Architectures dynamiques dans le contexte des applications à base de composants et orientées service

Mohammed Karim Guennoun

École doctorale EDSYS

Architectures Dynamiques dans le Contexte des Applications à Base de Composants et Orientées Services

THÈSE

pour l'obtention du

Doctorat de l'université TOULOUSE III – PAUL SABATIER

(spécialité Systèmes Informatiques)

par

Mohammed Karim GUENNOUN

Composition du jury

Président :	Guy JUANOLE
Rapporteurs :	Laurence DUCHIEN Mohamed MOSBAH
Examinateurs :	Khalil DRIRA Michel DIAZ Dirck JANSSENS

Laboratoire d'Analyse et d'Architecture des Systèmes, LAAS-CNRS

Remerciements

Ce travail a été effectué au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Aussi, je tiens à remercier Monsieur Malik Ghallab, Directeur du L.A.A.S pour m'avoir accueilli dans son laboratoire.

C'est un plaisir pour moi de remercier tout d'abord Monsieur Khalil Drira, Directeur de cette thèse, dont j'ai pu apprécier les qualités tant scientifiques qu'humaines. La rigueur scientifique, la très grande compétence, ses encouragements et sa gentillesse, ont été déterminants pour les résultats obtenus.

Mes remerciements vont également à Monsieur Michel Diaz, Co-encadrant de ma thèse, pour sa vision scientifique et son support important.

Je tiens aussi à remercier Messieurs J.P. Courtiat et F. Vernadat respectivement, ancien et actuel Chefs du groupe "Outils et Logiciels pour la communication"(OLC), pour m'avoir accueilli au sein de leur groupe.

Madame Isabelle Duchien et Monsieur Mohamed Mosbah ont accepté de consacrer une partie de leur temps précieux pour juger mon travail en tant que rapporteurs. Je leur témoigne tout mon respect et ma reconnaissance.

Je remercie Messieurs Dirck Janssens et Guy Juanole pour avoir accepté de participer au jury de cette thèse. Leurs remarques et leurs questions m'ont permis de mieux exprimer les contributions de ce travail et d'appréhender ses perspectives ; je les en remercie.

Les résultats présentés dans ce mémoire sont le fruits de collaborations importantes avec monsieur Christophe Chassot chercheur au LAAS, et avec messieurs Rony Ghostine, Moncef Sadoq et Mahmoud Al Samad dans le cadre de leurs Masters de recherche. Je les remercie de leurs précieuses contributions.

Ce doctorat doit, aussi, beaucoup à ma Mère et à mon Père, dont les bénédictions m'ont suivi tout au long, et sans lesquelles je n'en serais pas là. Je leur exprime toute mon admiration, mon affection et ma gratitude.

Mes remerciement vont aussi à Safia qui, à sa manière, m'a énormément aidé ; je lui exprime mon admiration pour son courage et sa patience et ma gratitude pour son soutien indéfectible.

Je ne saurais terminer cette page, sans adresser mes remerciements à tous les amis qui m'ont encouragé et assisté de leur mieux : ce sont Slim Abdellatif, Riadh BenHalima, Ismael Bouassida-Rodriguez, Otmane Bouchama, Najla Chamssedine, Mohamed El Masri, Tarik Khoutaif, Francisco Moo-Mena, Olga Nabuco, Florent Peres, Yassir Salah ainsi que mes autres amis des groupes OLC et TSF.

Mes remerciements vont, enfin, à tout le personnel technique du LAAS dont le concours a grandement facilité la réalisation de ce travail.

Mohammed Karim GUENNOUN
Décembre 2006

Aime la vérité mais pardonne à l'erreur.
François Marie Arouet, dit Voltaire

Table des matières

Table des figures

Introduction

Les travaux présentés dans ce mémoire se situent dans le contexte des applications distribuées à base de composants et orientées services. La problématique traitée concerne la description, la gestion, la vérification et l'implémentation de l'auto-reconfiguration dans ce type de systèmes. Cette problématique constitue l'un des enjeux et défis scientifiques les plus importants dans la mesure où l'adaptabilité automatique d'une application est devenue impérative dans un contexte de systèmes fortement distribués et contenant un large nombre de composants. Dans ce cas, une reconfiguration ad-hoc n'est pas envisageable. Le caractère adaptable ou dynamique de ces nouvelles applications est lié, par exemple, à des contraintes relatives à la qualité de service, à la robustesse ou à la sécurité. Le but étant d'offrir des services de plus en plus autonomes, robustes et sûrs, et qui s'adaptent aux différents profils et besoins des clients.

L'adaptabilité dans les applications logicielles peut être séparée en deux catégories. La première concerne l'adaptation comportementale appelée aussi adaptation algorithmique. Cette adaptation traite la redéfinition du comportement de l'application et de ses composants et impliquerait, par exemple, l'introduction d'une nouvelle méthode dans l'interface d'un composant ou le changement du protocole d'orchestration qui coordonne un ensemble de services. La deuxième catégorie, dans laquelle on peut classer nos travaux, concerne l'adaptation structurelle et implique une reconfiguration au niveau architectural. Ce type de reconfiguration porte sur l'organisation de l'architecture et consiste, par exemple, à remplacer un composant défaillant par un autre composant qui possède les mêmes fonctionnalités ou rediriger un client d'un service qui ne respecte pas le contrat de QdS vers un service susceptible d'offrir de meilleures garanties de QdS.

Nous introduisons, dans ce mémoire, une approche orientée-modèle pour une adaptation structurelle automatique pendant l'exécution. Cette approche est basée sur des techniques orientées-règles pour la reconfiguration architecturale dynamique. Ainsi, chaque étape élémentaire de l'évolution de l'architecture est décrite par une combinaison de règles de transformation de l'architecture. Chacune de ces règles spécifie les actions de transformation élémentaires relatives, par exemple, aux composants logiciels qu'il faut introduire, supprimer, activer ou désactiver, ou aux liens d'interdépendances (entre composants) à introduire ou à supprimer. Le protocole de gestion dynamique de l'architecture est défini par l'ensemble des événements impliquant une reconfiguration (e.g. requête d'un client, violation de la QdS, panne d'un service...etc) et la combinaison des règles de transformation à appliquer pour exécuter les actions correspondantes.

Le méta-modèle que nous définissons dans ce manuscrit, couvre plusieurs maillons de la chaîne de description architecturale. Les instances des architectures sont décrites par des graphes étendus où les composants (ou les services) sont représentés par des nœuds et les interdépendances (e.g. les connexions, les relations de contrôle ...etc) sont décrites par des arcs. Les styles architecturaux, permettant de décrire toutes les instances consistantes d'une application à architecture dynamique, sont spécifiés par des grammaires de graphes. Ces grammaires de graphes sont étendues pour augmenter le pouvoir d'expression du méta-modèle. Les extensions concernent, par exemple, l'introduction de variables typées dans les productions de grammaires et la notion de restrictions à l'applicabilité de ces productions. Le méta-modèle permet aussi de décrire le protocole de gestion de l'architecture par la spécification des règles de transformation de graphes correspondant aux étapes élémentaires de son évolution dynamique.

Ce méta-modèle traite aussi la description des contraintes et des propriétés architecturales d'une application (e.g. structure architecturale de l'application pouvant être par exemple arborescente ou en anneau, nombre maximal d'imbrications pour la composition des services, cardinalité des liens entre différents types de composants...etc). Ce type de contraintes est spécifié par l'absence ou la présence

de motifs dans les graphes décrivant l'architecture. Leur vérification automatique est réalisée à l'aide d'algorithmes de recherche d'homomorphismes.

La définition de systèmes à base de grammaires de graphes étendues permet d'obtenir des descriptions raffinées selon différents points de vues. Ces points de vues sont relatifs aux différentes facettes de la description et peuvent concerner, par exemple, le passage d'une description complète de l'architecture vers une description de plus haut niveau faisant abstraction des services élémentaires et ne prenant en compte que les services de plus haut niveau. Un autre intérêt de l'approche de description par points de vues, concerne la possibilité d'extraire des paramètres et des propriétés architecturales (e.g. le nombre maximal de services élémentaires composant un service composite, structure de l'orchestration, interdépendances entre composants) et permettre la vérification automatique de ces propriétés. De plus, cette vérification peut s'avérer beaucoup plus aisée quand elle est réalisée au niveau de chaque point de vue en comparaison à une vérification au niveau de description globale dont le modèle peut être de très grande taille.

Dans des travaux antérieurs, nous avons éprouvé notre approche par différents cas d'étude. Dans un premier temps, nous l'avons appliquée aux styles architecturaux élémentaires pour les applications orientées composants. On a spécifié les grammaires de graphes permettant de décrire les styles architecturaux 2-tiers, 3-tiers, composites et hiérarchiques. Pour les applications orientées services nous avons défini les grammaires de graphes permettant la description du style d'interaction de base (i.e. producteurs-consommateurs de services) ainsi que des styles architecturaux prenant en compte l'orchestration et la composition des Services Web. Nous avons aussi défini des descriptions de styles relatifs à des applications coopératives telles que l'édition partagée et la revue coopérative. Pour tous ces différents styles architecturaux, nous avons défini des protocoles de gestion de l'architectures et des contraintes architecturales. Nous avons, aussi, défini plusieurs points de vues pour la description et proposé des systèmes basés sur les grammaires de graphes pour extraire des descriptions et des contraintes relatives à différents points de vue. Nous avons présenté les preuves pour la préservation des styles architecturaux par les protocoles proposés ainsi que pour la validité des propriétés architecturales.

Dans ce manuscrit, nous traitons de manière détaillée un cas d'étude relatifs aux opérations d'intervention d'urgence. Dans le cadre de ce cas d'étude, nous considérons trois niveaux architecturaux relatifs aux niveaux application, middleware et réseau. Pour chaque niveau, nous spécifions les styles architecturaux ainsi que les protocoles de reconfiguration. Des systèmes d'abstraction et de raffinement sont définis pour permettre de transformer des descriptions considérées à un niveau architectural vers les descriptions correspondantes dans un autre niveau architectural.

