments et leurs connecteurs pondérés, $\mathbb{ELT\_CON}$.

Cette activité consiste à grouper les classes en des éléments selon un type de relation. Plus précisément, un type de relation est choisi par l'architecte, et les classes ayant ce type de relation entre elles sont regroupées en un élément. Concernant les connecteurs pondérés entre les éléments, leur élaboration dépend des relations entre les classes. Un connecteur pondéré est élaboré entre deux éléments s'il existe au moins une relation de classes entre deux classes appartenant chacun à un élément différent. La mesure utilisée pour pondérer un connecteur entre deux éléments est la somme des relations de classes entre les classes des deux éléments.

Par exemple, si l'architecte choisit la relation de composition pour grouper les classes du système de réservation, la sortie sera Figure 5.11. Les classes *Package, Customer, Flight reservation, Hotel reservation* et *Rental reservation* sont groupés en un élément E_1. Les classes *Rental* et *VehiculeRentalCompany* sont groupées en E_4. La classe *Reservation* n'a aucune relation de composition avec une autre classe, donc la classe *Reservation* est groupée seule dans un élément appelé E_9. Figure 5.11 montre les classes de chaque élément. En ce qui concerne les relations, une relation est établie entre E_1 et E_4 pondéré 1 car il existe une relation (dépendance) entre *Rental reservation* et *Rental* pondérée 1. En outre, une relation est établie entre E_1 et E_9 pondérés 9 car il existe trois relations de généralisation pondérées 3 entre les classes des éléments E_1 et E_9.

5.5.6 La classification des éléments

Input de l'activité : les éléments et leurs connecteurs pondérés, $\mathbb{ELT\_CON}$, qui sont l'output de l'activité 5.

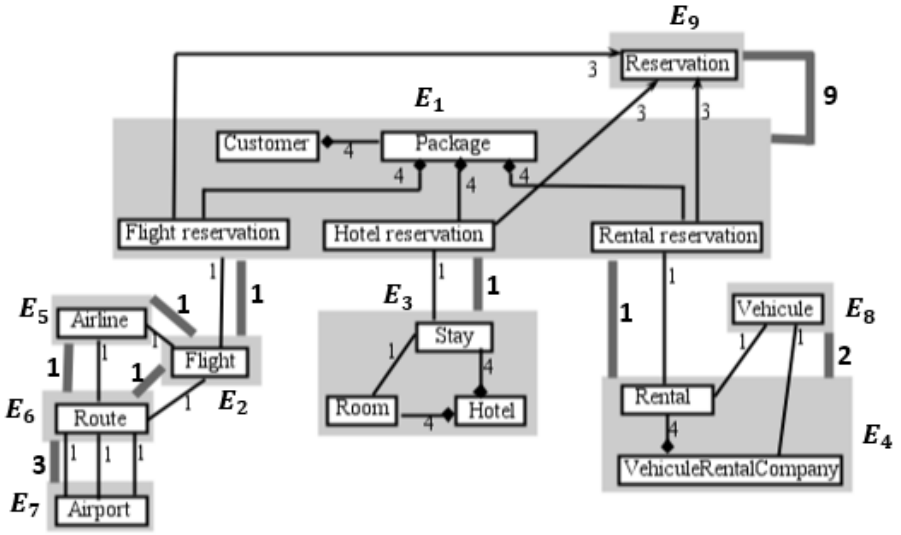

FIGURE 5.11 : Les éléments et leurs connecteurs pondérés.

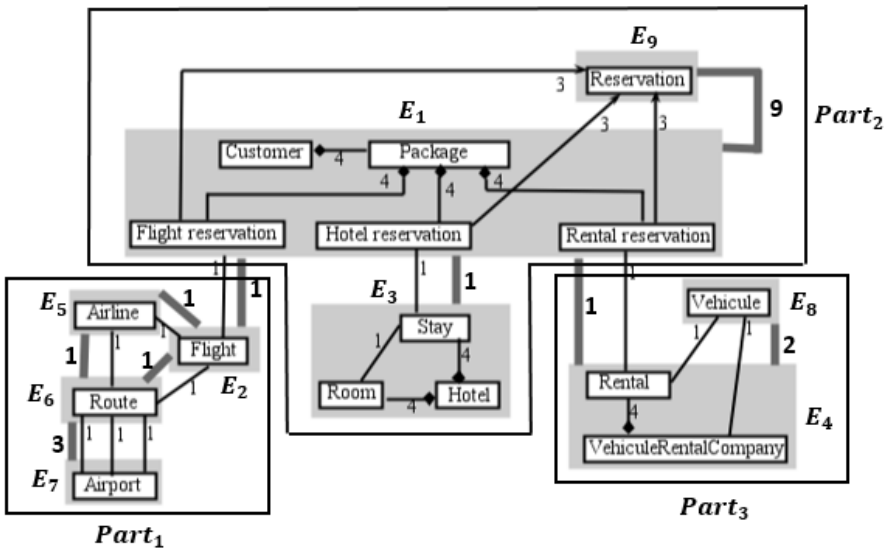

FIGURE 5.12 : Une pseudo-architecture.

Output de l'activité : Une partition des éléments, $\mathbb{PART}$.

Cette activité consiste à grouper les éléments élaborés dans l'activité 5 en des parties tout en tenant compte de deux critères importants : Avoir une forte cohésion entre les éléments d'une même partie et un couplage faible entre les éléments de deux parties différentes.

Pour lancer cette activité, l'architecte propose une architecture conceptuelle, autrement dit, il choisit le nombre de parties qui composent le système à son avis. Puis, l'architecte choisit la méthode (algorithme) de partitionnement tout en précisant certains paramètres. Par exemple, l'architecte choisit le partitionnement des éléments en utilisant la fonction MQ (présentée dans l'équation 2.1). Afin de générer une partition des éléments, une partition initiale est créée, puis, elle sera améliorée en tenant compte des deux critères. La figure 5.12 illustre une partition de trois parties, qui est l'output de cette activité pour le système de réservation.

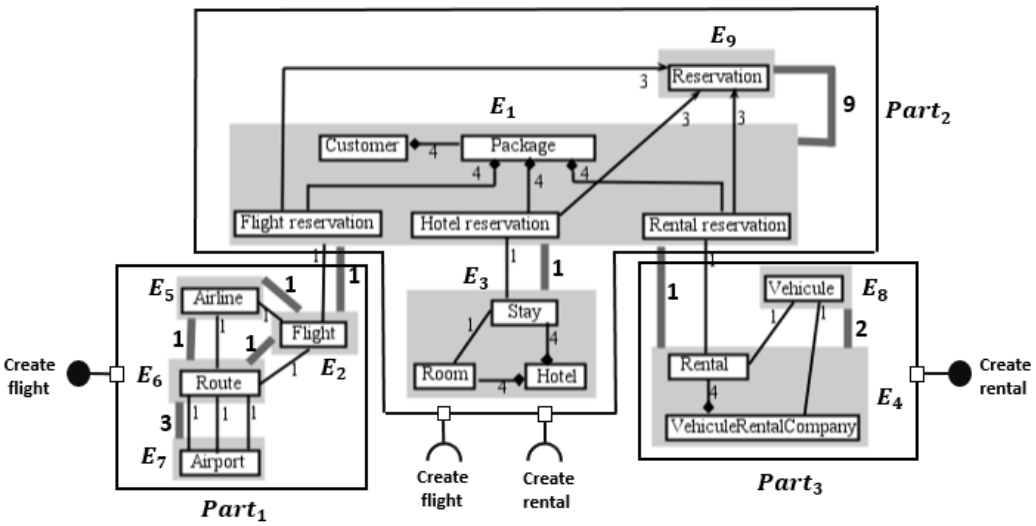

FIGURE 5.13 : Une pseudo-architecture.

5.5.7 L'identification des interfaces fournies et requises

Input de l'activité : Une partition des éléments, $\mathbb{PART}$, qui est l'output de l'activité 6.
Output de l'activité : Une pseudo-architecture, $\mathbb{PS\_A}$.

Cette activité consiste à identifier les interfaces fournies et requises des parties de la partition de l'activité 6. Une interface fournie $IF_i(Part_j)$ d'une partie $Part_j$ d'une partition $\mathbb{PART}$ est générée s'il existe une méthode $m \in M_i$ de la partie telle que m est appelée par une méthode appartenant à une classe d'une autre partie. Ainsi, une interface fournie d'une partie est le nom de la méthode appelée par au moins une méthode d'autres parties. Plus formellement, $IF_i(Part_j)$ est le nom de la méthode $m_1 \in M_l$ de la partie $Part_j$ appelée par une méthode $m_2 \in M_m$ de la partie $Part_k$.

Une interface requise $IR_i(Part_j)$ d'une partie $Part_j$ d'une partition $\mathbb{PART}$ est générée s'il existe une méthode de la partie qui appelle une autre méthode $m \in M_i$ appartenant à une classe d'une autre partie. Ainsi, une interface requise d'une partie est le nom de la méthode appelée par une méthode de la partie. Plus formellement, $IR_i(Part_j)$ est le nom d'une méthode $m_1 \in M_l$ de la partie $Part_k$ appelée par une méthode $m_2 \in M_m$ de la partie $Part_j$.

5.5.8 Le raffinement de la pseudo-architecture

Input de l'activité : Une pseudo-architecture, $\mathbb{PS\_A}$, qui est l'output de l'activité 7.
Output de l'activité : Une architecture logicielle, $\mathbb{AL}$.

Dans cette activité, l'architecte raffine la pseudo-architecture. L'architecte déplace les classes, les éléments ou raffine les interfaces. Par exemple, un architecte peut déplacer une classe d'un élément à l'autre.

5.5.9 La génération d'une description de l'architecture logicielle

Input de l'activité : Une architecture logicielle, $\mathbb{AL}$, qui est l'output de l'activité 8.
Output de l'activité : Une description de l'architecture logicielle, $\mathbb{AD}$.

Dans cette activité, un document décrivant l'architecture du système est généré. Ainsi, l'output de cette activité décrit la partition, les éléments de chaque partie, et leurs interfaces.

5.6 Les rôles

Les rôles sont les performants des activités. Dans notre modèle de processus d'extraction d'une architecture logicielle (SAD), les rôles sont les suivants :

5.6.1 Un architecte

Un architecte est celui qui souhaite découvrir une architecture de son système et qui est chargé d'extraire une architecture du système. Dans notre modèle SAD, l'architecte est responsable :

- De spécifier les critères de nettoyage du modèle source dans l'activité 2 (nettoyer le modèle source).
- De définir les poids des types de relations de classes dans l'activité 3.
- De choisir un type de relations dans l'activité 5 (Grouper les classes reliées).
- De proposer une architecture conceptuelle dans l'activité 6 (Classifier les éléments).
- De raffiner la pseudo-architecture.

5.6.2 Un analyseur d'un code source

L'analyseur d'un code source est un outil qui analyse les mots du code source pour extraire des informations du code. Dans notre modèle de processus, ce rôle est responsable d'extraire les classes, leurs relations et les appels des méthodes également. En d'autres termes, cette technique exécute l'activité 1 (Extraire un modèle source).

5.6.3 Une technique de jointure

La technique de jointure fait joindre deux ensembles ayant un certain lien entre eux. Cette technique est utilisée pour générer un modèle source pondéré, ce qui est l'activité 4 du modèle SAD. Donc, à partir du modèle source et des poids des types de relations, ce rôle donne un poids à chaque relation du modèle source.

5.6.4 Une technique de groupement

La technique de groupement groupe des éléments selon un critère. Cette technique est utilisée pour grouper les classes en des éléments selon un type de relation. Donc, dans notre modèle de processus, ce rôle est responsable de grouper les classes générées par l'activité 4 et d'élaborer les éléments et leurs relations.

5.6.5 Une technique de partitionnement

La technique de partitionnement génère une partition des éléments. Dans notre modèle SAD, cette technique est utilisée pour générer une partition des éléments selon deux critères : avoir un couplage faible entre les éléments d'une même partie et une cohésion forte dans les éléments d'une même partie.

5.6.6 Une technique d'analyse des relations de méthodes

Une technique d'analyse des relations de méthodes étudie les relations de méthodes d'une partition. Ces relations sont établies entre les méthodes qui appartiennent à deux parties différentes de la partition. Dans notre modèle SAD, cette technique est responsable de générer les interfaces des parties de la partition fournie par l'activité 6. Elle identifie les interfaces selon les relations de méthodes établies entre les méthodes impartissant à deux parties différentes.

5.6.7 Une technique de génération d'un document

La technique de génération d'un document donne une representation textuelle de certaines informations. Donc, dans notre modèle de processus, ce rôle est responsable de générer une description architecturale de l'architecture identifiée par l'activité 8.

5.7 La synthèse

Nous évaluons le modèle SAD selon le modèle de qualité proposé dans le chapitre 3. Plus précisément, l'évaluation est établie sur le facteur méthode (voir table 5.3). Tout d'abord, SAD permet à l'architecte d'intégrer ses connaissances en proposant une architecture conceptuelle et en lui permettant de pondérer les poids des relations et de modifier après chaque étape les artefacts du processus. De plus, un architecte peut interagir avec le processus vu qu'il peut raffiner les artefacts après chaque étape, et choisir certains paramètres comme ceux de la méthode de partitionnement. Ensuite, concernant les types des informations extraites, notre modèle extrait les classes, leur attributs et méthodes et leurs relations, et non pas le flux de données durant l'exécution, ce qui est de qualité moyenne.