Afin de pouvoir générer de manière automatique les différentes transformations de l'architecture ainsi que les vérifications de validité de propriétés, nous avons été amenés à développer un algorithme de recherche d'homomorphismes de graphes et un algorithme de transformation de graphes. Ces algorithmes ont été définis pour pouvoir traiter ces problématiques dans le cadre des graphes et des grammaires de graphes étendues définies pour notre méta-modèle. L'analyse de complexité des algorithmes ainsi que les résultats expérimentaux obtenus ont permis de conclure à leur efficacité quant au traitement de la gestion des architectures. Une autre version des deux algorithmes a été définie profitant de la spécificité de notre contexte d'utilisation (i.e. l'ensemble des règles de transformation est généralement très stable). L'analyse de complexité des nouvelles versions donne des résultats encore plus performants.

L'étape suivante a concerné l'implantation des algorithmes de recherche d'homomorphismes et de transformation de graphes. Cette implantation a été réalisée en C++ préféré à Java pour des considérations liées à la performance. Le logiciel réalisé a été étendu pour la définition et l'exécution du protocole de gestion de l'architecture. Deux API C++ et Java ont été définies permettant d'initialiser ce protocole en définissant les différents graphes de description initiaux, les règles de transformation, les événements déclenchant les différentes reconfigurations de l'architecture ainsi que les politiques d'application des règles pour chaque type d'événement.

Chapitre 1

État de l'Art

1.1 Introduction

Nous exposons, dans ce chapitre dédié à l'état de l'art, différents travaux de la littérature qui se situent dans les mêmes domaines ou adressent les mêmes problématiques que les approches et contributions présentées dans ce manuscrit. Dans les chapitres suivants, nous positionnerons nos choix de modélisation par rapport à ces travaux et confronterons nos résultats théoriques et expérimentaux aux leurs.

Nous couvrons, dans ce chapitre des domaines de recherche très divers. Nous commençons par présenter les architectures logicielles et leur description. Nous donnons les définitions et spécifications introduites par les organismes de standardisation. Ensuite, nous introduisons les langages de descriptions d'architectures (ADLs). Pour ces langages, nous focalisons surtout sur la description de l'aspect dynamique des architectures. Dans ce cadre, nous étudions les ADLs *Darwin*, *Rapide*, *Wright* et *C2SADL* réputés êtres les langages les plus aboutis et les plus utilisés dans le monde académique et industriel. Nous illustrons les différentes approches utilisées par le même exemple simple du producteur-consommateur.

Pour situer le cadre formel de nos contributions, nous introduisons, ici, le modèle formel des graphes et des grammaires de graphes. Dans un premier temps nous présentons différents types de graphes correspondant à différents besoins de modélisation. Ensuite, nous ferons un tour d'horizon des modèles utilisés pour la spécification des règles de réécriture, des limitations de ces modèles et des diverses extensions qui peuvent être considérées.

Nous introduisons, par la suite, la problématique de la recherche d'homomorphismes de graphes. Nous présentons les techniques et les algorithmes classiques qui traitent de cette problématique.

1.2 Description des architectures logicielles

Les systèmes logiciels actuels sont de plus en plus larges et complexes. Leur bon fonctionnement requiert des propriétés liées à l'adaptabilité, la réutilisabilité, et la mobilité. L'architecture logicielle est une discipline qui a émergé pour répondre à ce contexte en faisant la distinction entre le niveau conception et organisation d'un système et entre le niveau implémentation. Le premier niveau concerne la structure conceptuelle et l'organisation des différents composants tandis que le deuxième niveau est relatif à la réalisation des entités logicielles concernant, par exemple, l'algorithmique et les structures de données. Cette séparation et la prise en compte du seul haut niveau d'abstraction permet de faciliter la compréhension et la vérification de systèmes de grande taille puisqu'elles allègent la description et la débarrassent des détails relatifs à l'implémentation. D'autre part, cela permet aussi de favoriser la réutilisation des composants et aider à la maintenance des architectures.

Plusieurs définitions on été énoncées pour définir l'architecture logicielle. Une première définition a été donnée par le standard IEEE 610.12 [Std90] où une architecture est définie comme "*la structure organisationnelle d'un système ou d'un composant*". Dans [GP95] l'architecture logicielle est définie comme "*la structure des composants d'un programme ou d'un système, leur interdépendances, et les principes et*

directives régissant leur conception et leur évolution". Une dernière définition a été introduite par le standard IEEE 1471 [Std00] qui définit l'architecture comme étant "*l'organisation fondamentale d'un système relative aux composants qui le constituent, aux relations entre ces composants et avec l'environnement, et aux principes guidant la conception du système et son évolution*". Il paraît donc, d'après ces définitions[1], que l'évolution dynamique a été identifiée comme un aspect important des architectures logicielles.

Le standard IEEE 1471 définit la description d'architecture comme "*une collection de produits pour documenter une architecture*" et le comité de standardisation pour l'ingénierie logicielle (Software Engineering Standards Committee : *SESC*)[EHP+96] la définit comme étant "*le modèle ou le document ou tout autre artefact permettant de communiquer et d'enregistrer l'architecture d'un système*". Cet artefact est censé inclure la description des aspects de haut niveau tels que l'organisation globale du système, la décomposition sous forme de composants, la description des fonctionalités des composants et la manière avec laquelle ces composants interagissent [SDK+95]. *SESC* définit aussi les trois concepts élémentaires de la description de l'architecture logicielle comme étant :

- Le *composant*, défini comme l'élément structurel principal de l'architecture.
- La *connexion* (ou le *connecteur*), définie comme une relation entre les composants, et pouvant, par exemple, correspondre à des liens de contrôle ou des flux de données.
- Les *contraintes*, représentant les lois que le système doit observer. Ces contraintes peuvent être relatives à des composants ou à des connexions, et sont de trois types :
  - les contraintes de type style d'architecture.
  - les contraintes reflétant des exigences non fonctionnelles telles que la qualité de service ou la sécurité.
  - les lois qui régissent l'accès aux ressources.

Nous retrouvons deux types principaux désignés par le *SESC* pour la description des architectures : 1) la description de type conception, utilisée comme un moyen d'exprimer des caractéristiques architecturales de haut niveau du système, et de définir et organiser ses éléments et leurs interactions, et 2) la description de type style, permettant de raisonner sur l'architecture, par exemple, en terme de compatibilité, d'interopérabilité, et d'interchangeabilité de ses composants.

1.2.1 Les ADLs

Les langages de description des architectures (Architecture Description Language : *ADL*) ont pour objectif de fournir des notations expressives afin de représenter la structure conceptuelle d'un système. De tels langages ont pour ambition d'aider à communiquer et à raisonner sur les systèmes et doivent pour cela définir : 1) une syntaxe et une sémantique précises pour résoudre les ambiguïtés et aider à la détection des inconsistances, et 2) un ensemble de techniques et d'outils pour raisonner sur les propriétés des systèmes décrits.

Nous retrouvons, dans la littérature, un nombre considérable d'ADLs qui peuvent être classés en ADLs spécialisés ou généralistes. Nous pouvons citer parmi les ADLs spécialisés, les AADLs (Avionics ADL) [AAD06] tels que le langage MetaH [VB93, McD01] traitant la description des applications temps-réel embarquées et le langage Cotre [FBB+03, COT06] développé en partie au LAAS-CNRS et chez Airbus. On peut citer aussi le langage SADL (Structural ADL) [MR97] visant la description et la vérification d'architectures hiérarchiques et multi-niveaux, et C2SADL [MRT99] qui traite de la description des architectures dynamiques. Parmi les ADLs généralistes on retrouve les langages Rapide [LKA+95, LV95], Darwin [MDK92, MDEK95], et Wright [All97].

La totalité des ADLs intègre la notion du *composant* qui permet de modéliser des unités de calcul ou de communication indépendantes et réutilisables ou d'autres objets du niveau utilisateur tels que des fichiers ou des bases de données. Ces composants sont définis selon l'approche boîte noire où leur structure interne et leur implémentation sont cachées et seule leur interface d'intéraction est visible par les autres composants.

Concernant les connexions, nous retrouvons globalement deux visions : la première vision considère une connexion comme un simple lien entre exactement deux composants (c'est le cas notamment de Darwin et Rapide), tandis qu'une deuxième vision introduit une entité spécifique appelée *connecteur* qui

[1] les définitions sont citées dans l'ordre chronologique de leur publication.

permet de lier un groupe de composants (c'est le cas notamment de Wright et C2SADL). Le connecteur offre la possibilité, en plus de spécifier le lien de communication qu'il matérialise, de décrire le protocole d'interaction liant les composants qu'il connecte (i.e. le type des messages échangés et l'ordre causal de leur envoi et de leur réception). La considération de l'entité connecteur ou sa non prise en compte n'est cependant pas une caractéristique déterminante pour la puissance d'expression des ADLs. Les connecteurs peuvent être simulés, dans la description, par des composants qui décrivent le protocole d'interaction qui leur correspond.

1.3 Description des architectures logicielles dynamiques

Les architectures dynamiques sont définies dans [CPT99] comme étant "*celles qui décrivent comment les composants sont créés, interconnectés, et supprimés pendant l'exécution, et qui permettent la reconfiguration de leur topologie de communication pendant l'exécution*".

La littérature liée aux architectures dynamiques couvre un large panel de domaines d'application où l'évolution de l'architecture est une exigence essentielle. Ainsi, la reconfiguration de l'architecture peut, dans le cadre d'une politique de sécurité, être la réponse à des attaques extérieures. [WHK+00] donne deux exemples qui rentrent dans ce cas de figure concernant une reconfiguration dite *proactive* pour augmenter la résilience du système à un type spécifique d'attaque et une reconfiguration dite *réactive* pour restaurer l'intégrité du système si une intrusion est détectée et le système suspecté d'être dans un état corrompu. L'évolution dynamique peut aussi être vue comme un mécanisme de tolérance aux fautes [RBC+03] permettant d'offrir des services fiables en reconfigurant l'architecture du système suite à l'occurrence de pannes. [BSHL01] utilise la reconfiguration de l'architecture par le redéploiement de ses composants et par le changement de leur comportement comme un outil pour la gestion de la charge et de la mobilité. [NSDC04] utilise la reconfiguration de l'architecture pour répondre à des exigences relatives à la qualité de service.