Ce qui est le plus important dans notre modèle, est qu'il est basé sur la stratégie de correspondance entre les groupes des classes (les éléments) et une architecture conceptuelle proposée par l'architecte, qui est l'architecture prévue. Le modèle permet à un architecte de fournir une architecture conceptuelle, puis, d'une manière automatique les groupes des classes sont assignés à l'architecture conceptuelle. Cette classification des éléments dans l'architecture conceptuelle est raffinée par la technique de partitionnement. Ainsi, notre modèle groupe les entités logicielles d'une part et fait correspondre les groupes à l'architecture conceptuelle. Et finalement, une architecture complète en termes d'éléments et d'interfaces est générée.

5.8 Conclusion

Dans ce chapitre, nous avons présenté un modèle de processus SAD, en spécifiant ses activités, artefacts et rôles. Principalement SAD répond aux limitations existantes dans d'autres approches, surtout celles concernant l'intégration et l'interaction de l'architecte durant le processus. Ainsi, notre modèle permet à un architecte de construire son processus d'extraction et de découvrir progressivement une architecture qui le satisfait.

Ce modèle de processus est supporté par un outil automatique, qui est un outil étendu de d'ECD. Dans le chapitre suivant, nous détaillons les relations qui existent entre notre modèle et le principe d'ECD et nous présentons également une extension d'un outil ECD qui permet l'exécution automatique d'un processus ECD.

TABLE 5.3 : La projection du modèle SAD sur le modèle de comparaison.

Facteurs	**Critères**	**Métriques**	Q
Méthode	Intégration des connaissances de l'architecte	Intégration des connaissances de l'architecte en lui permettant de proposer une architecture conceptuelle et d'ajouter d'autres informations	B
	Interaction de l'architecte	Interaction de l'architecte en lui permettant de raffiner les artefacts et configurer le processus selon ses besoins	B
	Types des informations	des informations ayant une signification moyenne	MY
	Groupement des entités logicielles	Groupement deux fois (un pour les classes, et l'autre pour les éléments)	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Correspondance selon les deux critères	B
	Identification d'une architecture logicielle complète	une architecture logicielle en termes d'éléments architecturaux et leurs relations et propriétés (interfaces)	B

6

Une extension de KDD pour l'extraction des architectures logicielles

6.1 Introduction

Dans le chapitre précédent, nous avons proposé un modèle de processus d'extraction d'une architecture logicielle nommé SAD *(Software Architecture Discovery)*. Cependant, notre but est de fournir une méthode qui aide un architecte à construire son processus d'extraction d'une architecture logicielle et extraire une architecture logicielle qui le satisfait. Ainsi, nous présentons dans ce chapitre notre approche d'extraction qui consiste à exécuter un processus d'extraction comme étant un processus de découverte de nouvelles connaissances. Nous présentons tout d'abord dans ce chapitre une étude des relations qui existent entre le modèle de processus SAD, qui est un modèle générique proposé dans le chapitre 5, et le modèle de processus ECD *(Extraction de Connaissances à partir de Données)*. Grâce à cette étude, nous suggérons qu'un processus SAD peut être exécuté comme un processus ECD. Finalement, nous présentons dans ce chapitre une extension d'un outil ECD, nommé KNIME, qui complète notre approche et permet à un architecte d'exécuter une instance du modèle SAD.

6.2 Le modèle de processus ECD

Le terme ECD (Extraction de Connaissances à partir de Données), ou KDD en anglais (Knowledge Discovery in Databases), est utilisé depuis 1989. C'est le résultat de la convergence de recherches en apprentissage automatique, reconnaissance de formes, bases de données, statistique, intelligence artificielle, visualisation de données, etc. Plusieurs définitions existent pour souligner le concept ECD. D'après Han et Kamber [Han et Kamber, 2006], le processus ECD est l'analyse de la base de données, ayant souvent une grande taille, afin de découvrir de relations insoupçonnées et de résumer les données d'une manière à la fois compréhensible et utile. Dans [Fayyad et al., 1996a], Fayyad a défini ECD comme l'ensemble de processus d'extraction de connaissances utiles à partir de données. Ce processus vise à transformer des données volumineuses, multiformes et stockées sous différents formats en connaissances qui peuvent s'exprimer sous forme d'un concept général : un rapport ou un graphique par exemple.

Il existe une certaine confusion au sujet du terme ECD et le terme data mining (l'exploration de données). L'extraction de connaissances à partir de données telle qu'elle a été définie selon Fayyad [Fayyad et al., 1996a] est un processus consistant à identifier des informations et des relations valides, nouvelles, potentiellement utiles, et compréhensibles dans les données. Cependant, l'exploration de données (data mining) est définie comme la recherche de pépites d'information utiles dans un grand ensemble de données. Bien que les deux définitions semblent être similaires, l'exploration de données n'est qu'une petite partie (estimée de 15% à 25%) du processus global d'ECD.

Un modèle de processus ECD consiste en un ensemble d'étapes à suivre ; il prend comme input des données et fournit comme output final des nouvelles connaissances. La figure 6.1 résume le concept des modèles ECD. Selon l'étude fournie dans [Kurgan et Musilek, 2006], le modèle ECD le plus connu et qui est considéré comme base pour d'autres modèles a été proposé par Fayyad et al. [Fayyad et al., 1996a]. Depuis lors, différents modèles ECD ont été développés dans les milieux universitaires et industriels [Cabena et al., 1998, Anand et Büchner, 1998, Shearer, 2000, Wirth et Hipp, 2000, Cios et Kurgan, 2005]. En général, tous les modèles de processus se composent de plusieurs étapes exécutées dans une séquence, qui contient souvent des boucles et des itérations. Chaque étape est initiée lors de la réussite de l'étape précédente, et nécessite un résultat généré par l'étape précédente comme input. Tous les modèles proposés mettent également l'accent sur la nature itérative et interactive du modèle, en termes de nombreuses boucles de rétroaction et de répétitions, déclenchées par un processus de révision.

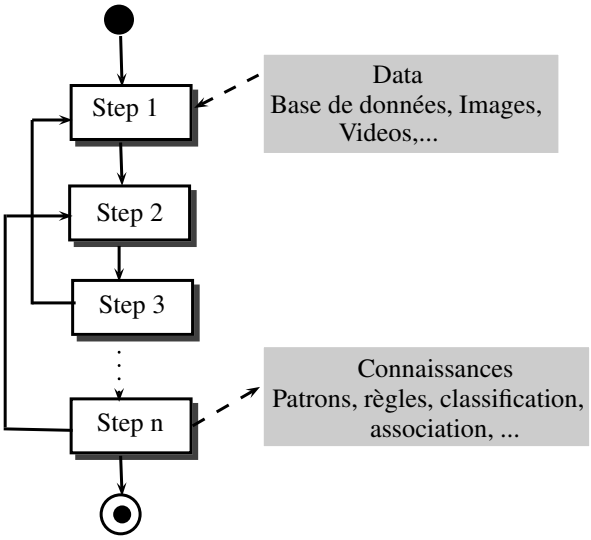

FIGURE 6.1 : Un diagramme d'activité qui illustre le concept général d'un modèle de processus ECD.

6.3 Le modèle de processus ECD selon Fayyad et al.

Selon Fayyad et al., parmi les caractéristiques d'ECD, l'on trouve un processus interactif et itératif impliquant de nombreuses étapes avec des différentes décisions prises par l'utilisateur [Fayyad et al., 1996a] (voir figure 6.2). Afin de découvrir des connaissances significatives, les principales étapes d'ECD effectuent la préparation et la sélection des données, le nettoyage des données, l'exploration de données, l'interprétation des données ... Selon Fayyad, qui est le principal contributeur dans le domaine d'ECD, ECD est composé de neuf étapes qui sont

décrites comme suit (nous résumons les étapes dans le diagramme d'activité présenté à la figure 6.3) :

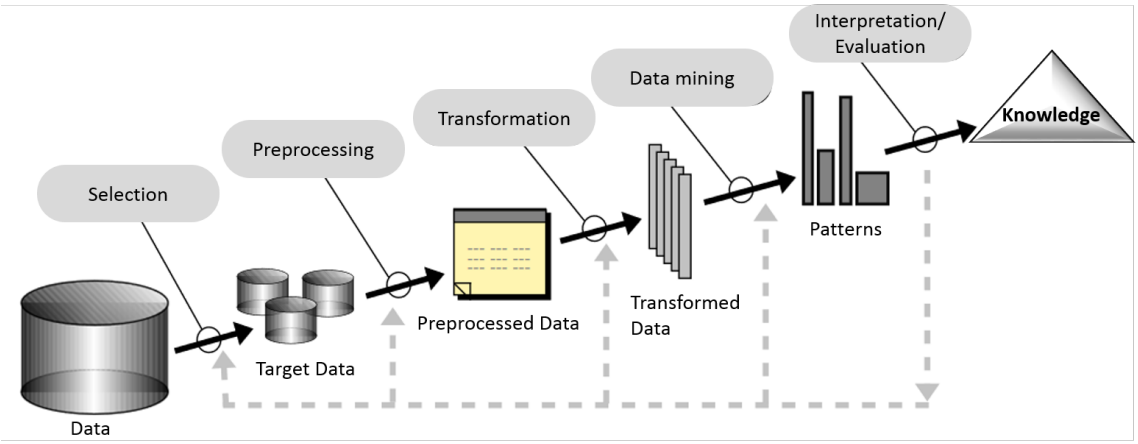

FIGURE 6.2 : Le processus d'extraction de connaissances à partir de données : extrait de [Fayyad et al., 1996a].

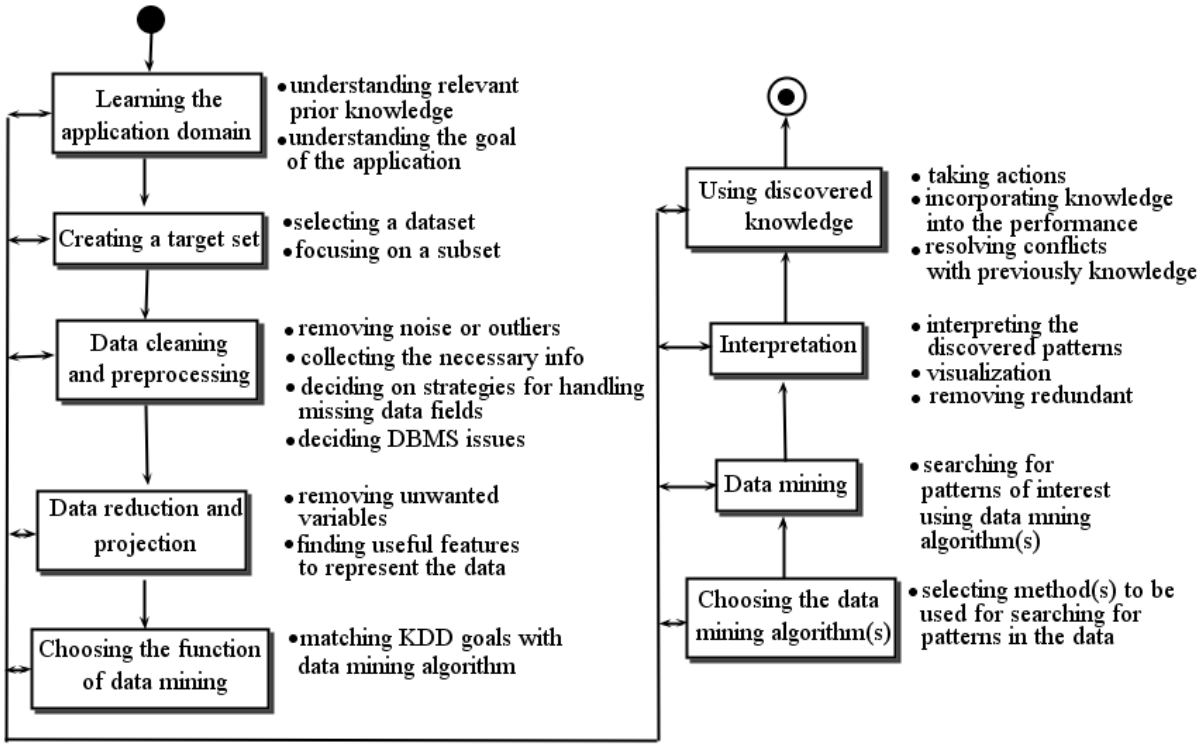

FIGURE 6.3 : Un diagramme d'activité qui illustre le modèle du processus ECD.

- Étape 1 : La compréhension du domaine d'application
  Cette étape inclut la compréhension du domaine de l'application, la découverte des connaissances antérieures pertinentes et la définition de l'objectif du processus ECD.

- Étape 2 : La création de l'ensemble de données cible
  Cette étape inclut la sélection d'un ensemble de données ou la concentration sur un sous-ensemble de variables ou d'échantillons de données. En addition, elle inclut l'intégration de données si les données sont de différentes ressources hétérogènes. L'output final de cette étape est l'ensemble des données sur lequel la découverte doit être effectuée.