Le caractère dynamique dans les architectures logicielles introduit des difficultés supplémentaires concernant la description. En effet, Dans le cas d'une architecture statique, la description consiste à "simplement" énumérer les différents composants et connexions qui la constituent. Cette approche est inadaptée pour la description de architectures dynamiques dont la configuration, par définition, change pendant l'exécution et dont les composants et les connexions peuvent être introduits et supprimés tout au long du cycle de vie de l'application.

Pour la description des architectures dynamiques, nous distinguons deux aspects :

- Les aspects de type déclaratif, qui incluent la description de toutes les instances valides de l'architecture, l'ensemble des changements permis sur l'architecture, ainsi que les contraintes de type topologique ou fonctionnel,
- les aspects opérationnels, qui spécifient la gestion de l'évolution de l'architecture. Cet aspect concerne la définition des événements impliquant une reconfiguration de l'architecture et la spécification des actions de transformation qu'elle subit en réaction à ces événements.

1.3.1 Description de l'évolution dynamique des architectures dans les ADLs

Les ADLs abordent plusieurs problématiques de description mais portent, dans leur grande majorité, un regard statique sur les architectures [ADG97, MT00, Geo02, Tor02, RS02]. Nous présentons, dans ce qui suit, les ADLs qui constituent les exceptions les plus notables considérant la description de l'aspect dynamique. Nous étudions les langages Darwin, Rapide, C2SADL et Wright. Nous illustrons les différents mécanismes par un exemple d'architecture orientée événements selon le modèle producteur-consommateur. Nous intégrons des actions de reconfiguration selon les possibilités offertes par chaque langage.

Darwin

Le langage de description Darwin, se base sur une description hiérarchique d'instances de composants interconnectés. Chaque niveau de la hiérarchie représente un composant dit de type composite, et les

feuilles de la hiérarchie représentent les instances des composants dits de type primitif. Chaque type de composant est décrit via son interface qui est constituée d'une collection de services qu'il procure (i.e. déclarés par le composant), et de services qu'il attend (i.e. qui se produisent dans l'environnement). Les configurations de l'architecture du système sont construites par la déclaration des instanciations des composants et le raccordement (*binding*) entre services procurés et attendus.

Un exemple de description de l'exemple producteur-consommateur est donné dans la figure 1.1 où nous définissons le composant composite *ProdCons* avec une interface vide. Ce composant est constitué d'une instance d'un composant de type *Producteur* et d'une deuxième instance appartenant au type *Consommateur*. Le type producteur offre un service nommé *out* tandis que le composant de type consommateur requiert un service nommé *in*. Les deux instances des composants primitifs *Producteur* et *Consommateur* sont reliés en connectant le service offert par le premier au service requis par le deuxième.

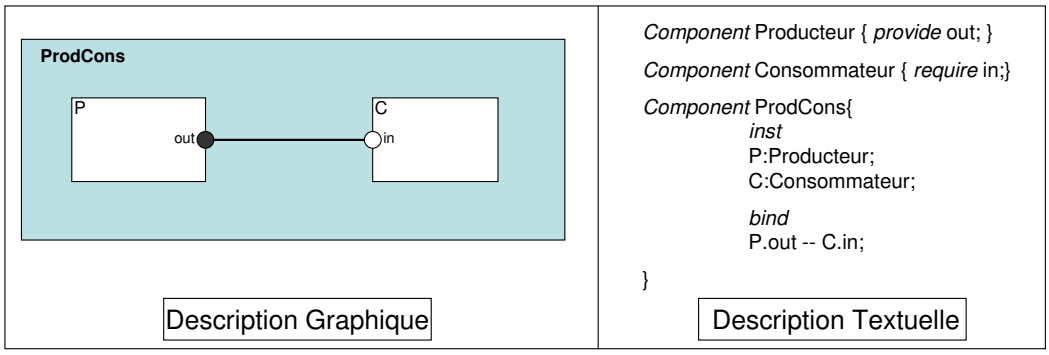


FIG. 1.1 – Exemple d'une description Darwin pour un composant composite constitué d'un producteur et d'un consommateur

Darwin [MK06] offre deux approches pour décrire l'aspect dynamique des architectures. La première approche appelée *instanciation paresseuse* (*lazy instantiation*) permet de retarder l'instanciation de certains composants. Ainsi, un composant offrant un service et déclaré dynamiquement en utilisant cette approche, ne sera instancié que lorsqu'un utilisateur de ce service tente d'y accéder.

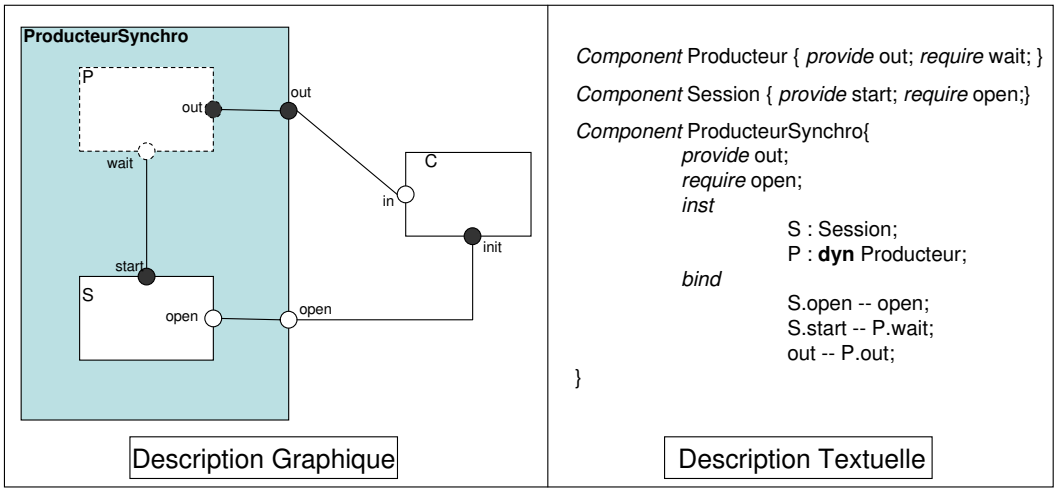


FIG. 1.2 – Exemple d'une description Darwin impliquant une instanciation dynamique paresseuse

Dans l'exemple donné dans la figure 1.2, nous décrivons un composant composite ProducteurSynchro. Ce composant contient un composant *Session* qui ouvre la session entre le consommateur et le producteur et introduit une étape de synchronisation entre les deux composants. La déclaration de l'instance du composant de type producteur avec le préfixe "*dyn*" implique que son instanciation ne sera effective qu'à la suite de la requête d'accès à son service "*wait*" par l'instance du composant *Session* à travers l'événement "*start*".

La deuxième approche permettant de décrire l'évolution dynamique de l'architecture concerne ce qui est appelé dans la sémantique Darwin l'instanciation dynamique directe (*direct dynamic instantiation*). Cette approche permet la définition de structures qui peuvent évoluer par réplication de composants. Chaque événement introduit une nouvelle instance d'un composant spécifié. Les connexions des composants créés selon cette approche sont définies de manière générique pour chaque composant répliqué.

Un exemple illustrant cette approche est présenté dans la figure 1.3. Dans cet exemple, nous avons défini un composant de type Producteur qui possède un port "*out*" permettant d'envoyer des messages vers les clients. Ce composant, tel qu'il est décrit ici, possède aussi un port permettant de créer des instances d'autres composants. Le composant de type "multiDiffusion" est, donc, décrit de telle manière qu'à chaque requête sur le port "*new*" du composant producteur, une nouvelle instance de type consommateur est créée et son port "*in*" connecté au port "*out*" du producteur.

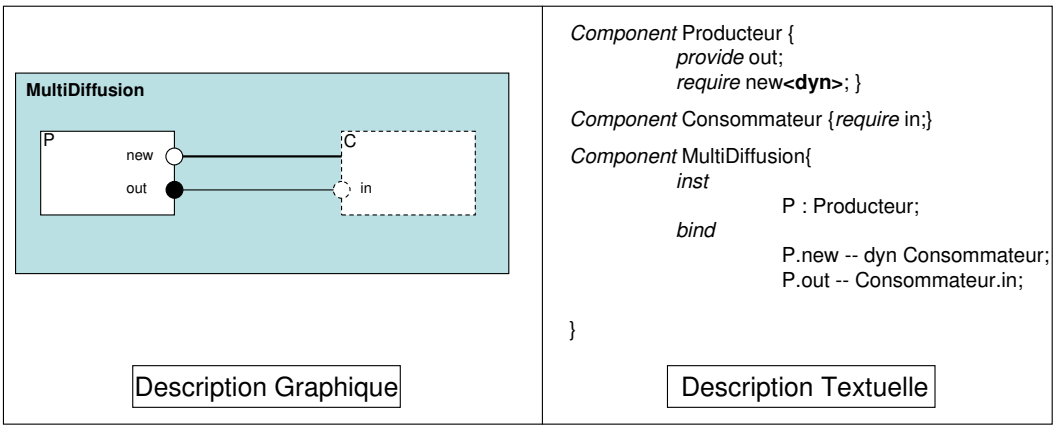


FIG. 1.3 – Exemple d'une description Darwin impliquant une instanciation dynamique directe

Les deux approches définies par le langage Darwin pour la description des architectures dynamiques correspondent au traitement de plusieurs cas classiques de reconfiguration. Elles présentent, néanmoins, plusieurs limitations.

Ainsi, la première approche nécessite d'énumérer et de déclarer tous les composants potentiels de l'architecture. Elle est inadéquate si, par exemple, l'architecture possède un nombre non borné de composants. Cette approche n'adresse réellement l'évolution dynamique de l'architecture que du point de vue de l'activation des composants.

La deuxième approche adresse l'évolution dynamique par la création de nouvelles instances de composants, mais le fait que les instances créées par réplication soient anonymes et soient, donc, toutes connectées de la même manière constitue une limitation très contraignante. De plus, comme le langage Darwin interdit de connecter plusieurs services offerts à un seul service requis, le raccordement des services offerts par les composants créés par réplication ne peut être décrit par Darwin. Un autre aspect non considéré par Darwin est la spécification des suppressions de composants ou de raccordements.