- Étape 3 : Le nettoyage et pré-traitement des données
  Cette étape inclut la suppression du bruit ou des valeurs aberrantes, la collecte des informations nécessaires pour modéliser ou traiter le bruit, l'utilisation des connaissances antérieures pour supprimer les incohérences et les doublons à partir des données, le choix ou l'utilisation de stratégies pour gérer les champs de données manquantes. En fait, la plupart des données recueillies sont incomplètes et inconsistantes, ce qui aboutit à confondre la suite du processus et conduit à des résultats non fiables et non valides. Pour cela, cette étape doit être établie pour améliorer la qualité des données et arriver à un bon résultat comme output final.

- Étape 4 : La réduction des données et leur projection
  Cette étape inclut la recherche des fonctionnalités utiles pour représenter les données, en fonction de l'objectif du processus. Plus spécifiquement, des techniques de transformation sont utilisées dans cette étape pour rendre les données plus appropriées pour l'exploitation comme l'agrégation et la normalisation... Des techniques de réduction sont appliquées pour produire une représentation réduite des données (avec un nombre de variables réduit) comme l'agrégation et la réduction de dimension.

- Étape 5 : Le choix de la fonction d'exploration de données (data mining)
  Cette étape inclut la décision de la fonction de l'algorithme d'exploration de données. Plus précisément, cette décision consiste à choisir parmi la récapitulation, la classification, la régression, et le regroupement.

- Étape 6 : Le choix de(s) l'algorithme(s) d'exploration de données
  Cette étape inclut d'une part la sélection de la/des méthode(s) à utiliser pour la recherche de patrons dans les données, et, d'autre part, elle inclut des décisions concernant les modèles, les paramètres, et les fonctions les plus appropriés pour la recherche.

- Étape 7 : L'exploration de données *(data mining)*
  Cette étape inclut l'application des algorithmes d'exploration de données choisis à l'ensemble de données cibles et la recherche des modèles d'intérêt dans une forme de représentation particulière ou dans un ensemble de représentations. Ainsi, cette étape consiste à exécuter les algorithmes sur les données pour chercher des nouvelles modèles.

- Étape 8 : L'interprétation
  Cette étape inclut l'interprétation des modèles découverts et peut-être revenir à l'une des étapes précédentes, aussi la visualisation des modèles extraits. En addition, cette étape peut inclure la suppression des modèles redondants ou non pertinents, et la traduction des modèles utiles en un format compréhensible par les utilisateurs.

- Étape 9 : L'utilisation de connaissances découvertes
  Cette étape inclut l'incorporation des connaissances découvertes dans la performance du système, ou simplement la documentation des parties intéressées, ainsi que la vérification et la résolution des conflits potentiels avec les connaissances existantes.

6.4 Le modèle de processus ECD et le modèle de processus d'extraction d'une architecture logicielle SAD

Un modèle de processus ECD souligne les étapes d'extraction des nouvelles connaissances à partir de données ; ceci est presque le concept du modèle de processus d'extraction d'une architecture logicielle SAD, proposé dans le chapitre 5. En fait, le modèle SAD n'est qu'une

séquence d'activités ayant comme objectif la découverte d'une architecture logicielle à partir des informations disponibles comme le code source. En partant de cette définition, nous pensons qu'un processus SAD peut être planifié comme un processus ECD. Plus clairement, nous pensons qu'un processus SAD n'est autre qu'une exécution d'un processus ECD tout en prenant le code source, la documentation existante, les informations de l'architecte comme données initiales, et fournissant l'architecture logicielle comme output final. La figure 6.4 résume cette idée. Pour valider notre suggestion, nous présentons dans la suite de cette section les relations qui existent entre le modèle de processus d'extraction SAD (proposé dans le chapitre 5) et le modèle de processus ECD. Cette étude de relations entre les deux modèles est effectuée en élaborant des relations entre les entités des deux modèles (comme des relations d'héritage et de composition). Ainsi, l'étude est divisée en trois parties : La première étudie les relations entre les entités de type activités, la deuxième étudie les relations entre les entités de type artefacts, et la troisième étudie les relations entre les entités de type rôles.

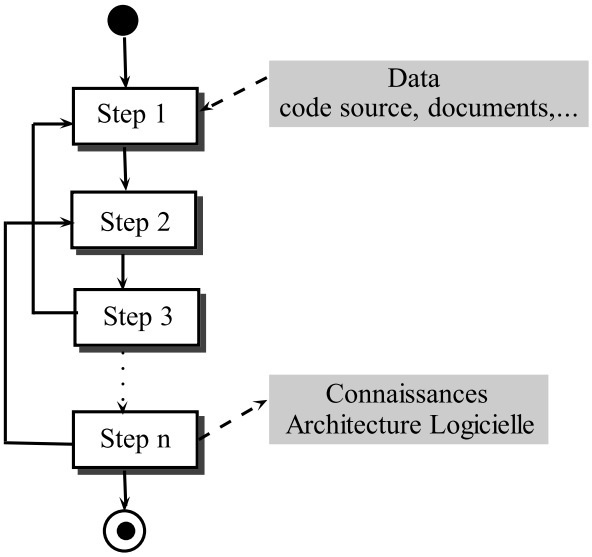

FIGURE 6.4 : Un diagramme d'activité qui illustre le concept général du modèle de processus d'extraction SAD.

6.4.1 Les relations entre les activités du modèle SAD et les étapes du modèle ECD

Dans cette section, nous présentons les relations entre les entités du modèle ECD, présentées dans la section 6.3, et les activités du modèle SAD, présentées dans la section 5.5 du chapitre 5. Plus précisément, nous expliquerons comment chaque activité du modèle de processus d'extraction SAD peut être considérée une activité du modèle ECD.

L'extraction d'un modèle source

L'extraction d'un modèle source dans le modèle SAD est une sorte de création de l'ensemble de données cible dans le modèle ECD. Une relation de généralisation/spécialisation existe entre ces deux activités (voir figure 6.5). En fait, dans l'activité d'extraction d'un modèle source, un modèle source est produit. Ce modèle représente les informations (comme les classes, les relations...) à partir desquelles l'architecture logicielle doit être découverte. D'autre part, la création de l'ensemble de données cibles dans le modèle ECD consiste à sélectionner l'ensemble

de données sur lesquelles la découverte doit être effectuée. Ainsi, l'extraction des informations (comme les classes, les méthodes...) à partir des données (les fichiers du code source) est une création d'un ensemble de données sur lesquelles la découverte doit être effectuée.

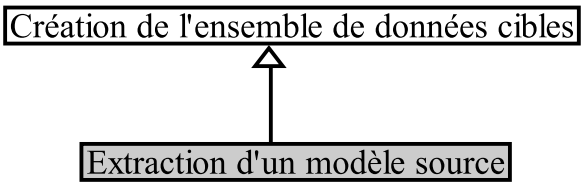

FIGURE 6.5 : Les relations entre l'extraction d'un modèle source et la création de l'ensemble de données cible.

Le nettoyage du modèle source

Le nettoyage du modèle source dans le modèle SAD est un nettoyage et pré-traitement des données dans le modèle ECD. Une relation de généralisation/spécialisation existe entre l'activité 'nettoyage du modèle source' et l'activité 'nettoyage et pré-traitement des données' (voir figure 6.6). En fait, dans l'activité de nettoyage du modèle source, l'architecte supprime des informations du modèle source qui peuvent fournir du bruit sur le processus d'extraction et traite le données, par exemple, il supprime une classe ayant des relations avec presque toutes les autres classes. D'autre part, le nettoyage et pré-traitement des données dans le modèle ECD consiste à supprimer des données qui peuvent fournir un bruit sur le processus d'extraction de connaissances à partir des données. Ainsi, le fait de nettoyer le modèle source est un nettoyage et pré-traitement des données.

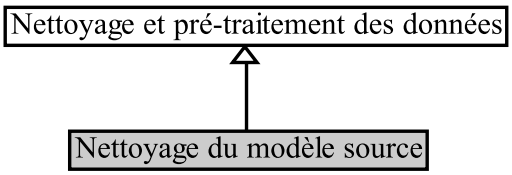

FIGURE 6.6 : Les relations entre le nettoyage du modèle source et le nettoyage et pré-traitement des données.

La définition des poids des relations

La définition des poids des relations dans le modèle SAD est une projection des données dans le modèle ECD. Une relation de généralisation/spécialisation existe entre l'activité 'définition des poids des relations' et l'activité 'projection des données' (voir figure 6.7). En fait, dans l'activité de définition des poids des relations, l'architecte définit les poids des relations de classes. Ces nouvelles informations rendent les types de relations plus appropriés à la découverte d'une architecture logicielle. Ceci signifie que dans cette activité, les données (qui sont les types de relations) sont représentées d'une manière plus convenable à la découverte d'une architecture logicielle. D'autre part, la réduction des données et leur projection dans le modèle ECD est composée de deux étapes qui sont (1) la réduction des données et (2) la projection des données. Cette deuxième étape consiste à représenter les données en fonction des objectifs du processus pour qu'elles deviennent plus appropriées à l'extraction. Ainsi, l'activité de définition des

poids des relations est une projection des données (qui est une étape composante de l'étape de réduction des données et leur projection).

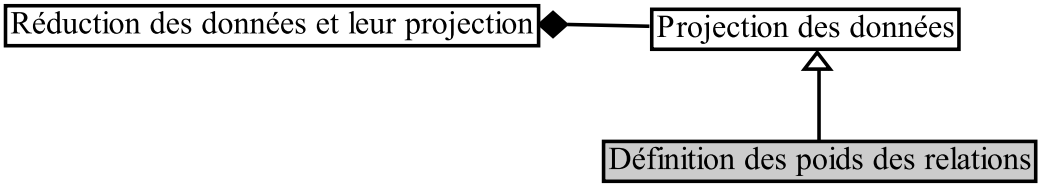

FIGURE 6.7 : Les relations entre la définition des poids des relations et la projection des données.

La génération d'un modèle source pondéré

La génération d'un modèle source pondéré dans le modèle SAD est aussi une projection des données dans le modèle ECD. Une relation de généralisation est munie de l'activité 'génération d'un modèle source pondéré' et l'activité 'projection des données' (voir figure 6.8). En fait, dans l'activité de génération d'un modèle source pondéré, les données existantes (qui sont les poids des relations de classes et le modèle source) sont réunies pour former des données ayant un sens dans le domaine d'extraction d'une architecture logicielle. Donc, ces nouvelles informations, qui constituent un modèle source pondéré, ne sont autres qu'un résultat de transformation du modèle source et des poids des types de relations de classes en un modèle source pondéré, qui est plus approprié pour la découverte d'une architecture logicielle. Ainsi, l'activité de génération d'un modèle source pondéré est une projection des données (qui est une étape composante de l'étape de réduction des données et leur projection).

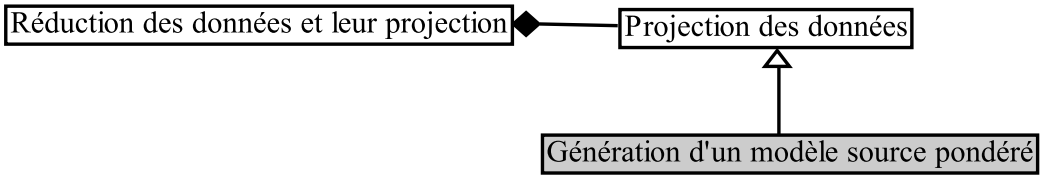

FIGURE 6.8 : Les relations entre la génération d'un modèle source pondéré et la projection des données.

Le groupement des classes reliées

Le groupement des classes reliées dans le modèle SAD est composée de deux étapes qui sont (1) le choix d'un type de relation de classes et (2) le groupement des classes selon le type choisi. Ces deux étapes sont successivement le choix de l'algorithme de découverte et l'exploration de données. Deux relations de généralisation/spécialisation existent : La première entre l'activité 'choix d'un type de relation de classes' et l'activité 'choix de l'algorithme de découverte' et la deuxième entre l'activité 'groupement des classes selon le type choisit' et l'activité 'exploration de données'(voir figure 6.9). En fait, dans l'activité de groupement des classes reliées, les classes ayant un type donné de relations entre elles sont groupées dans des éléments. Donc, l'exécution de cette activité nécessite la réalisation de deux étapes qui sont les suivantes :

- La première étape consiste à choisir un type de relation de classes : Dans cette activité, l'architecte choisit le paramètre (le type de relation) selon lequel le groupement doit être

établi. Vu que l'activité du choix de l'algorithme de découverte dans le modèle ECD inclut le fait de choisir des paramètres de l'algorithme, on peut dire que le choix du type de relation de classes est le choix de l'algorithme de découverte.

- La deuxième étape consiste à grouper les classes selon le type choisi : Dans cette activité des nouveaux patrons, qui sont les éléments avec leurs connecteurs, sont générés à partir des données. D'autre part, l'activité d'exploration de données du modèle ECD inclut la découverte des nouveaux patrons à partir des données. Ainsi, le groupement des classes selon le type choisi est une exploration de données.

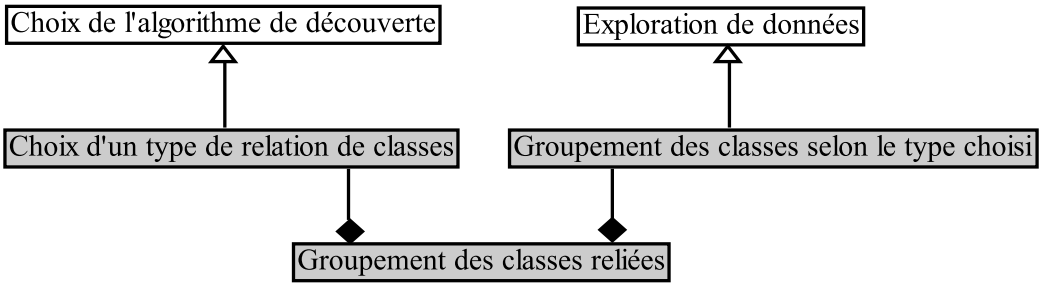

FIGURE 6.9 : Les relations entre le groupement des classes reliées d'une part et le choix de l'algorithme de découverte et l'exploration de données d'autre part.

La classification des éléments

La classification des éléments dans le modèle SAD est composée de trois étapes qui sont la proposition d'une architecture conceptuelle , (2) le choix de l'algorithme de partitionnement avec ses paramètres et (3) l'exécution de l'algorithme choisi. Les deux premières étapes sont des choix de l'algorithme de découverte, et, la troisième étape est l'exploration de données. Trois relations de généralisation/spécialisation existent : La première entre l'activité 'proposition d'une architecture conceptuelle' et l'activité 'choix de l'algorithme de découverte', la deuxième entre l'activité 'choix de l'algorithme avec ses paramètres' et l'activité 'choix de l'algorithme de découverte' et la troisième entre l'activité 'exécution de l'algorithme choisi' et l'activité 'exploration de données'(voir figure 6.9). En fait, dans l'activité de classification des éléments, les éléments sont groupés pour former une partition ayant une cohésion forte entre les éléments d'une même partie de la partition et un coupage faible entre les éléments de deux parties différentes. Donc, l'exécution de cette activité nécessite la réalisation de trois étapes qui sont les suivantes :

- La première étape consiste à proposer une architecture conceptuelle : Dans cette activité, l'architecte choisit le nombre de parties qui, à son avis, composent le système. D'autre part, l'activité choisir l'algorithme de découverte dans le modèle ECD inclut le choix des paramètres pour l'algorithme. Ainsi, la proposition d'une architecture conceptuelle est un choix l'algorithme de découverte.

- La deuxième étape consiste à choisir l'algorithme de partitionnement avec ses paramètres : Dans cette activité, l'architecte choisit l'algorithme qui doit générer une partition des éléments et configure également les paramètres de l'algorithme. Par la suite, le choix de l'algorithme de partitionnement avec ses paramètres est comme la première étape, un choix de l'algorithme de découverte.

- La troisième étape consiste à exécuter l'algorithme choisi : Dans cette activité un nouveau patron, qui est la partition, est généré à partir des données. D'autre part, l'activité d'exploration de données consiste à découvrir des nouveaux patrons à partir des données. Ainsi, l'activité d'exécution de l'algorithme choisi est une exploration de données.

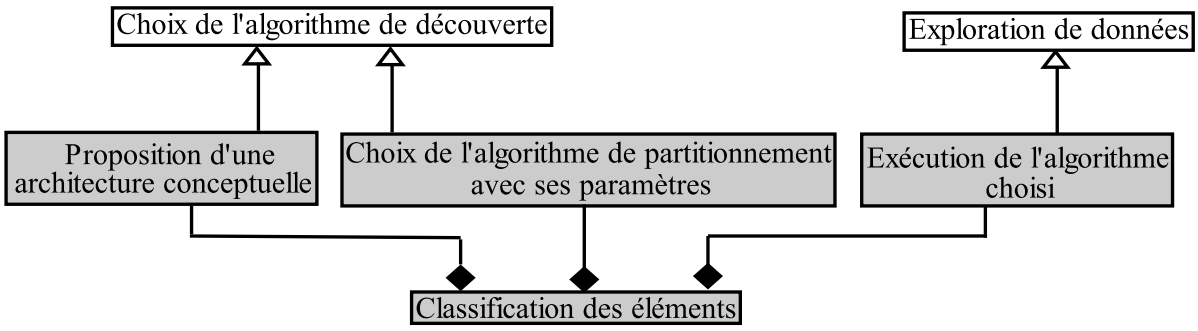

FIGURE 6.10 : Les relations entre la classification des éléments d'une part et le choix de l'algorithme de découverte et l'exploration de données d'autre part.

L'identification des interfaces fournies et requises

L'identification des interfaces fournies et requises dans le modèle SAD est l'exploration de données dans le modèle ECD. Une relation de généralisation/spécialisation existe entre l'activité 'identification des interfaces fournies et requises' et l'activité 'exploration de données' (voir figure 6.11). En fait, dans cette activité des nouveaux patrons, qui sont des interfaces fournies et requises, sont générés à partir des données. Ainsi, l'activité d'identification des interfaces fournies et requises est une exploration de données.

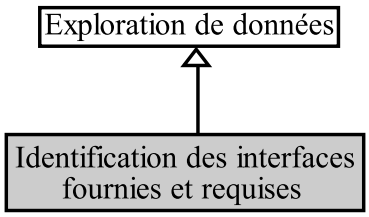

FIGURE 6.11 : Les relations entre l'identification des interfaces fournies et requises et l'exploration de données.

Le raffinement de la pseudo-architecture

Le raffinement de la pseudo-architecture dans le modèle SAD est l'interprétation dans le modèle ECD. Une relation de généralisation est munie de l'activité 'raffinement de la pseudo-architecture' vers l'activité 'interprétation' (voir figure 6.12). En fait, dans l'activité de raffinement de la pseudo-architecture, la pseudo-architecture est évaluée par l'architecte. Celle-ci peut invoquer des modifications dans la pseudo-architecture comme le déplacement des éléments d'une partie de la partition à l'autre et peut invoquer aussi un retour à l'une des activités du processus. D'autre part, l'activité interprétation inclut l'interprétation des résultats et invoque le retour à l'une des étapes précédentes. Ainsi, l'activité de raffinement de la pseudo-architecture est une interprétation.

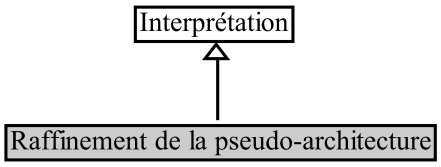

FIGURE 6.12 : Les relations entre le raffinement de la pseudo-architecture et l'interprétation.

La génération d'une description de l'architecture logicielle

La génération d'une description de l'architecture logicielle dans le modèle SAD est aussi l'interprétation dans le modèle ECD. Une relation de généralisation est munie de l'activité 'génération d'une description de l'architecture logicielle' vers l'activité 'interprétation' (voir figure 6.13). En fait, dans l'activité de génération d'une description de l'architecture logicielle, une description de l'architecture est élaborée. Celle-ci n'est autre qu'une visualisation de l'architecture du système. D'autre part, l'activité interprétation inclut la visualisation des résultats. Ainsi, l'activité de génération d'une description de l'architecture logicielle est une interprétation.

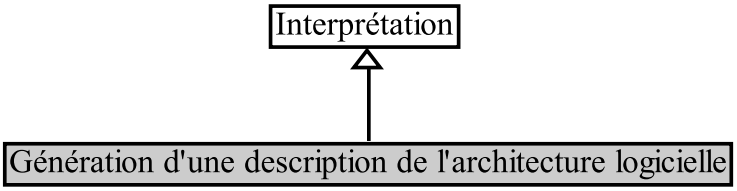

FIGURE 6.13 : Les relations entre la génération d'une description de l'architecture logicielle et l'interprétation.

6.4.2 Les relations entre les artefacts du modèle SAD et ceux du modèle ECD

Dans la section précédente, nous avons présenté comment les différentes activités définies dans le modèle SAD peuvent être appliquées en tant qu'étapes dans le processus ECD. Cependant, le modèle de processus SAD ne se limite pas aux concepts d'activités. SAD est modélisé comme une collaboration entre les activités d'extraction, les artefacts d'extraction et les rôles d'extraction. Ainsi, nous étudions dans cette section les relations qui existent entre les artefacts du modèle ECD et ceux du modèle SAD présentés dans la section 5.4. Néanmoins, les artefacts présentés dans le modèle ECD ne sont pas directement exprimés, ils sont mentionnés implicitement dans le modèle avec un faible niveau de détail. Mais, nous pouvons encore les révéler. Pour cela nous présentons dans ce qui suit, les artefacts qu'on a dérivé du modèle ECD. Puis, nous détaillons les relations entre les artefacts du modèle SAD et ceux dérivés du modèle ECD.

Les artefacts dérivés du modèle ECD

Les artefacts dérivés du modèle ECD sont des artefacts manipulés durant un processus ECD. Ces artefacts ne sont pas définis directement dans le modèle ECD.

- Les données
  Cet artefact représente toutes sortes de données, par exemple les supports électroniques, fichiers multiples ou base de données de type data warehouse...

- Les données optimales
  Cet artefact représente les données importantes ; c'est comme une échantillon des données, ou une partie d'elles.

- Les données nettoyées
  Cet artefact représente les données sans le bruit ; c'est une partie des donnés optimales ayant le bruit ou les valeurs aberrantes supprimées.

- Les données plus appropriées pour l'exploration de données
  Cet artefact représente les données dans une manière plus convenable au processus d'extraction. Généralement, ces données représentent les données nettoyées d'une manière plus convenable à la découverte des nouvelles connaissances.

- Les modèles
  Cet artefact décrit des faits *(facts)*. Généralement, un modèle représente des certains connaissances, forme ou caractéristiques d'une chose.

- L'interprétation
  Cet artefact représente une explication d'une chose ou un fait. Généralement, l'interprétation donne un un sens à un acte ou à un fait, dont l'explication n'apparaît pas de manière évidente.

- Les connaissances
  Cet artefact décrit une compréhension de quelque chose. Ainsi, il représente des informations utiles, valides et compréhensibles.

Les relations entre les artefacts des deux modèles

Dans cette section, nous présentons les relations qui existent entre les artefacts du modèle SAD et ceux du modèle ECD élaborés dans la section précédente. La figure 6.14 illustre ces relations.

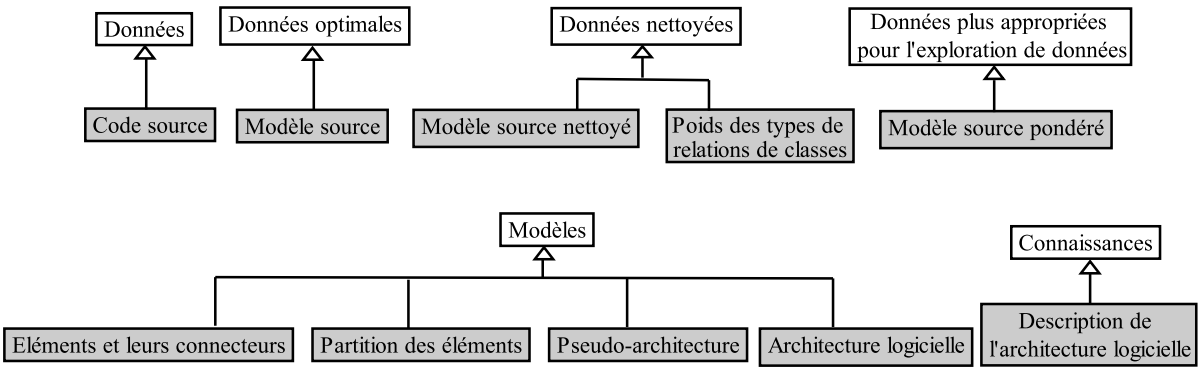

FIGURE 6.14 : Les relations entre les artefacts du modèle SAD et ceux du modèle ECD.

- Le code source
  Le code source, qui est l'artefact initial dans le modèle SAD contient toutes les instructions du système dont on veut extraire l'architecture. Ainsi, cet artefact correspond aux données dans le modèle ECD.

- Le modèle source
  Le modèle source dans le modèle SAD représente certaines informations du code source.

Précisément, il décrit les classes, leurs attributs et méthodes et leurs relations. Cet artefact est modifié lors de l'activité du nettoyage du modèle source ; par suite, avant cette activité, cet artefact correspond aux données optimales dans le modèle ECD et après le nettoyage, le modèle source correspond aux données nettoyées. Pour cela, nous distinguons à la figure le modèle source et le modèle source nettoyé.

- Les poids des types de relations de classes
  Les poids des types de relations de classes dans le modèle SAD sont des informations données par l'architecte. Ces informations ne contiennent aucun bruit ou des valeurs aberrantes. Cet artefact correspond aux données nettoyés dans le modèle ECD.

- Le modèle source pondéré
  Le modèle source pondéré dans le modèle SAD n'est autre qu'une représentation de la jointure entre le modèle source et les poids des types de relations de classes. Cette représentation est plus convenable à l'extraction d'une architecture logicielle. Cet artefact correspond aux données plus appropriées pour l'exploration de données dans le modèle ECD.