Wright

Le langage Wright permet, en plus de la description de la structure de l'architecture, la description formelle du comportement des composants et des connecteurs. Le comportement des composants ainsi que la coordination réalisée par les connecteurs sont décrits en utilisant une notation basée sur l'algèbre de processus CSP. Cette notation formelle permet aussi de réaliser des vérifications de contraintes ainsi que la définition de styles d'architecture grâce à l'introduction des déclarations de types d'interfaces et à la paramétrisation de ces types. Trois notions principales sont considérées par le langage Wright :

- Le composant, dont la description contient une partie interface et une partie calcul (*Computation*). L'interface est constituée d'un ensemble de *ports* où chaque port décrit une interaction à laquelle peut participer le composant alors que la partie calcul (*Computation*) décrit la coordination entre les différentes interactions sur les différents ports.
- Le connecteur représentant l'interaction entre un ensemble de composants. Il est composé d'un ensemble de *rôles* qui spécifient le comportement des participants dans l'interaction, et d'une partie *glue* décrivant comment est réalisée la coordination entre les différents participants.
- la configuration permettant de décrire toute ou partie de l'architecture d'un système. Sa description est spécifiée par la collection d'instances de composants et de connecteurs qui la constituent, ainsi que par la description des connexions reliant ces différentes instances.

Un exemple décrivant une architecture producteur-consommateur est donné dans la figure 1.4. Nous présentons dans cet exemple la description Wright pour les deux types relatifs aux producteurs et consommateurs ainsi que la description du type de connecteur "*Lien*" permettant la coordination de la production et de la consommation de messages. Les notations CSP [Hoa85] présentés dans l'exemple doivent être interprétées de la manière suivante : $\overline{abc}$ correspond à un événement produit par le composant, abc correspond à un événement consommé par le composant, l'opérateur $\sqcap$ exprime un choix interne au composant tandis que l'opérateur $\Box$ exprime un choix externe au composant. § représente le processus vide.

La première version de Wright [Gar06] présente une description purement statique des architectures, mais une extension pour la description de l'aspect dynamique a été introduite dans [AGD97, ADG98]. Cette extension, appelée *Dynamic Wright*, a introduit des événements de contrôle permettant de spécifier les conditions sous lesquelles les transformations de l'architecture sont autorisées. Ces événements de contrôle induisent l'exécution d'une séquence d'actions élémentaires de reconfiguration comprenant l'introduction et la suppression de composants et de connecteurs (respectivement les actions *new* et *del*) et l'introduction et la suppression des liens (respectivement les actions *attach* et *detach*). Les programmes de reconfiguration (appelés *Configuror*) spécifient les politiques qui caractérisent l'évolution dynamique de l'architecture globale.

L'exemple présenté dans la figure 1.5 présente le cas d'une architecture producteur-consommateur tolérante aux pannes. La configuration décrite considère un producteur, un consommateur principal et un consommateur secondaire. Deux événements de type contrôle pour la reconfiguration sont pris en compte. Le premier événement concerne une occurrence de pannes dans le consommateur principal (i.e. Primaire.control.panne) et le second événement concerne sa réparation (i.e. Primaire.control.OK). Le composant *"Configuror"* permet de gérer la reconfiguration de l'architecture en spécifiant qu'à l'occurrence d'une panne du consommateur principal, les messages du producteur sont redirigés vers le consommateur secondaire. À la réparation du consommateur principal, le composant de reconfiguration rétablit le lien d'interaction avec celui-ci.

Le formalisme de description pour les architectures dynamiques défini par l'extension *dynamic Wright* est plus expressif que celui spécifié par Darwin. Cependant, sa puissance d'expression reste limitée par le fait qu'il implique l'énumération de toutes les configurations possibles de l'architecture. Dans la mesure où la taille des architectures dynamiques (en nombre de composants et de connecteurs) est généralement non bornée et que, par conséquent, le nombre de configurations possibles est infini, cette limite réduit le champ d'application de ce formalisme aux applications de petite taille avec une faible composante dynamique.

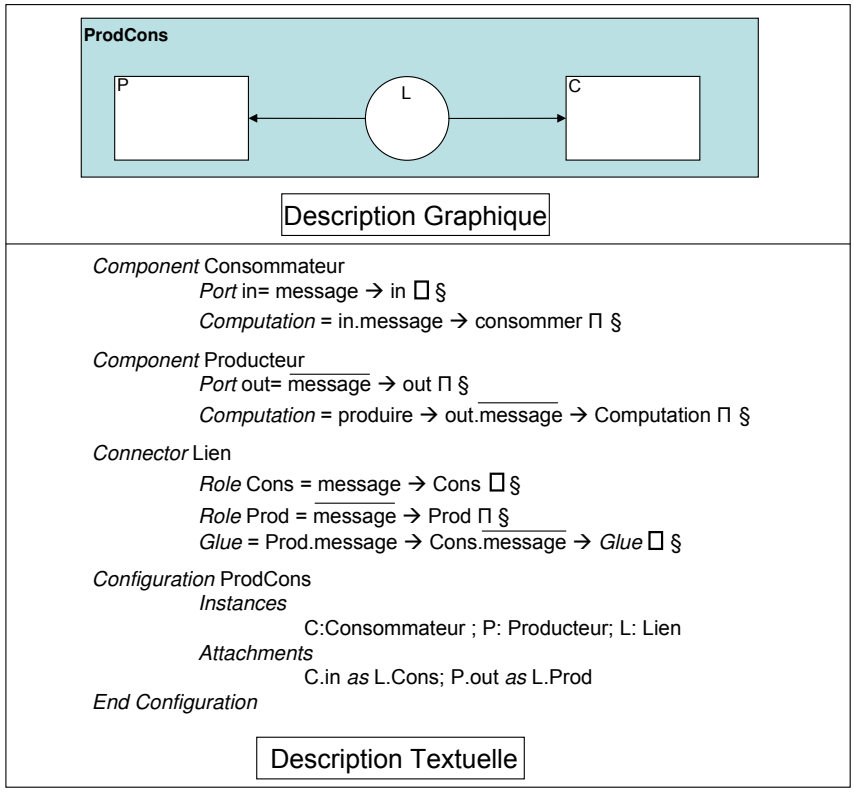


FIG. 1.4 – Exemple d'une description Wright pour le cas du producteur-consommateur

Rapide

Rapide est un langage de description généraliste et exécutable basé sur les événements. Ce langage de description est supporté par plusieurs outils permettant la simulation de l'architecture et du comportement des composants.

Une architecture Rapide [Luc06] est définie par ses composants, par ses connexions et par les différentes contraintes sur sa topologie. Le type des composants Rapide est appelé *interface*. Ces interfaces sont spécifiées par les activités qui sont réalisées ou observées par les composants. Rapide définit deux types de communications, les communications synchrones via la déclaration de fonctions (terminologie provides/requires), et les communications asynchrones via la spécification d'événements (terminologie : in/out). La spécification des interfaces peut aussi contenir une description de leur comportement via des contraintes causales ou temporelles sur les activités. Les connexions entre composants relient les parties émission d'une interface vers les parties réception d'une autre.

Un exemple de description de l'architecture producteur-consommateur est donné dans la figure 1.6. La description qu'on donne ici comprend un consommateur et un producteur.

Le type du "*Producteur*" produit une action d'envoie de message et reçoit une action qui indique l'état courant du consommateur. Le comportement de ce type implique que le composant ne produit des messages que s'il reçoit un état (i.e. *true*) qui indique que le consommateur est prêt à les recevoir.

D'un autre côté, le comportement du type "*Consommateur*" implique qu'il notifie qu'il est prêt dès qu'il a consommé le message reçu.

L'architecture du producteur-consommateur possède un comportement où les deux instances qui le composent (i.e. une instance du type "Producteur" et une instance du type "Consommateur") envoient et consomment des messages à tour de rôles.

La description de l'évolution dynamique des architectures dans la version originelle de Rapide est très

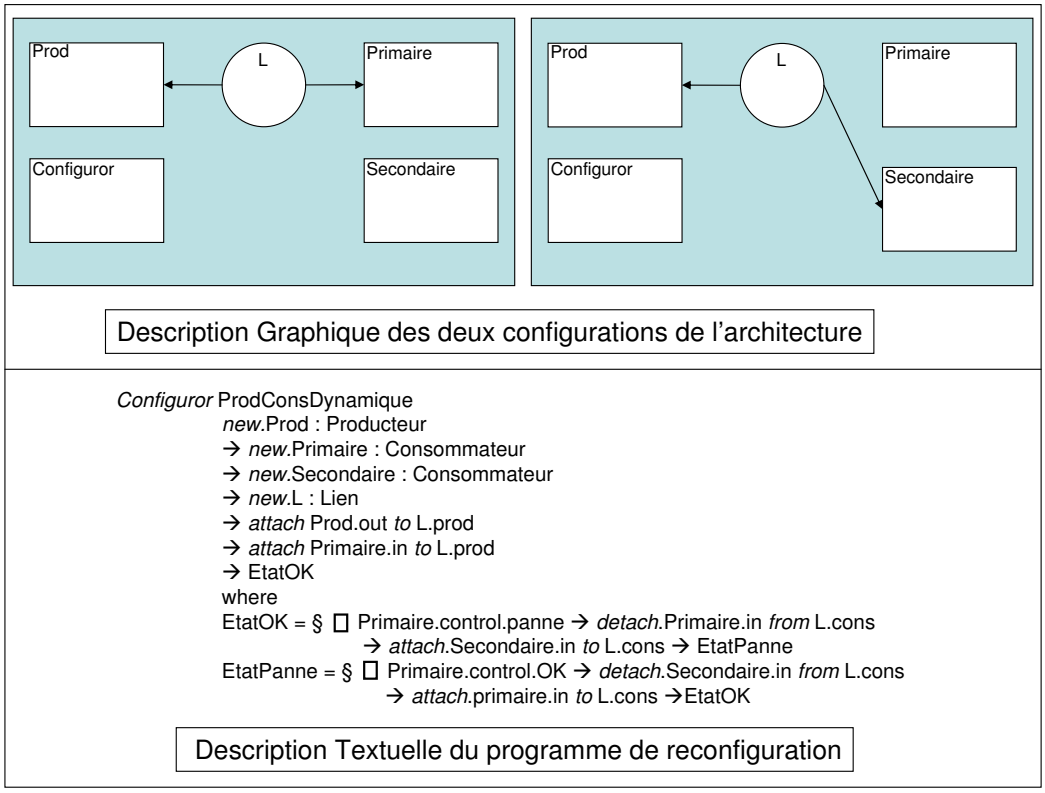


FIG. 1.5 – Exemple d'une description Wright pour le cas de consommateurs avec tolérance aux pannes

liée au comportement de l'architecture et concerne essentiellement le routage des événements et le changement de la topologie de communication. La réflexion concernant les autres aspects dynamiques est présentée, par les auteurs, comme étant encore à ces débuts. Les spécifications publiées dans [Luc06, VPL99] donnent, néanmoins, un début de description de l'évolution dynamique à travers l'introduction d'événements de reconfiguration et des actions "*link*" et "*unlink*" correspondant à l'ajout et à la suppression d'interfaces.