- Les éléments et leurs connecteurs pondérés
  Les éléments et leurs connecteurs pondérés dans le modèle SAD sont des nouvelles informations qui représentent les groupes de classes et leurs connecteurs. Cet artefact est un modèle dans le modèle ECD.

- La partition des éléments
  La partition des éléments dans le modèle SAD concerne des nouvelles informations qui représentent les éléments groupés selon certains critères. Cet artefact est aussi un modèle dans le modèle ECD.

- La pseudo-architecture
  De même, la pseudo-architecture dans le modèle SAD est des nouvelles informations qui représentent les différences interfaces des parties d'une partition. Cet artefact est un modèle dans le modèle ECD.

- L'architecture logicielle
  L'architecture logicielle dans le modèle SAD représente la structure du système. Elle est semblable à la pseudo-architecture mais, peut être, avec certaines modifications. Cet artefact est un modèle dans le modèle ECD.

- La description de l'architecture logicielle
  La description de l'architecture logicielle dans le modèle SAD décrit d'une manière compréhensible l'architecture du système. Plus précisément, cet artefact fournit les éléments architecturaux et leurs interfaces, qui sont des informations utiles, valides et compréhensibles pour un être humain. Par la suite, cet artefact est une connaissance dans le modèle ECD.

6.4.3 Les relations entre les rôles du modèle SAD et ceux du modèle ECD

En ce qui concerne les rôles, sept rôles sont définis dans le modèle de processus SAD. Pareil aux artefacts, les rôles ne sont pas exprimés dans le modèle ECD. Les rôles dérivés du modèle ECD sont responsables de l'exécution des étapes. Ces rôles sont alors soit un utilisateur, soit un outil. L'utilisateur est un être humain chargé de découvrir des nouvelles connaissances à partir des données ; c'est lui qui essaie de comprendre le domaine d'application du processus ECD

et les connaissances antérieures. Concernant les outils, ils ne sont pas définis directement dans le modèle ECD, de plus ils sont si nombreux. Il existe une pléthore de techniques, outils et algorithmes qui exécutent les étapes d'ECD. Afin de présenter les relations qui existent entre les rôles du modèle SAD et ceux du modèle ECD, nous fournissons directement les relations dans ce qui suit.

- L'architecte
  Ce rôle représente la personne chargée de l'extraction dans le modèle SAD. Il est l'utilisateur chargé d'exécuter les activités d'ECD dans le modèle ECD.

- L'analyseur du code source
  Dans le modèle SAD, ce rôle analyse le code source afin d'extraire certaines informations. Dans le modèle ECD, il existe plusieurs techniques qui permettent l'extraction des informations à partir de données afin de créer un ensemble de données cibles ; Par exemple, il existe des techniques permettant l'extraction de certaines informations à partir des fichiers XML. Cependant l'extraction des informations à partir d'un code source n'est pas supportée par une technique dans le modèle ECD. Ainsi, un analyseur du code source peut être considéré comme un extracteur des informations dans le modèle ECD, mais réellement, il n'existe pas un extracteur dans le modèle ECD permettant l'analyse d'un code source.

- La technique de jointure
  Dans le modèle SAD, cette technique fait joindre un modèle source avec les poids des relations pour former un modèle source pondéré. Dans le modèle ECD, les techniques de jointure existent. Ainsi, la technique de jointure dans le modèle SAD est une technique de jointure dans le modèle ECD.

- La technique de groupement
  Ce rôle exécute un groupement d'entités logicielles selon un type de relation de classes dans le modèle SAD. Dans le modèle ECD, il existe des techniques de groupement mais ne supportant pas le groupement selon le critère de type. Ainsi, la technique de groupement peut être considérée comme une technique de groupement dans le modèle ECD, mais réellement, le groupement selon un type de relation n'est pas supporté par le modèle ECD.

- La technique de partitionnement
  Ce rôle génère une partition des éléments d'une manière qui garantit le couplage faible entre les parties différentes de la partition et la cohésion forte à l'intérieur de chacune. Pareil aux deux techniques précédentes, dans le modèle ECD, il existe des techniques de groupement mais ne supportant pas un partitionnent sur les critères mentionnés. Ainsi, la technique de partitionnement peut être considérée comme une technique de groupement dans le modèle ECD, mais réellement, ce groupement n'est pas supporté par le modèle ECD.

6.5 L'extension d'un outil KDD pour l'extraction des architectures logicielles

Dans une image globale, un processus d'extraction SAD peut être exécuté en tant qu'un processus ECD. Mais, il existe un manque au niveau des rôles (techniques) empêchant l'exécution

complète d'un processus SAD en utilisant un outil ECD. Pour cela, afin de profiter des fonctionnalités classiques d'ECD qui permettent un utilisateur de découvrir itérativement et d'une manière interactive des nouvelles connaissances, nous avons élaboré une extension d'un outil KDD, nommé KNIME, pour l'extraction des architectures logicielles.

6.6 KNIME

Traditionnellement, la découverte de connaissances a été réalisée manuellement. Au fur et à mesure que le temps passe, la taille des données dans de nombreux systèmes devenait supérieure à la taille du téraoctet et ne pouvait plus être maintenue manuellement. En conséquence, plusieurs outils logiciels ont été développés pour découvrir des données et faire des hypothèses ; ces nouveaux outils sont appelés ECD. L'un des outils les plus connus dans ECD, la plate-forme KNIME [Berthold et al., 2009]. KNIME (Konstanz Information Miner) est une plate-forme ouverte développée pour l'exécution des étapes ECD. Les fonctionnalités sont disponibles en permettant à un utilisateur de construire et exécuter des *workflows* à l'aide des composants prédéfinis, appelés nœuds. Dans KNIME, un utilisateur crée un *workflow* en ajoutant des nœuds et en les configurant selon ses besoins. Par exemple, le *workflow* présenté à la figure 6.15 extrait des informations à partir d'un fichier XML décrivant des livres écrits par/ou à propos de Georges W. Bouche. Le *workflow* extrait toutes les informations telles que l'ISBN, l'auteur, le titre, etc. et donne comme nouvelle connaissance le nombre de livres pour chaque auteur. L'atout le plus important de KNIME est que des fonctionnalités supplémentaires pourraient être ajoutées. Chaque nouvelle fonctionnalité pourrait être écrite en Java à l'aide de la plate-forme Eclipse SDK et ajoutée comme extension à KNIME.

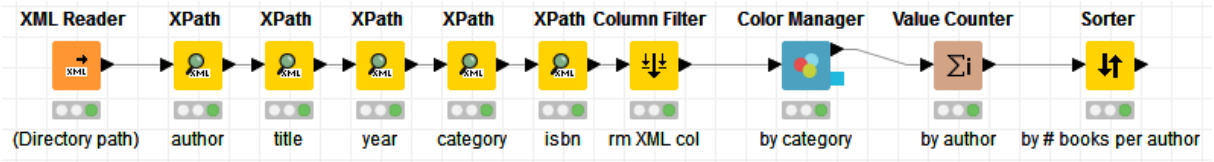

FIGURE 6.15 : Un exemple d'un *workflow* KNIME.

6.7 L'extension de KNIME

La plate-forme existante de KNIME permet à un utilisateur, à l'aide des nœuds, d'exécuter différentes étapes ECD telles que l'exploration de données, le nettoyage, l'interprétation et la visualisation les données... Mais ces nœuds existants ne sont pas suffisants pour exécuter une instance du modèle de processus SAD. Plus précisément, la plate-forme existante ne contient pas de nœuds qui exécutent les activités 1, 5, 6, 7 et 9 qui sont respectivement l'extraction d'un modèle source, le groupement des classes reliées, la classification des éléments, l'identification des interfaces fournies et requises, et la génération d'une description de l'architecture logicielle. Dans cette section, nous présentons une extension de KNIME composée de 5 nœuds. L'extension, avec les fonctionnalités existantes de KNIME, permettent à un architecte d'exécuter tout un processus d'extraction d'architecture logicielle tel qu'il apparaît dans le *workflow* de la figure 6.16. Un architecte peut ajouter d'autres nœuds au *workflow* afin de mieux comprendre l'architecture, comme l'ajout de nœuds de manipulation et de visualisation.

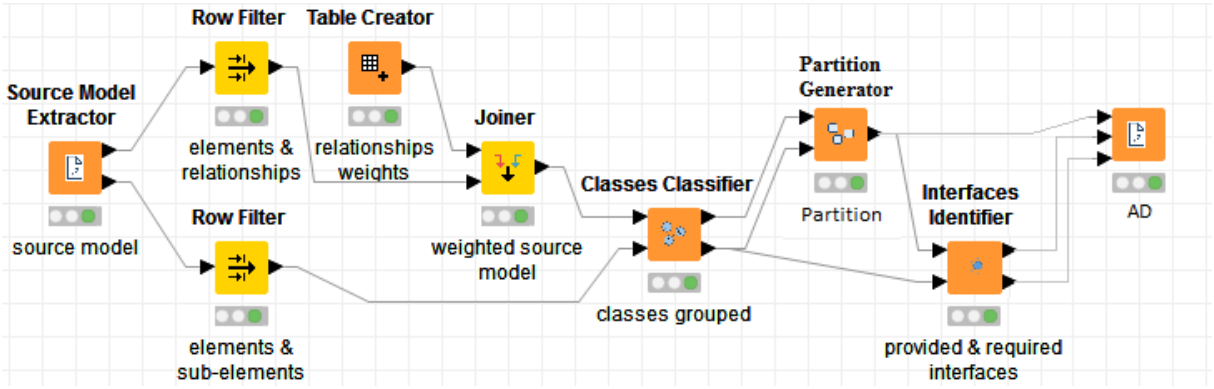

FIGURE 6.16 : Un *workflow* KNIME qui extrait une architecture de logicielle.

6.7.1 Le nœud SME pour l'extraction d'un modèle source

Le nœud SME *(Source Model Extractor)* exécute l'activité 1 décrite dans le paragraphe 5.5.1. Il extrait les classes, leurs méthodes et leurs relations (les relations de méthodes et celles de classes) et les décrit dans deux tables KNIME. Le nœud prend le chemin du code source en tant qu'input et donne un modèle source comme sortie représenté par deux tableaux comme suivant :

- Un tableau d'éléments et leurs relations qui a comme méta-données : *Élément 1*, *Élément 2*, *Type de relation*. *Élément 1* et *Élément 2* sont des classes ou des méthodes. Le *type de relation* est un appel de méthode ou une relation de classe (composition, agrégation, généralisation, association ou dépendance).
- Un tableau d'éléments et sous-éléments qui a comme méta-données : *Élément*, *Sous-élément*, *Granularité*. Un *Élément* est une classe, et un *Sous-élément* est une méthode. La *Granularité* est la granularité du sous-élément. Le numéro 1 est donné pour la granularité d'une méthode car c'est un élément à grain fin.

Les tableaux 6.1 et 6.2 montrent l'output de ce nœud pour le système de réservation.

6.7.2 Le nœud CG pour le groupement des classes reliées

Le nœud CG *(Classes Grouper)* exécute l'activité 5 décrite dans le paragraphe 5.5.5. Il regroupe les classes selon un type de relation choisi par l'architecte. Le nœud prend comme entrée une table d'éléments et leurs relations et un tableau d'éléments et de sous-éléments. Le nœud regroupe les classes en groupe de classes (qui sont des éléments à granularité supérieure) selon le type de relation choisi par l'architecte et donne les tables mises à jour en tant que sortie. Les tableaux 6.3 et 6.4 montrent l'output de ce nœud pour le système de réservation lorsque la relation de composition est choisie.

6.7.3 Le nœud PaG pour la classification des éléments

Le nœud PaG *(Partition Generator)* exécute l'activité 6 décrite dans le paragraphe 5.5.6. Il prend comme input un tableau d'éléments et leurs relations et un tableau d'éléments et de sous-éléments. Le nœud regroupe les groupe de classes (les éléments) en fonction de la fonction objective MQ 2.1. Plus spécifiquement, le nœud est configuré par l'utilisateur. Ce dernier saisit le nombre de parties du système, ce qui est comme une architecture conceptuelle. Puis, un

TABLE 6.1 : Les éléments et leurs relations générés par le nœud SME.

Élément 1	**Élément 2**	**Type de relation**
Package	Customer	composition
Flight reservation	Package	composition
Hotel reservation	Package	composition
Package	Rental reservation	composition
Reservation	Rental reservation	généralisation
Reservation	Hotel reservation	généralisation
Reservation	Flight reservation	généralisation
Rental	VehiculeRenatalCompany	composition
Rental	Vehicule	dépendance
vehicule	VehiculeRenatalCompany	dépendance
Room	Hotel	composition
Hotel	Stay	composition
Room	Stay	dépendance
Stay	Hotel reservation	dépendance
Rental	Rental reservation	dépendance
Flight	Flight reservation	dépendance
Airline	Flight	dépendance
Route	Flight	dépendance
Airline	Route	dépendance
Route	Airport	dépendance
Route	Airport	dépendance
Route	Airport	dépendance
Hotel\List<Room> :getFreeRooms()	Room\Boolean :roomIsAvailable()	appel de méthode
Room\int :GetRoomsCount()	Hotel\List<Room> :GetHotelRooms()	appel de méthode

TABLE 6.2 : Les éléments et leurs sous-éléments générés par le nœud MSE.

Élément	**Sous-élément**	**Granularité**
Reservation	Double :getPrice()	1
Reservation	Int :getNbOfDay()	1
Package	createPackage()	1
Customer	String :getName()	1
Customer	Int :getAge()	1
Flight	String :getName()	1
Flight	void :createFlight()	1
Flight reservation	Void :setNbOfDay()	1
Airline	String :getName()	1
Airline	String :getPlace()	1
Airline	getCountries()	1
Airport	String :getName()	1
Airport	getCountry()	1
Airport	Date :getOpenHour()	1
Airport	Date :getCloseHour()	1
Route	String :getName()	1
Route	List<String> :getPrimariesRoute()	1
Route	List<String> :getSecondariesRoute()	1
Stay	getCheckIn()	1
Stay	getCheckOut()	1
Room	Int :getRoomCount()	1
Room	Boolean :roomIsAvailable()	1
Room	String :getName()	1
Room	Int :getNbRoom()	1
Rental	Void :createRental()	1
Rental	Date :getStartDate()	1
Rental	Date :getEndDate()	1
RentalVehiculeCompany	String :getName()	1
Vehicule	String :getName()	1
Vehicule	String :getMake()	1
Vehicule	String :getModel()	1
Hotel reservation	Void :SetNbOfDay()	1
Hotel reservation	Void :SetDay()	1
Hotel reservation	Int :GetNbOfDay()	1
Hotel reservation	Void :setContry()	1
Rental reservation	Void :SetNbOfDay()	1
Rental reservation	Int :GetNbOfDay()	1
Rental reservation	Void :SetPrefferedModel()	1
Rental reservation	Void :SetDay()	1
Hotel	String :getName()	1
Hotel	List<Room> :GetHotelRooms()	1
Hotel	List<Room> :getFreeRooms()	1
Hotel	String :GetAddress()	1
Hotel	Hotel : Reserve()	1
...	...	...

TABLE 6.3 : Les éléments et leurs relations générés par le nœud CG.

Élément 1	**Élément 2**	**Type de la relation**	**Poids de la relation**
Package	Customer	composition	4
Flight reservation	Package	composition	4
Hotel reservation	Package	composition	4
...	...	...	...
E1	E2	connecteur pondéré	1
E1	E3	connecteur pondéré	1
E1	E4	connecteur pondéré	1
E1	E9	connecteur pondéré	9
E4	E8	connecteur pondéré	2
E2	E5	connecteur pondéré	1
E2	E6	connecteur pondéré	1
E5	E6	connecteur pondéré	1
E6	E7	connecteur pondéré	3

TABLE 6.4 : Les éléments et leurs sous-éléments générés par le nœud CG.

Élément	**Sous-élément**	**Granularité**
Reservation	Double :getPrice()	1
Reservation	Int :getNbOfDay()	1
Package	createPackage()	1
...	...	...
E1	Package	2
E1	Customer	2
E1	Flight reservation	2
E1	Hotel reservation	2
E1	Rental reservation	2
E2	Flight	2
E3	Room	2
E3	Hotel	2
E3	Stay	2
E4	Rental	2
E4	VehiculeRentalCompany	2
E5	Airline	2
E6	Route	2
E7	Airport	2
E8	Vehicule	2
E9	Reservation	2

TABLE 6.5 : Les éléments et leurs sous-éléments générés par le nœud PaG.

Élément	**Sous-élément**	**Granularité**
Reservation	Double :getPrice()	1
Reservation	Int :getNbOfDay()	1
Package	createPackage()	1
...	...	...
E1	Package	2
E1	Customer	2
...	...	...
E9	Reservation	2
E10	E2	3
Part1	E5	3
Part1	E6	3
Part1	E7	3
Part2	E1	3
Part2	E3	3
Part2	E9	3
Part3	E4	3
Part3	E8	3

algorithme génétique est exécuté en utilisant la fonction MQ. Cet algorithme commence par une partition initiale composée de nombre de parties donné par l'architecte, puis cherche une meilleure partition. La sortie est le tableau des éléments et des sous-éléments mis à jour en ajoutant les parties. Table 6.5 montre la sortie de cette activité pour le système de réservation.

6.7.4 Le nœud II

Le nœud II *(Interfaces Identifier)* exécute l'activité 7 décrite dans le paragraphe 5.5.7. Ce nœud identifie les interfaces de chaque partie d'une partition selon les relations de méthodes. Ainsi, il fournit les interfaces représentées par deux tableaux comme suit :

- Un tableau des interfaces fournies qui a comme méta-données : *Élément*, *Sous-élément*, *Classe* et *Méthode fournie*. L'élément est la partie à laquelle appartient l'interface. La méthode fournie est la méthode appelée à partir d'autres parties de la partition. La classe est la classe contenant la méthode appelée. Le sous-élément est l'élément (groupe de classes) qui contient la classe.

- Un tableau des interfaces requises qui a comme méta-données : *Élément*, *Sous-élément*, *Classe* et *Méthode appelante*, *Méthode requise*. L'élément est la partie à laquelle appartient l'interface. La méthode appelante est la méthode appartenant à la partie et qui appelle une méthode d'une autre partie. La méthode requise est une méthode nécessaire pour la partie pour qu'elle fonctionne, donc c'est une interface fournie par une autre partie. La classe est la classe contenant la méthode appelante. Le sous-élément est l'élément (groupe de classes) qui contient la classe.

Les tableaux 6.6 et 6.7 montrent l'output de ce nœud pour le système de réservation.

TABLE 6.6 : Les interfaces fournies.

Élément	Sous-élément	Classe	Méthode fournie
Part1	E2	Flight	createFlight()

TABLE 6.7 : Les interfaces requises.

Élément	Sous-élément	Classe	Méthode appelante	Méthode requise
Part2	E1	Flight reservation	reserveFlight ()	createFlight()
Part2	E1	Rental reservation	reserveRental ()	createRental()

6.7.5 Le nœud ADG

Le nœud ADG *(Architecture Description Generator)* exécute l'activité 9 décrite dans le paragraphe 5.5.9. Ce nœud génère une description textuelle des tableaux. Plus spécifiquement, il détaille quelles sont parties qui constituent le système et leurs interfaces.

6.8 La synthèse

Dans cette section, nous projetons l'extension de KNIME sur le facteur d'outils du modèle de comparaison. La table 6.8 résume cette projection. Tout d'abord l'outil permet à l'architecte d'intégrer ses connaissances en lui permettant de proposer une architecture conceptuelle. Cette dernière est saisie sous forme de nombre de parties pour la partition ; c.à.d en utilisant le nœud PaG décrit dans le paragraphe 6.7.3. D'autres informations sont encore saisies par l'architecte, par exemple les poids des relations. De plus, vu que l'outil est une extension d'ECD, il fournit à l'architecte des fonctionnalités classiques d'ECD. Ainsi un architecte peut configurer les nœuds comme choisir le type de groupement des classes dans le nœud CG (voir 6.7.2) et peut également profiter de ces fonctionnalités pour découvrir et comprendre les artefacts manipulés et raffiner par la suite les résultats des nœuds tout au long du processus. Ce qui est le plus important est que l'outil supporte efficacement les deux stratégies : le groupement des entités logicielles et la correspondance des entités à une architecture conceptuelle et ceci en utilisant le nœud PaG. Concernant les autres critères, pour le moment, l'extension est limitée à l'extraction de la structure statique, la representation des informations sous formes des tableaux et l'extraction à partir du code source implémentés en Java.

6.9 Conclusion

Dans ce chapitre, nous avons étudié les relations qui existent entre le modèle d'extraction de connaissances à partir des données (ECD) et notre modèle de processus (SAD) proposé dans le chapitre précédent. Cette étude a souligné que le modèle SAD a un aspect semblable à celui d'ECD. Alors qu'ECD se focalise sur le principe de découverte de nouvelles connaissances à partir de différentes données, SAD se focalise sur le principe de découverte d'une representation architecturale. Bien que les deux modèles ont des points communs, les rôles qui exécutent les activités d'ECD ne supportent pas l'exécution de certaines activités du modèle SAD. Ainsi, nous avons fourni une extension d'un outil ECD, dénoté KNIME, pour supporter l'exécution des processus d'extraction.

TABLE 6.8 : La projection de l'extension de KNIME sur le modèle de comparaison.

Facteurs	**Critères**	**Métriques**	Q
Outil	Intégration des connaissances de l'architecte	Supporte l'intégration des connaissances de l'architecte	B
	Interaction de l'architecte	Supporte l'interaction de l'architecte	B
	Structure extraite	Structure statique	MY
	Représentation des informations	Pas selon un modèle	MY
	Langages de programmation supportés	Un seul langage de programmation	MV
	Types de programmation supportés	type orienté objet	MY
	Groupement des entités logicielles	Supporte plusieurs groupement des entités logicielles	B
	Correspondance des entités logicielles à l'architecture conceptuelle	Supporte la correspondance des entités à une architecture logicielle	B

Conclusion et Perspectives

Les principales contributions de cette thèse relèvent du domaine d'extraction d'une architecture logicielle. En particulier, nous avons proposé un méta-modèle qui spécifie les concepts d'extraction d'une architecture logicielle, une méthode d'extraction et un outil supportant la méthode d'extraction. Dans ce qui suit, nous soulignons les résultats et les perspectives.

Les contributions

Dans ce qui suit nous résumons les principales contributions de notre travail :

1. Nous avons élaboré un modèle d'évaluation de qualité des processus d'extraction d'architectures logicielles. Ce modèle est basé sur le modèle McCall proposant des facteurs, critères et métriques propres à l'extraction d'architectures logicielles. Le modèle de qualité nous a servi pour l'évaluation de la plupart des approches connues dans ce domaine.

2. Nous avons proposé un méta-modèle dénoté SArEM *(Software Architecture Extraction Meta-model)* qui détaille les concepts des processus d'extraction d'une architecture logicielle. Le méta-modèle répond à la problématique d'absence d'un outil conceptuel facilitant l'analyse, l'évaluation et la comparaison des processus d'extraction existants. Plus précisément, SArEM hérite les concepts du méta-modèle SPEM et les étend par des nouveaux concepts dédiés à l'extraction. Ces concepts sont le résultat d'une étude des concepts présents dans les processus d'extraction les plus représentatifs et cités dans la littérature. Le méta-modèle a été validé en présentant une étude de conformité entre ses concepts et les concepts des approches existantes.

3. Nous avons présenté un modèle de processus d'extraction dénoté SAD *(Software Architecture Discovery)*. Ce modèle permet à un architecte d'intégrer ses connaissances durant le processus et d'interagir avec les artefacts du processus ; il permet ainsi à l'architecte de construire le processus et de le configurer selon son point de vue. Le modèle de processus est basé sur une stratégie de correspondance entre des groupes d'entités et une architecture conceptuelle proposée par l'architecte. Ainsi, il suit l'une des stratégies d'extraction les plus abouties à savoir la conciliation entre une architecture extraite et une architecture conceptuelle.

4. Nous avons également fourni une extension d'un outil ECD qui supporte notre modèle d'extraction. Cette extension permet à un architecte d'exécuter un processus SAD comme étant un processus d'ECD. Ainsi, un architecte peut profiter des fonctionnalités classiques de l'outil ECD, nommé KNIME, et utiliser l'extension pour les activités spécifiques à l'extraction. En fait la valeur ajoutée dans cette contribution est l'étude des points communs qui existent entre les deux domaines : L'extraction d'une architecture logicielle et la découverte des nouvelles connaissances à partir des données. A notre avis, cette étude

ouvre une nouvelle piste dans le domaine d'extraction ; elle fournit au domaine d'extraction d'une architecture logicielle des nouvelles techniques et des nouveaux outils aidant à avancer dans le domaine d'extraction.

Les limites

Bien que nos contributions apportent une solution originale au domaine d'extraction d'architectures logicielles, il demeure qu'elles souffrent de quelques limitations principales :

Premièrement, bien que notre méta-modèle SArEM détaille les activités, les artefacts et les rôles des processus d'extraction, il serait intéressant de détailler également les niveaux d'abstraction sur lesquels opèrent les processus d'extraction. De plus, SArEM pourrait être enrichi en prenant en compte le concept des styles architecturaux [Ding et Medvidovic, 2001, Medvidovic et al., 2003]. Ainsi, notre méta-modèle doit intégrer la notion des styles architecturaux comme un concept essentiel lié au principe d'extraction d'une architecture logicielle.

Deuxièmement, le modèle de processus SAD doit prendre en considération d'autres informations comme le flux de données durant l'exécution. En d'autres termes, notre modèle considère la vue logique et dynamique du système ; ces vues représentent uniquement les classes, leurs relations et les appels entre les méthodes et n'incluent pas d'autres informations comme la distribution des fichiers, les invocations des méthodes durant l'exécution.