Nous présentons dans la figure 1.7 un exemple d'une architecture producteur-consommateur dynamique prenant en compte la tolérance aux pannes. Le système, décrit dans cet exemple, permet la prise en compte d'un ensemble dynamique de consommateurs. Nous considérons des actions impliquant une reconfiguration de l'architecture par la suppression des consommateurs défaillants (suite à l'action "*Enpanne*") et par l'introduction de nouveaux consommateurs (suite à l'action "*Introduire*").

C2SADL

C2SADL [TMO06] est un ADL basé sur le style architectural C2 (*Chiron-2*) [TMA+96, C2S06]. Les composants C2SADL communiquent de façon asynchrone via l'envoi de messages définis par un nom et une liste de paramètres typés. Cette communication s'effectue à travers deux interfaces appelées "*top*" et "*bottom*" et spécifiées par l'ensemble des messages qui peuvent être envoyés ou reçus.

C2SADL considère les connecteurs comme des entités à part entière chargées essentiellement de la diffusion et du routage des messages entre composants. Les connecteurs permettent le filtrage des messages selon quatre politiques : 1) La politique sans filtrage (appelée "*no_filtering*") qui implique l'envoi du message à tous les composants situés sur le côté concerné (i.e. "*bottom*" pour les notifications et "*top*" pour les requêtes). 2) La notification filtrée (appelée *notification_filtering*) où chaque notification n'est envoyée que vers les composants qui l'attendent. 3) La politique de notification avec priorités (appelée

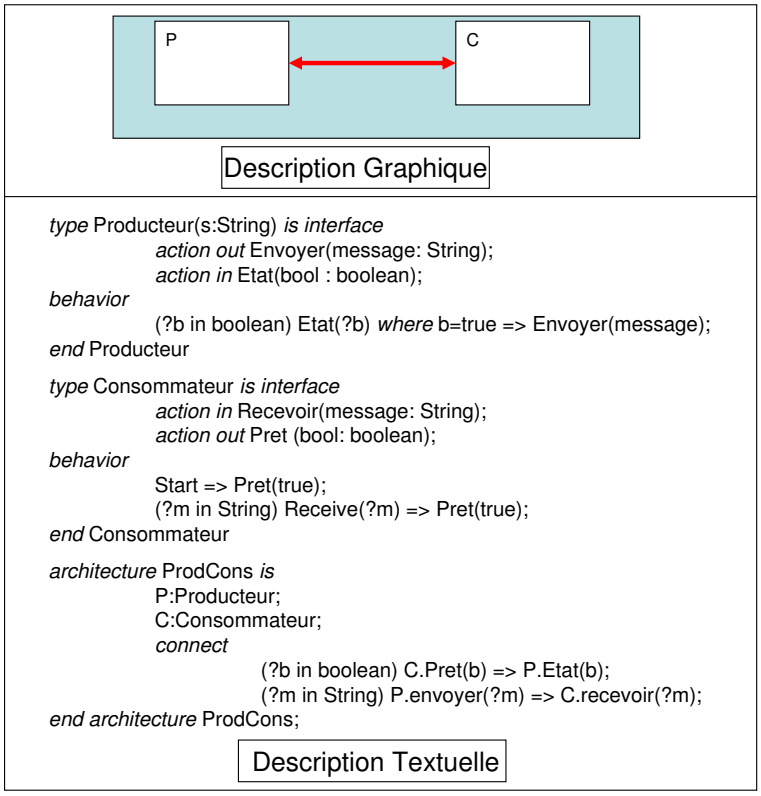


FIG. 1.6 – Exemple d'une description selon Rapide pour le producteur-consommateur

prioritized) permettant de définir des priorités entre les composants et d'envoyer les notifications par ordre de priorité. 4) le puits de messages (*message_sink* dans la terminologie C2) où le connecteur ignore les messages qui lui sont envoyés.

Le style C2 impose des contraintes sur le raccordement des composants et des connecteurs qui impliquent qu'un connecteur peut être relié à plusieurs composants et connecteurs mais spécifie qu'un composant ne peut être connecté, à travers chacune de ces deux interfaces, qu'au maximum à un connecteur.

Les messages échangés sont de deux types correspondant à des *requêtes* ou à des *notifications*. Les notifications sont des publications de la réalisation d'une opération ou du changement d'état d'un composant tandis que les requêtes sont des directives demandant l'exécution d'une opération par un composant. Les messages de type requête ne peuvent être envoyés que vers le haut (i.e. via le port "*top*") et les notifications que vers le bas (i.e. via le port "*bottom*"). La définition d'une architecture implique l'énumération de ses composants et connecteurs et la description de la topologie de leurs interconnexions. Une instance de l'architecture est donnée par l'instanciation des ses composants.

La figure 1.8, donne la description C2SADL de l'architecture producteur-consommateur. Le système comprend une instance d'un producteur de messages et une instance d'un consommateur. Les deux instances sont reliées par un connecteur (appelé *Lien*). Le système est spécifié de telle manière que le producteur attend la requête du consommateur (i.e. *ready()*) qui lui signifie qu'il est prêt. Le producteur lui envoie, par la suite, une notification qui contient le message à consommer (i.e. *Message(m)*). Le filtrage de messages réalisé par le connecteur *Lien* est de type "*no_filtering*".

Pour la spécification de l'évolution dynamique, une notation appelée ACN (*Architecture Construction Notation*) a été introduite dans [MRT99, OT98] permettant la spécification d'opérations de reconfiguration par rajout et suppression de composants et de connexions (*add, remove, weld, et unweld*) et la mise

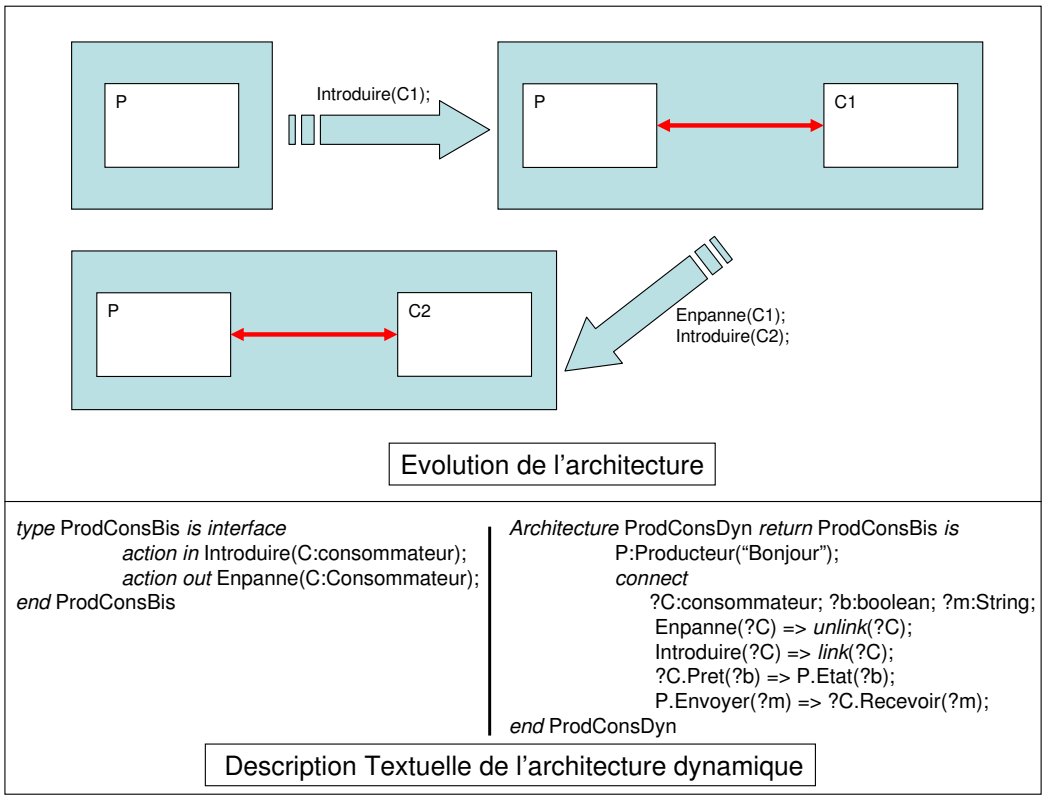


FIG. 1.7 – Exemple d'une description Rapide pour une architecture producteur-consommateur dynamique

à jour des interfaces (*addTopPort, removeTopPort, addBottomPort, removeBottomPort*[2]).

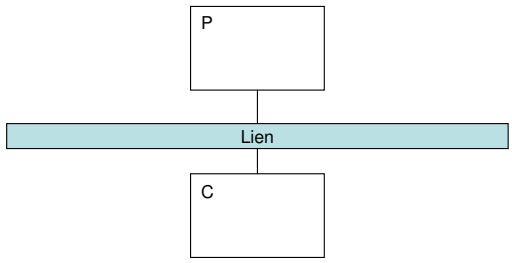
La figure 1.9 donne un exemple d'opérations de reconfiguration pour l'architecture producteur- consommateur. À la suite d'une panne irréversible d'un consommateur, nous l'isolons du producteur en le déconnectant (i.e. unweld), le supprimer (i.e. remove), et connecter à sa place un consommateur auxiliaire (i.e. weld).

C2SADL est certainement, parmi les ADLs présentés ici, celui qui offre la spécification de l'évolution dynamique la plus complète puisqu'il admet toutes les actions de reconfiguration de base. Cependant les contraintes introduites par le style C2 se révèlent très contraignantes.

La communication entre composants ne peut être structurée qu'en couches superposées où les interactions (concernant chaque type) sont unidirectionnelles. Ceci exclut, par exemple, la description de deux composants s'envoyant mutuellement des requêtes ou des architectures comprenant des cycles.

De plus, le fait que les composants soient contraints à exactement un port *top* et un port *bottom* leur interdit d'être connectés à plusieurs connecteurs ce qui, combiné au fait que les composants ne possèdent qu'un port pour chaque direction, établit une restriction supplémentaire pour la communication entre composants : par exemple, si un composant C est connecté en haut d'un composant C' et si un autre composant C'' est connecté en haut du même composant C', alors C et C'' sont forcément au même niveau et ne peuvent pas, par exemple, s'envoyer des requêtes ou des notifications.

[2] les quatre opérateurs de rajout et de suppression de ports top et bottom ne peuvent être utilisés que pour les connecteurs puisque les composants possèdent exactement un port top et un port bottom et seuls les connecteurs peuvent en posséder plusieurs.




```
component Producteur is
interface
  top_domain is
    out
      null;
    in
      null;
  bottom_domain is
    out
      Message(m:String);
    in
      ready;
parameters
  null;
methods
  function Produire( ) return String;
behavior
  received_message ready;
  invoke Produire( );
  always_generate Message;
end Producteur
```

```
component consommateur is
interface
  top_domain is
    out
      ready( );
    in
      Message(m);
bottom_domain is
    out
      null;
    in
      null;
parameters
  null;
methods
  procedure consommer(m: String)
behavior
  received_messages Message;
  invoke_methodes consommer;
  always_generate ready;
end Producteur
```

```
Architcture ProdCons
conceptual_components
  top_most
    Producteur;
  internal
    null;
  bottom_most
    Consommateur;
connectors
  connector Lien is
    message_filtering no_filtering;
  end Lien
architectural_topology
  connector Lien connections
    top_ports
      Producteur;
    bottom_ports
      Consommateur;
end ProdCons

System ProdConsSys is
  architecture ProdCons with
    Producteur is_bound_to P;
    Consommateur is_bound_to C;
end ProdConsSys
```

FIG. 1.8 – Exemple d'une description C2SADL pour le producteur-consommateur

1.3.2 La description des architectures dynamiques par les graphes

Les graphes présentent l'avantage d'offrir un formalisme simple et facilement assimilable permettant de décrire un large panel de structures. Par ailleurs, la transformation des graphes et les grammaires de graphes constituent des modèles efficaces pour considérer l'aspect dynamique relatif à la transformation et à l'évolution de ces structures. Les travaux théoriques sur ce formalisme, offrent des moyens formels de vérification et de raisonnement sur les propriétés de ces systèmes et la validité des contraintes qu'ils doivent observer.

De manière générale, dans le cas de la description des architectures logicielles orientées-composants, les nœuds décrivent des composants logiciels alors que les arcs correspondent aux différentes interdépendances entre ces composants. Les interdépendances peuvent correspondre, par exemple, à des canaux de communication ou à des liens de contrôle ou de composition.

Une approche de description basée sur les graphes est encore plus pertinente dans le cas des architectures dynamiques. D'une part, les aspects déclaratifs correspondant à la définition de toutes les instances correctes de l'architecture peuvent être rigoureusement décrits par les grammaires de graphes. D'autre part, les aspects opérationnel correspondant à la reconfiguration de l'architecture peuvent être spécifiés par des systèmes à base de transformation de graphes. Les étapes de simulation et de vérification par les outils basés sur les graphes et les grammaires de graphes concernent l'élaboration de preuve pour la validité des propriétés topologiques et fonctionnelles des architectures et l'analyse de la consistance de leur évolution dynamique.

Les travaux de la description des architectures par les graphes

Nous retrouvons dans la littérature différents travaux basés sur les graphes qui traitent des architectures logicielles. Nous pouvons citer principalement les travaux de *Le Metayer* qui constituent probable-

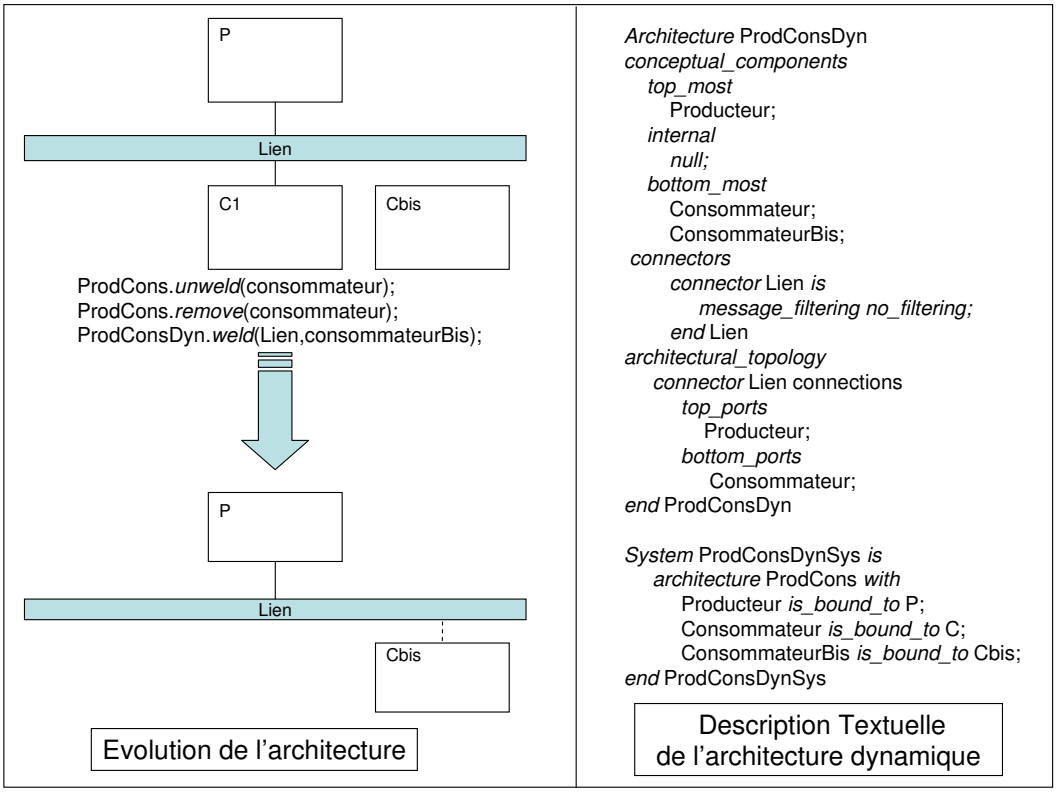


FIG. 1.9 – Exemple d'une description C2SADL pour une architecture producteur-consommateur dynamique

ment les premiers travaux de description basée sur les graphes.

Le modèle décrit dans [Met98] est structuré en deux niveaux, le premier décrit l'architecture sous la forme d'un graphe, et le second décrit les styles architecturaux par une grammaire de graphes. L'évolution de l'architecture est décrite par des règles de transformation de graphes. Le modèle définit une approche formelle basée sur un algorithme de vérification permettant de vérifier à priori et de manière statique la consistance des règles de l'évolution de l'architecture. Cette approche permet de prouver que les contraintes considérées par la description de l'architecture sont préservées par ces règles.

Dans [HIM99], les auteurs reprennent le même modèle pour la description de l'architecture et de son évolution dynamique. Ils décomposent la description de l'architecture en trois parties : 1) la première description spécifie les règles de la construction de la configuration initiale, 2) la deuxième spécifie les règles régissant l'évolution dynamique de l'architecture, et 3) la troisième spécifie les règles régissant la communication.

Ces travaux abordent aussi la problématique de la consistance du point de vue de la communication et des états de composants. Le cycle de vie des composants (comprenant par exemple leur activation et leur désactivation) est simulé en affectant des labels (correspondants à l'état courant des composants) aux nœuds des graphes et des règles de réécriture. La partie décrivant la communication est spécifiée via l'introduction d'événements (en notation CSP) et leur prise en compte en étiquetant les arcs des graphes représentant les connexions entre composants.

[FH00] présente une utilisation de la réécriture de graphe comme un outil de description, d'analyse, et de modification de l'architecture. Il classifie les transformations de graphes en trois catégories : 1) les transformations pour la compréhension. Ces transformations permettent de comprendre et d'explorer la structure de l'architecture passant, par exemple, d'un graphe de l'architecture de haut niveau à un graphe de plus bas niveau (i.e. passage d'un composant composite aux composants primitifs le constituant),

2) les transformations pour l'analyse, utilisées pour extraire des informations sur le système pour la maintenance, et 3) les transformations pour la modification permettent de faire évoluer l'architecture du système pour s'adapter à un nouvel environnement, ou pour le recouvrement d'erreurs.

L'analyse de l'architecture est basée sur des règles de transformation permettant d'obtenir une description de l'architecture à un niveau d'abstraction $n1$ à partir d'une description donnée pour un autre niveau d'abstraction $n2$. La maintenance, qui constitue l'objectif de ces travaux, se décomposera en trois étapes comprenant : 1) le passage d'un haut niveau d'abstraction vers un plus bas niveau pour déterminer les dépendances entre les sous systèmes, 2) l'étape de diagnostique permettant l'identification des dépendances non consistantes au niveau le plus bas, et finalement 3) l'étape de réparation qui consiste à éliminer ses dépendances inconsistantes par transformation de graphes.

1.4 Les grammaires de graphes

Le formalisme de base définit un graphe par l'ensemble de ses nœuds et l'ensemble de ses arêtes. Ces graphes basiques sont décrits par un couple *G=(N,A)* où N est l'ensemble des nœuds constituant le graphe. A est l'ensemble (de cardinalité 2) des arêtes du graphe où chaque arête est définie par un ensemble de deux éléments correspondant aux deux nœuds qu'elle relie.

D'autres types de graphes permettent de répondre à des systèmes nécessitant des spécifications plus riches. Nous retrouvons, pour prendre en compte des relations asymétriques, les graphes orientés où les arcs[3] possèdent un nœud de départ et un nœud d'arrivée. Dans ce cas les arcs sont spécifiés par un couple[4] de nœuds où le premier élément décrit le nœud de départ et le second élément correspond au nœud d'arrivée.