Troisièmement, à propos de l'outil d'extraction, notre but est de fournir un outil générique qui permet à l'architecte de construire son propre processus. Bien que l'outil aide l'architecte à achever ce but en lui permettant de configurer certains paramètres, l'outil reste limité par les mesures proposées. En effet, nous souhaitons fournir une manière plus générique permettant à un architecte de définir ou choisir des mesures liés spécifiquement à l'extraction. Par exemple, au lieu de limiter le groupement des entités à la mesure d'addition des poids des relations de classes et la fonction MQ, nous aimerions permettre à l'architecte de définir ses propres mesures.

Les perspectives

Nous pouvons évoquer différentes pistes pour poursuivre le travail commencé. Tout d'abord, au niveau du méta-modèle d'extraction, SArEM doit être modifié d'une manière qui respecte le modèle MOF *(Meta Object Facility)* [OMG, 2013] défini par le groupe OMG. En catégorisant ses entités selon les niveaux de MOF, SArEM aura l'avantage d'être considéré comme un cadre formel pour la spécification des processus d'extraction. En plus, vu qu'une architecture logicielle est liée aux notions des styles architecturaux et vues architecturales, il sera avantageux de prendre en considération ces notions dans un méta-modèle dédié à l'extraction d'une architecture logicielle. Comme perspective à long terme, nous proposons de poursuivre l'étude sur la modélisation des processus d'extraction. Comme il existe des travaux sur la modélisation des processus d'ingénierie des systèmes logiciels [Hug, 2009] et des outils spécifiques à la modélisation de ces outils, nous suggérons l'élaboration des méthodes pour la modélisation de ces processus et des outils supportant ces méthodes également.

Ensuite, concernant le modèle SAD, vu que les systèmes récents sont implémentés en termes de composants, le modèle SAD doit être étendu pour supporter ces types d'entités. En effet, une extension du modèle peut aboutir à réorganiser un système à base de composants vers une meilleure architecture, ayant un plus faible couplage entre les composants et une plus forte cohésion. De même, nous pourrons aborder l'extension du modèle pour fournir une architecture du système à base de services. En fait, pour le moment, le modèle considère les interfaces

comme étant des méthodes appelées entre les parties, cependant, une architecture doit fournir les interfaces en termes de services, qui sont plus significatifs que l'appel des méthodes.

Finalement, à propos de l'outil, comme nous l'avons déjà mentionné dans la section « limite », notre but est de permettre à un architecte d'exécuter son propre processus. En effet, une élaboration d'un outil générique pour l'extraction peut aider aussi les chercheurs dans le domaine d'extraction à définir plus facilement des fonctions objectives et des mesures concernant l'extraction. Ainsi, nous suggérons l'extension de l'outil pour qu'il soit une plateforme aidant à proposer des mesures et fonctions liées à l'extraction.

Bibliographie

[Abboud et al., 2016a] Abboud, M., Naja, H., Oussalah, M., et Dbouk, M. (2016a). SArEM : A SPEM extension for software architecture extraction process. *International Journal on Computer Science and Engineering*, 8(4), 152–159. 73, 96

[Abboud et al., 2016b] Abboud, M., Naja, H., Oussalah, M., et Dbouk, M. (2016b). SArEM : Un méta-modèle pour la spécification des processus d'extraction d'architectures logicielles. In *16ème Journées Francophones Extraction et Gestion des Connaissances, {EGC} 2016, 18-22 Janvier 2016, Reims, France* (pp. 521–522). 73, 96

[Abboud et al., 2017a] Abboud, M., Naja, H., Oussalah, M., et Dbouk, M. (2017a). Kdd extension tool for software architecture extraction. In *International Conference on Software Engineering Research and Practice (SERP'17), July 17-20, 2017, Las Vegas, Nevada, USA* (pp. 120–126). 103

[Abboud et al., 2017b] Abboud, M., Naja, H., Oussalah, M., et Dbouk, M. (2017b). Towards using kdd for an interactive software architecture extraction. In *18th IEEE International Conference on Information Reuse and Integration (IEEE IRI 2017), Aug 4-6, 2017, San Diego, CA, USA*. 103

[Albin-Amiot et Guéhéneuc, 2001] Albin-Amiot, H. et Guéhéneuc, Y.-G. (2001). Meta-modeling design patterns : Application to pattern detection and code synthesis. In *Proceedings of ECOOP Workshop on Automating Object-Oriented Software Development Methods*. 44

[Anand et Büchner, 1998] Anand, S. S. et Büchner, A. G. (1998). *Decision support using data mining*. Financial Times Management. 128

[Bachmann et al., 2011] Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., et Stafford, J. (2011). *Documenting Software Architectures : Views and Beyond*. Addison-Wesley Professional. 26, 28

[Berthold et al., 2009] Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., et Wiswedel, B. (2009). KNIME-the konstanz information miner : version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1), 26–31. 21, 140

[Beszédes et al., 2005] Beszédes, A., Ferenc, R., et Gyimóthy, T. (2005). Columbus : A reverse engineering approach. In *Proc. 13th Workshop Software Technology and Eng. Practice* (pp. 93–96). 44, 68

[Bowman et al., 1999] Bowman, I. T., Holt, R. C., et Brewster, N. V. (1999). Linux as a case study : Its extracted software architecture. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99 (pp. 555–563). : ACM. 13, 19, 37, 38, 62, 88, 91, 94, 95

[Cabena et al., 1998] Cabena, P., Hadjinian, P., Stadler, R., Verhees, J., et Zanasi, A. (1998). *Discovering data mining : from concept to implementation*. Prentice-Hall, Inc. 128

[Chardigny, 2009] Chardigny, S. (2009). Extraction d'une architecture logicielle à base de composants depuis un système orienté objet. une approche par exploration. 33, 45, 56, 69

[Chen, 1995] Chen, Y.-F. (1995). *Reverse engineering, Practical reusable UNIX software*. John Wiley & Sons, Inc., New York, NY. 45

[Chen et al., 1998] Chen, Y.-F., Gansner, E. R., et Koutsofios, E. (1998). A c++ data model supporting reachability analysis and dead code detection. *Software Engineering, IEEE Transactions on*, 24(9), 682–694. 45

[Chen et al., 1997] Chen, Y.-F. R., Gansner, E. R., et Koutsofios, E. (1997). A c++ data model supporting reachability analysis and dead code detection. In *Software Engineering—ESEC/FSE'97* (pp. 414–431). Springer. 36

[Chikofsky et Cross, 1990] Chikofsky, E. J. et Cross, J. H. (1990). Reverse engineering and design recovery : A taxonomy. *IEEE software*, 7(1), 13–17. 30

[Cios et Kurgan, 2005] Cios, K. J. et Kurgan, L. A. (2005). Trends in data mining and knowledge discovery. *Advanced techniques in knowledge discovery and data mining*, (pp. 1–26). 128

[Commission et others, 2011] Commission, I. E. et others (2011). *Systems and software engineering : architecture description*. ISO. 17

[Company et al., 1977] Company, G. E., McCall, J. A., Richards, P. K., et Walters, G. F. (1977). *Factors in software quality*. Information Systems Programs, General Electric Company. 49

[Councill et Heineman, 2001] Councill, B. et Heineman, G. T. (2001). Definition of a software component and its elements. *Component-based software engineering : putting the pieces together*, (pp. 5–19). 29

[De Silva et Balasubramaniam, 2012] De Silva, L. et Balasubramaniam, D. (2012). Controlling software architecture erosion : A survey. *Journal of Systems and Software*, 85(1), 132–151. 34, 58

[Dijkstra, 1972] Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, 15(10), 859–866. 25

[Ding et Medvidovic, 2001] Ding, L. et Medvidovic, N. (2001). Focus : A light-weight, incremental approach to software architecture recovery and evolution. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on* (pp. 191–200). : IEEE. 13, 19, 41, 44, 66, 91, 94, 95, 96, 150

[Doval et al., 1999] Doval, D., Mancoridis, S., et Mitchell, B. S. (1999). Automatic clustering of software systems using a genetic algorithm. In *Software Technology and Engineering Practice, 1999. STEP'99. Proceedings* (pp. 73–81). : IEEE. 35, 60

[Ducasse et Pollet, 2009] Ducasse, S. et Pollet, D. (2009). Software architecture reconstruction : A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4), 573–591. 34, 58, 69

[Fayyad et al., 1996a] Fayyad, U., Piatetsky-Shapiro, G., et Smyth, P. (1996a). From data mining to knowledge discovery in databases. *AI magazine*, 17(3), 37. 14, 20, 104, 127, 128, 129

[Fayyad et al., 1996b] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., et Uthurusamy, R. (1996b). Advances in knowledge discovery and data mining. *AAAI press Menlo Park*. 20, 22, 104

[Ferenc et al., 2002] Ferenc, R., Beszédes, r., Tarkiainen, M., et Gyimóthy, T. (2002). Columbus-reverse engineering tool and schema for c++. In *Software Maintenance, 2002. Proceedings. International Conference on* (pp. 172–181). : IEEE. 44, 68

[Ferenc et Beszedes, 2002] Ferenc, R. et Beszedes, A. (2002). : (pp. 59–66). : IEEE Comput. Soc. 44

[Ferenc et al., 2004] Ferenc, R., Siket, I., et Gyimóthy, T. (2004). Extracting facts from open source software. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on* (pp. 60–69). : IEEE. 44, 68

[Gamma, 1995] Gamma, E. (1995). *Design patterns : elements of reusable object-oriented software*. Pearson Education India. 38

[Gargiulo et Mancoridis, 2001] Gargiulo, J. et Mancoridis, S. (2001). Gadget : A tool for extracting the dynamic structure of java programs. In *SEKE*, volume 1 (pp. 244–251). 45, 68

[Garlan et Perry, 1995] Garlan, D. et Perry, D. E. (1995). Introduction to the special issue on software architecture. *IEEE Trans. Software Eng.*, 21(4), 269–274. 26

[Garlan et Shaw, 1994] Garlan, D. et Shaw, M. (1994). An introduction to software architecture. 25, 26

[Gorton, 2011] Gorton, I. (2011). *Documenting a Software Architecture*, (pp. 117–128). Springer Berlin Heidelberg. 17

[Guéhéneuc, 2004] Guéhéneuc, Y.-G. (2004). A reverse engineering tool for precise class diagrams. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research* (pp. 28–41). : IBM Press. 44, 68

[Guo et al., 1999] Guo, G. Y., Atlee, J. M., et Kazman, R. (1999). *A software architecture reconstruction method*. Springer. 19, 38, 63, 87, 91, 94, 95

[Han et Kamber, 2006] Han, J. et Kamber, M. (2006). Classification and prediction. *Data mining : Concepts and techniques*, (pp. 347–350). 127

[Hilliard, 2000] Hilliard, R. (2000). Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems. *IEEE, http ://standards. ieee. org*, 12, 16–20. 26, 27

[Holt et al., 2000a] Holt, R., Winter, A., et Schurr, A. (2000a). : (pp. 162–171). : IEEE Comput. Soc. 44

[Holt et al., 2000b] Holt, R. C., Hassan, A. E., Laguë, B., Lapierre, S., et Leduc, C. (2000b). E/r schema for the datrix c/c++/java exchange format. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, WCRE '00 (pp. 284–). : IEEE Computer Society. 36

[Hug, 2009] Hug, C. (2009). *Méthode, modèles et outil pour la méta-modélisation des processus d'ingénierie de systèmes d'information*. PhD thesis, Université Joseph-Fourier-Grenoble I. 150

[ISO, 2011] ISO, M. (2011). *Systems and software engineering–architecture description*. Technical report, ISO/IEC/IEEE 42010. 13, 28, 29

[Jazayeri et al., 2000] Jazayeri, M., Ran, A., et Van Der Linden, F. (2000). *Software architecture for product families : principles and practice*. Addison-Wesley Longman Publishing Co., Inc. 30

[Kazman et Carrière, 1999] Kazman, R. et Carrière, S. J. (1999). Playing detective : Reconstructing software architecture from available evidence. *Automated Software Engineering*, 6(2), 107–138. 38, 40, 47, 55, 91, 94, 95

[Kazman et al., 1998] Kazman, R., Woods, S., et Jeromy Carriere, S. (1998). Requirements for integrating software architecture and reengineering models : CORUM II. In *Fifth Working Conference on Reverse Engineering, 1998. Proceedings* (pp. 154–163). 13, 31, 32, 55

[Korn et al., 1999] Korn, J., Chen, Y.-F., et Koutsofios, E. (1999). Chava : Reverse engineering and tracking of java applets. In *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on* (pp. 314–325). : IEEE. 45

[Koschke, 2000] Koschke, R. (2000). Atomic architectural component recovery for program understanding and evolution. 32, 56

[Koschke et Simon, 2003] Koschke, R. et Simon, D. (2003). : (pp.3̃6). : IEEE. 19, 40

[Krikhaar, 1997] Krikhaar, R. (1997). : (pp. 4–11). : , International Conference on Software Maintenance, 1997. Proceedings. 30

[Kumari et Srinivas, 2013] Kumari, A. C. et Srinivas, K. (2013). Software module clustering using a fast multi-objective hyper-heuristic evolutionary algorithm. *International Journal of Applied Information Systems*, 5(6), 12–18. 19

[Kumari et Srinivas, 2016] Kumari, A. C. et Srinivas, K. (2016). Hyper-heuristic approach for multi-objective software module clustering. *Journal of Systems and Software*, 117, 384–401. 19

[Kurgan et Musilek, 2006] Kurgan, L. A. et Musilek, P. (2006). A survey of knowledge discovery and data mining process models. *The Knowledge Engineering Review*, 21(1), 1–24. 128

[Len et al., 2003] Len, B., Paul, C., et Rick, K. (2003). Software architecture in practice. *Boston, Massachusetts Addison*. 17, 26, 27, 53

[Lüer et Van Der Hoek, 2002] Lüer, C. et Van Der Hoek, A. (2002). *Composition environments for deployable software components*. Citeseer. 29

[Mahdavi et al., 2003a] Mahdavi, K., Harman, M., et Hierons, R. (2003a). Finding building blocks for software clustering. In *Genetic and Evolutionary Computation (GECCO)* (pp. 2513–2514). : Springer Berlin Heidelberg. 19, 36, 60, 70, 94, 95

[Mahdavi et al., 2003b] Mahdavi, K., Harman, M., et Hierons, R. M. (2003b). A multiple hill climbing approach to software module clustering. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on* (pp. 315–324). : IEEE. 19, 60

[Mancoridis et al., 1994] Mancoridis, S., Holt, R. C., et Godfrey, M. W. (1994). : (pp. 60–65). : ACM Press. 36

[Mancoridis et al., 1999] Mancoridis, S., Mitchell, B. S., Chen, Y., et Gansner, E. R. (1999). Bunch : A clustering tool for the recovery and maintenance of software system structures. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on* (pp. 50–59). : IEEE. 35, 36, 45, 60

[Mancoridis et al., 1998] Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y.-F., et Gansner, E. R. (1998). Using automatic clustering to produce high-level system organizations of source code. In *IWPC*, volume 98 (pp. 45–52). : Citeseer. 19, 35, 46, 60, 94, 95

[Maqbool et Babri, 2004] Maqbool, O. et Babri, H. A. (2004). The weighted combined algorithm : A linkage algorithm for software clustering. In *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on* (pp. 15–24). : IEEE. 19

[Maqbool et Babri, 2007] Maqbool, O. et Babri, H. A. (2007). Hierarchical clustering for software architecture recovery. In *Software Engineering, IEEE Transactions on*, volume 33 (pp. 759–780). 19

[Medvidovic et al., 2003] Medvidovic, N., Egyed, A., et Gruenbacher, P. (2003). Stemming architectural erosion by coupling architectural discovery and recovery. In *STRAW*, volume 3 (pp. 61–68). 19, 42, 44, 67, 91, 94, 95, 96, 150

[Mitchell et al., 2001] Mitchell, B., Traverso, M., et Mancoridis, S. (2001). An architecture for distributing the computation of software clustering algorithms. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on* (pp. 181–190). : IEEE. 35, 60

[Mitchell et Mancoridis, 2006] Mitchell, B. S. et Mancoridis, S. (2006). On the automatic modularization of software systems using the bunch tool. In *Software Engineering, IEEE Transactions on*, volume 32 (pp. 193–208). 19, 35, 45, 60

[Mitchell et Mancoridis, 2008] Mitchell, B. S. et Mancoridis, S. (2008). On the evaluation of the bunch search-based software modularization algorithm. *Soft Computing*, 12(1), 77–93. 19, 35, 60

[Müller et Klashinsky, 1988] Müller, H. A. et Klashinsky, K. (1988). Rigi-a system for programming-in-the-large. In *Proceedings of the 10th international conference on Software engineering* (pp. 80–86). : IEEE Computer Society Press. 39, 44

[Muller et al., 2012] Muller, H. A., Norman, R. J., et Slonim, J. (2012). *Computer Aided Software Engineering*. Springer Science & Business Media. 44

[Murphy et Notkin, 1997] Murphy, G. C. et Notkin, D. (1997). Reengineering with reflexion models : A case study. *Computer*, 30(8), 29–36. 39, 46, 91

[Murphy et al., 1995] Murphy, G. C., Notkin, D., et Sullivan, K. (1995). Software reflexion models : Bridging the gap between source and high-level models. *ACM SIGSOFT Software Engineering Notes*, 20(4), 18–28. 39, 46

[Murphy et al., 2001] Murphy, G. C., Notkin, D., et Sullivan, K. J. (2001). Software reflexion models : Bridging the gap between design and implementation. *Software Engineering, IEEE Transactions on*, 27(4), 364–380. 19, 39, 46, 65, 94, 95

[OMG, 2013] OMG (2013). OMG meta object facility (MOF) core specification. In *OMG's industry-standard environment*. 150

[OMG et Notation, 2008] OMG, S. et Notation, O. (2008). Software & systems process engineering meta-model specification. *OMG Std., Rev*, 2. 20, 73, 74, 96

[Perry et Wolf, 1992] Perry, D. E. et Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4), 40–52. 25, 26

[Pollet et al., 2007] Pollet, D., Ducasse, S., Poyet, L., Alloui, I., Cimpan, S., et Verjus, H. (2007). Towards a process-oriented software architecture reconstruction taxonomy. In *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on* (pp. 137–148). : IEEE. 34, 58

[Praditwong et al., 2011] Praditwong, K., Harman, M., et Yao, X. (2011). Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, 37(2), 264–282. 19, 45

[Qiu et al., 2015] Qiu, D., Zhang, Q., et Fang, S. (2015). Reconstructing software high-level architecture by clustering weighted directed class graph. *International Journal of Software Engineering and Knowledge Engineering*, 25(04), 701–726. 19, 45

[Quatrani, 2002] Quatrani, T. (2002). *Visual modeling with rational rose 2002 and UML*. Addison-Wesley Longman Publishing Co., Inc. 44

[Rajalakshmi, M, 2014] Rajalakshmi, M (2014). Software system re-modularization using interactive genetic algorithm. 3(4), 105–107. 19, 37, 61, 70, 94, 96

[Riva, 2000] Riva, C. (2000). Reverse architecting : an industrial experience report. In *wcre* (pp.42̃). : IEEE. 19, 30, 33, 40, 56, 91, 94, 95

[Royce et Royce, 1991] Royce, W. et Royce, W. (1991). Software architecture : Integrating process and technology. *TRW Quest*, 14(1), 2–15. 25

[Rozanski et Woods, 2012] Rozanski, N. et Woods, E. (2012). *Software Systems Architecture : Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley. 27

[Sartipi et Kontogiannis, 2001] Sartipi, K. et Kontogiannis, K. (2001). A graph pattern matching approach to software architecture recovery. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)* (pp. 408). : IEEE Computer Society. 19

[Sartipi et Kontogiannis, 2003] Sartipi, K. et Kontogiannis, K. (2003). On modeling software architecture recovery as graph matching. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on* (pp. 224–234). : IEEE. 19

[Shearer, 2000] Shearer, C. (2000). The crisp-dm model : the new blueprint for data mining. *Journal of data warehousing*, 5(4), 13–22. 128

[Specification, 2007] Specification, O. A. (2007). Omg unified modeling language (omg uml), superstructure, v2. 1.2. *Object Management Group*. 30

[Szyperski, 2002] Szyperski, C. (2002). *Component software : beyond object-oriented programming*. Pearson Education. 29

[Tichelaar et al., 2000] Tichelaar, S., Ducasse, S., et Demeyer, S. (2000). FAMIX and XMI. In *Seventh Working Conference on Reverse Engineering, 2000. Proceedings* (pp. 296–298). 44

[Tilley et al., 1994] Tilley, S. R., Wong, K., Storey, M.-A. D., et Müller, H. A. (1994). Programmable reverse engineering. *International Journal of Software Engineering and Knowledge Engineering*, 4(4), 501–520. 47

[Wirth et Hipp, 2000] Wirth, R. et Hipp, J. (2000). Crisp-dm : Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining* (pp. 29–39). 128

[Woods et al., 1998] Woods, S., O'Brien, L., Lin, T., Gallagher, K., et Quilici, A. (1998). An architecture for interoperable program understanding tools. In *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on* (pp. 54–63). : IEEE. 31

Les publications de l'auteur

[1] Mira Abboud, Hala Naja, Mourad Oussalah and Mohamad Dbouk. SArEM : A SPEM extension for software architecture extraction process. International Journal on Computer Science and Engineering (IJCSE). Vol. 8, No.4, 2016. ISSN : 2229-5631.

[2] Mira Abboud, Hala Naja, Mourad Oussalah and Mohamad Dbouk. SArEM : Un méta-modèle pour la spécification des processus d'extraction d'architectures logicielles. In 16ème Journées Francophones Extraction et Gestion des Connaissances, (EGC), 2016, 18-22 Janvier 2016, Reims, France, pages 521-522.

[3] Mira Abboud, Hala Naja, Mourad Oussalah and Mohamad Dbouk. KDD extension tool for software architecture extraction. In the International Conference on Software Engineering Research and Practice (SERP'17), Las Vegas, Nevada, USA, (July 17-20, 2017).

[4] Mira Abboud, Hala Naja, Mourad Oussalah and Mohamad Dbouk. Towards using KDD for an interactive software architecture extraction. In the 18th IEEE International Conference on Information Reuse and Integration (IEEE IRI 2017), San Diego, CA, USA, (Aug 4-6, 2017).

[5] Mira Abboud, Hala Naja, Mourad Oussalah, Mohamed Dbouk. Adapting KNIME, a KDD tool, for software architecture extraction. LAAS 17, April 26-27, 2017, UL-Fanar, Lebanon.

[6] Mira Abboud, Hala Naja, Mourad Oussalah, Mohamed Dbouk. SArEM : Un méta-modèle pour la spécification des processus d'extraction d'une architecture logicielle. JFL3, 29-31 Octobre, 2015, UL-Hadath, Lebanon.

UNIVERSITE BRETAGNE LOIRE

الجامعة اللبنانية
UNIVERSITE LIBANAISE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Mira ABBOUD

Contribution à l'élaboration d'un processus d'extraction des architectures logicielles: Méta-modèle, méthode et outil

A contribution to the development of a software architecture extraction process: Meta-model, method and tool.

Résumé

Face à la complexité croissante des systèmes logiciels, les architectures logicielles sont apparues comme un allié précieux pour la conception et la maintenance de ces systèmes. Cependant, pour de nombreux systèmes, la représentation de leur architecture n'est pas fiable ; elle est soit indisponible, soit insuffisante ou soit non mise à jour. Pour pallier ce problème qui met en danger la maintenance, l'évolution, la réutilisation et la migration d'un système, l'extraction d'une architecture du système est souvent proposée comme une bonne alternative. L'extraction d'une architecture logicielle est définie comme la science de l'analyse et de la conversion du code source en une architecture logicielle. Cette thèse contribue à apporter une solution au problème d'inexistence d'outil de mesure pour les processus d'extraction d'une architecture logicielle. Ainsi, nous proposons un méta-modèle appelé SArEM qui spécifie les différents processus d'extraction d'une architecture logicielle. Le méta-modèle est basé sur le méta-modèle SPEM et couvre les principaux concepts des processus d'extraction d'une architecture logicielle.

En outre, nous fournissons un outil qui permet aux architectes de construire leur propre processus, d'interagir avec les sorties générées et de découvrir une architecture logicielle conforme à leurs souhaits. Plus précisément, nous proposons une approche d'extraction d'une architecture logicielle appelée SAD basée sur ECD. SAD consiste à considérer l'extraction d'une architecture logicielle comme un processus de découverte de nouvelles connaissances. Ainsi, notre contribution est articulée autour deux points : le premier point est la suggestion d'un processus générique pour l'extraction d'une architecture logicielle et le second point est l'élaboration d'une extension d'un outil ECD qui supporte l'exécution des processus d'extraction d'une architecture logicielle.

Mots clés

Architectures logicielles, SArEM, Extraction des architectures logicielles, Modélisation des processus d'extraction d'une architecture logicielle, Ingénierie inverse, SPEM, SAD, ECD.

Abstract

Face to the exponential growth in the size and complexity of software systems, software architectures emerge as a valuable ally for the design and maintenance of these systems. However, for many systems, their architecture representation is not reliable; it might be unavailable, insufficient, or out of date. To overcome this problem that puts the system maintenance, evolution, reuse and migration in danger, the extraction of the system architecture is proposed. The latter is defined as the science of analyzing and converting the source code to a software architecture. The thesis treats the gap of a measurement tool for software architecture extraction processes. We propose a meta-model called SArEM (Software Architecture Extraction Meta-model) that specifies the software architecture extraction processes. The meta-model is based on SPEM meta-model and covers the main concepts of software architecture extraction processes.

Furthermore, we provide a manner that allows the architects to build their own process, interact with the generated outputs and discover a software architecture that satisfies them. Specifically, we propose a software architecture extraction approach called SAD (Software Architecture Discovery) based on KDD. SAD consists in considering the extraction of a software architecture as a process of discovering new knowledge. Thus, the contribution is centered on two points: the first point is the suggestion of a generic software architecture extraction process and the second point is the elaboration of a KDD tool extension that supports the execution of software architecture extraction processes.

Key Words

Software architectures, SArEM, Software architectures extraction, Modeling of software architecture extraction processes, Reverse engineering, SPEM, SAD, KDD.