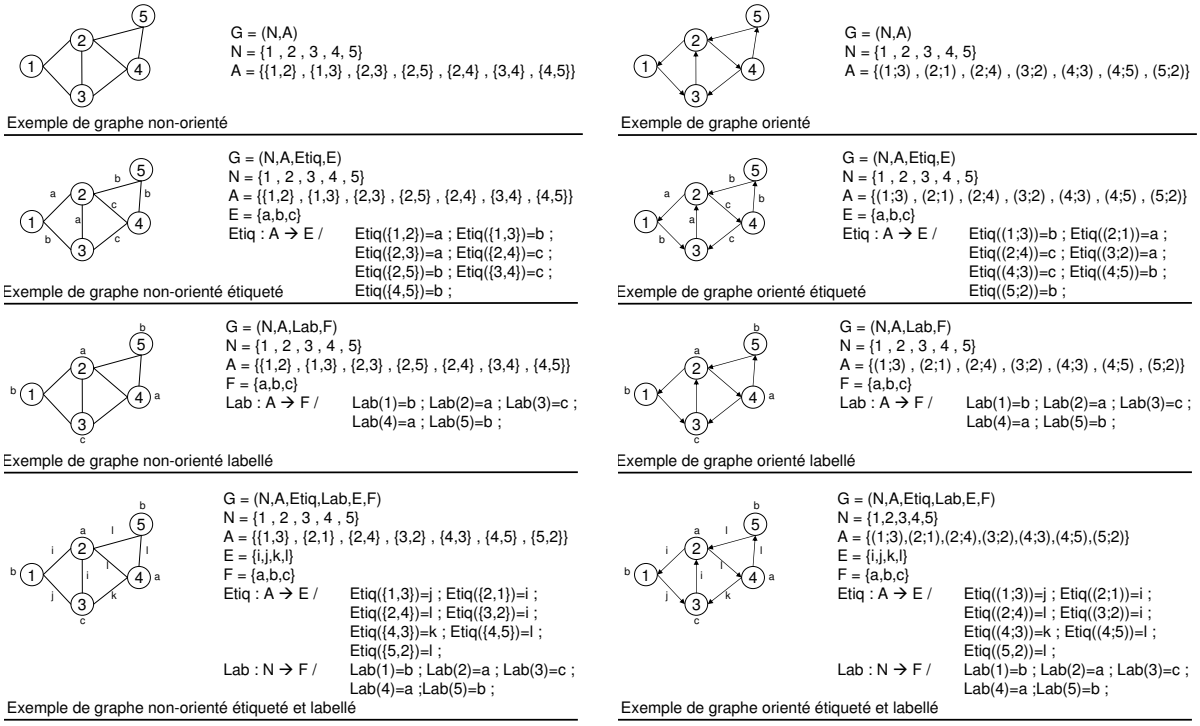


FIG. 1.10 – Les différents types de graphes de base

Dans le cas où le système comprend plusieurs types de relations entre les nœuds, on introduit une fonction de valuation qui, à chaque arc ou arête, associe un poids ou une valeur. Le graphe est défini par

[3] dans ce cas on utilise le terme arc plutôt que le terme arête.

[4] et non plus par un ensemble de nœuds de cardinalité 2

un quadruplet *G=(N,A,Etiq,E)* où N et A gardent la même sémantique que dans les cas précédents. E est l'ensemble des étiquettes possibles des arcs ou des arêtes et *Etiq*: $A \rightarrow E$ est la fonction permettant de donner les valuations des arcs.

Les valuations peuvent aussi concerner les nœuds permettant de leur associer des paramètres. On introduit, selon la même approche que pour les valuations des arcs, une fonction de valuation permettant d'associer, à chaque nœud, un label[5]. Ces graphes sont décrits par un quadruplet *G=(N,A,Lab,F)* où N et A correspondent respectivement à l'ensemble des nœuds et des arcs du graphe. F correspond à l'ensemble des labels possibles des nœuds et *Lab*: $N \rightarrow F$ est la fonction permettant d'associer à chaque nœud son label.

La figure 1.10 donne un exemple des différents types de graphes de base en considérant les différentes combinaisons selon qu'on considère des graphes orientés ou non, étiquetés ou non, et labellés ou non.

1.4.1 La transformation de graphes

La réécriture de graphes [Roz97, ER97, EK90] est un mécanisme pour transformer des graphes d'une manière mathématique rigoureuse. Ce formalisme a été introduit à la fin des années 60 pour traiter des problématiques telles que la construction de compilateurs, ou la spécification des types de données. Actuellement, ces techniques constituent la base formelle pour la résolution de problèmes relatifs à des domaines d'application tels que l'identification faciale [WFKM97, KTP00], la reconnaissance d'objets [Bun00, NL91], la reconnaissance de symboles [LS03], et l'identification de caractères et la graphologie [LRS91].

Dans la littérature, on retrouve principalement deux aspects de la réécriture de graphes : les grammaires de graphes et la transformation de graphes. Les mécanismes et les approches de réécriture sont identiques dans les deux cas mais la différence se situe dans leur finalité. On parle généralement de productions de grammaire dans le premier cas et de règles de transformation dans le second. Pour préciser la différence entre les deux, on peut dire que les grammaires de graphes s'inspirent et gardent la même logique que les grammaires génératives("*generative grammars*") de Chomsky [Cho56]. L'intérêt porte sur l'ensemble des graphes produits suite à l'application d'un ensemble de productions à partir d'un graphe de départ donné. Si on s'intéresse aux transformations elles-mêmes comme processus de calcul, nous parlons alors de transformation de graphes.

On fera le distinguo, dans le cas des grammaires de graphes, entre les nœuds terminaux et les non terminaux. Les applications successives des productions de grammaire permettent le passage d'un graphe intermédiaire (contenant des non terminaux) vers un autre graphe intermédiaire (contenant encore des non terminaux) jusqu'à atteindre un graphe final (ne contenant plus de non terminaux).

Dans le cas de la transformation de graphes, les nœuds sont tous du même type (tous des terminaux) et l'application d'une règle de transformation a pour but le passage d'un graphe représentant par exemple l'état courant d'un système vers un autre graphe représentant son nouvel état.

Nous proposons, dans ce qui suit, une présentation des différentes approches de réécriture qui peuvent donc être spécialisées pour la spécification des productions de grammaires et des règles de transformation avec les nuances que nous venons de décrire. Nous parlerons dans ce qui suit de règles de réécriture pour présenter les différentes approches pour les règles de transformation et les productions de grammaire.

Problème des arcs suspendus

Le moyen le plus basique de réécrire un graphe G en un graphe G' est de remplacer un sous graphe L de G par un graphe R. G' est le graphe résultant de ces deux opérations. G est appelé *graphe hôte* ("*host graph*"), L est appelé *graphe mère* ("*mother graph*") et R est appelé graphe fille ("*daughter graph*").

Dans cet esprit, une règle de réécriture est décrite dans le modèle de base par une paire de graphes $(L; R)$. Ce type de règle est applicable à un graphe G s'il existe une occurrence de L dans G. Son application a pour conséquence la suppression de cette occurrence de L du graphe G et son remplacement par une copie (isomorphe) de R.

[5]Pour faire le distinguo entre les deux cas précités, nous parlerons (dans ce manuscrit) de graphe étiqueté (et donc d'étiquettes) pour le cas où les valuations concernent les arcs, et de graphe labellé (et donc de labels) dans le cas où ces valuations concernent les nœuds

Ce type de définition pose le problème des arcs suspendus ("*dangling edges*"). La suppression de l'occurrence de L dans G, peut entraîner l'apparition d'arcs sans nœud de départ ou sans nœud d'arrivée ou sans les deux. Un exemple de ce cas de figure est présenté dans la figure 1.11. Pour résoudre cette problématique, deux approches principales sont utilisées, avec pour chacune, des choix différents concernant la caractérisation des règles de réécriture et le traitement du cas des arcs suspendus [EKL90].

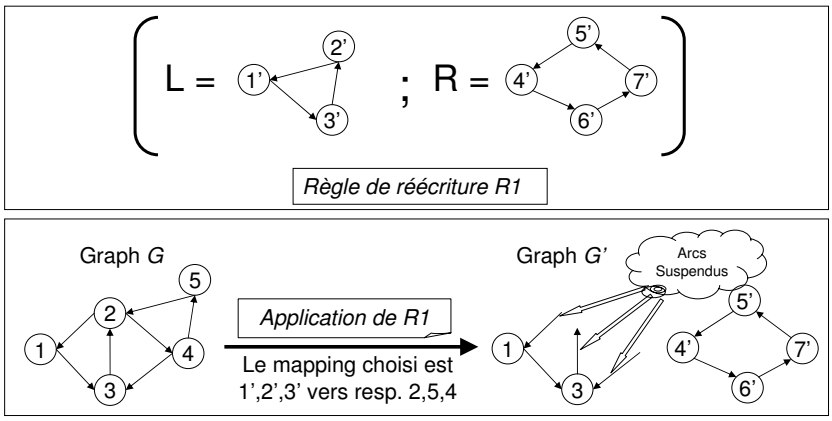


FIG. 1.11 – Application d'une règle de réécriture impliquant l'apparition d'arcs suspendus

L'approche SPO

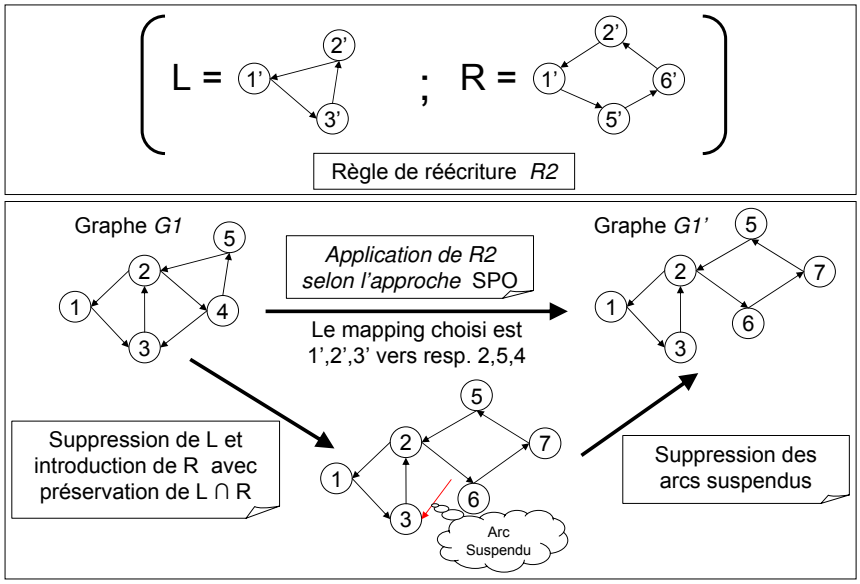


FIG. 1.12 – Application d'une règle de réécriture selon l'approche SPO

Une des approches les plus classiques est l'approche *SPO* (Single PushOut). Une règle de réécriture est spécifiée par une paire de graphes *(L ;R)*. Son application à un graphe G est subordonnée à l'existence d'une occurrence de L dans G. La différence, par rapport à l'approche de base présentée précédemment, est

que des parties du graphe L (des nœuds et des arcs appertenant à L) seront préservés après l'application de la règle. Ces parties sont spécifiées en les faisant apparaître dans les deux graphes L et R. L'application de la règle implique la suppression du graphe correspondant à $Del = (L \setminus (L \cap R))$ et le rajout du graphe correspondant à $Add = (R \setminus (L \cap R))$. Selon l'approche SPO, les arcs suspendus sont supprimés.

Un exemple de réécriture de graphe est donné dans la figure 1.12. L'application de la règle de réécriture est décomposée en deux étapes concernant : 1) la mise à jour du graphe en supprimant l'occurrence du graphe Del et en introduisant une copie du graphe Add, et 2) la suppression de l'unique arc suspendu (qui reliait le nœud 4 au nœud 3) dont l'apparition est due à la suppression du nœud 4.

L'approche DPO

Dans l'autre approche classique appelée DPO (Double PushOut), la règle est de la forme (L, K, R), où K permet de spécifier clairement la partie à préserver après l'application de la règle (au lieu de la déduire par l'opération $L \cap R$). Une règle de type DPO est applicable à un graphe G s'il y'a une occurrence de L dans G. Une différence capitale avec l'approche SPO est que l'application de cette règle est subordonnée à une condition supplémentaire appelée la condition de suspension ("*Dangling Condition*"). Cette condition stipule que la règle ne peut être appliquée que si son application ne va pas entraîner l'apparition d'arcs suspendus. Si les deux conditions d'existence de l'occurrence et d'absence d'arcs suspendus sont réunies, l'application de la règle implique la suppression du graphe correspondant à l'occurrence de $Del = (L \setminus K)$ et le rajout d'une copie du graphe $Add = (R \setminus K)$.

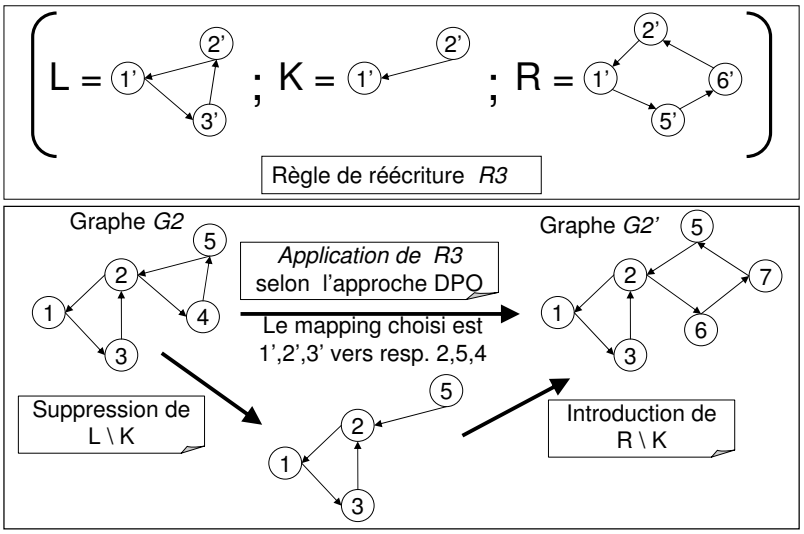


FIG. 1.13 – Application d'une règle de réécriture selon l'approche DPO

Un exemple de l'application d'une règle de type DPO est donné dans la figure 1.13. On notera que le graphe hôte de cet exemple (noté G2) est un peu différent de celui de l'exemple donné pour l'approche SPO (noté G1). La différence réside dans le fait que le graphe G2 ne contient plus l'arc reliant les nœuds 4 et 3 dans le graphe G1. Ceci est dû au fait que si on maintient cet arc dans G2, la règle R3 ne serait plus applicable parce que son application violerait la condition de suspension.

Les conditions d'application négatives

Pour déterminer l'applicabilité d'une règle de transformation, les approches présentées précédemment se basent, entre autres, sur l'existence d'une occurrence du sous graphe L. Dans certains cas, il est nécessaire de pouvoir exprimer des conditions supplémentaires permettant de spécifier des conditions

relatives à l'absence d'une occurrence dans le graphe hôte. Ce type de conditions est appelé restrictions ou conditions d'application négatives (Negative Application Conditions ou NACs).

L'intégration de conditions de type NAC ajoute donc une nouvelle zone dans la structure des règles de réécriture. Une règle SPO aura la structure *(L ;R ;NAC)* et sera applicable à un graphe G s'il y'a une occurrence de L dans G et s'il n'y a pas d'occurrence de NAC dans G. Dans le même esprit, une règle selon l'approche DPO aura la structure *(L ;K ;R ;NAC)* et sera applicable s'il y a occurrence de L et absence d'occurrences de NAC. Dans tous les cas, le motif NAC n'intervient qu'au niveau de l'applicabilité de la règle et non dans l'étape de transformation qui reste la même que dans le cas sans NAC (i.e. suppression du motif *Del* et introduction du motif *Add*).

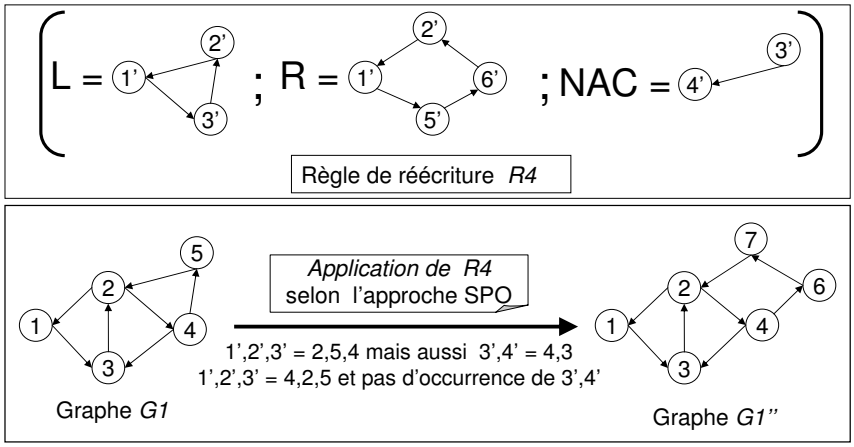


FIG. 1.14 – Application d'une règle de réécriture selon l'approche SPO et avec des conditions d'application négatives

La figure 1.14 présente un exemple de l'application d'une règle SPO avec NAC. Nous considérons le graphe de l'exemple donné dans le cas SPO basique (figure 1.12) et la règle de ce même exemple (i.e. $R2$) contrainte par une condition de type NAC. Cette condition spécifie que le nœud unifié avec le nœud $N3'$ ne doit pas être connecté avec un autre nœud (qui pourrait être unifié avec $N4'$). La règle $R4$ résultante n'est donc plus applicable sur G1 avec la même occurrence de la partie L (i.e. le sous graphe 2,5,4 occurrence du graphe 1',2',3') car il y a, dans ce cas, une occurrence de la partie NAC dans *G1* (i.e. le sous graphe 4,3 est une occurrence du sous graphe 3',4'). Par contre, cette règle est applicable si on prend en compte le sous graphe constitué par les nœuds 4,2,5 comme occurrence du graphe 1',2',3'. Dans ce cas, il n'existe aucune occurrence de 3',4' dans le graphe $G1$ (sachant que 3' est unifié avec 5 cela revient à dire qu'il n'y a aucun nœud fils du nœud 5). La règle $R4$ est donc applicable et transforme le graphe $G1$ pour obtenir le graphe $G1$".

1.4.2 Approches basées sur les instructions de connexion

Dans le cadre des approches SPO et DPO, la gestion de la connexion du graphe fille au reste du graphe hôte est traitée du point de vue de la gestion des arcs suspendus. Dans ce cas, les grammaires de graphes sont définies, de la même manière que les grammaires de *Chomsky*, par le quatruplet $(AX; NT; T; P)$ où AX est l'axiome, NT l'ensemble des nœuds non terminaux, T l'ensemble des nœuds terminaux, et P l'ensemble des productions de la grammaire (par exemple de type SPO ou DPO). Un graphe appartient à la grammaire s'il contient exclusivement des nœuds terminaux et s'il peut être obtenu en partant de l'axiome et en appliquant, dans n'importe quel ordre et autant de fois que nécessaire, une séquence de productions appartenant à P.

Dans cette section, nous aborderons des mécanismes et des formalisations de grammaires de graphes plus sophistiqués permettant une spécification plus riche des liens à introduire entre les nœuds du graphe

fille et les nœuds voisins du graphe mère. L'apport principal de ces approches concerne l'introduction de ce qu'on appelle les instructions de connexion.

Le mécanisme NLC (Node Label Controlled mechanism)

NLC est un mécanisme simple permettant la connexion du graphe fille au graphe hôte. Ce mécanisme a été introduit dans le cadre des graphes non orientés à nœuds labelés. Il concerne un type particulier de productions de grammaires appelé remplacement de nœuds (node-replacement). Dans le cadre de cette approche, les graphes mères des productions sont constitués d'un nœud non terminal unique. Des instructions de connexion sont introduites pour permettre la connexion du graphe fille aux nœuds voisins du graphe mère. Ces instructions se basent sur les labels des nœuds pour définir les arêtes supplémentaires à introduire.

Une production NLC est de la forme $X \rightarrow D$ où X est un label d'un nœud non terminal, et D est un graphe labelé non orienté constitué de nœuds terminaux et non terminaux. L'application d'une telle production implique, dans un premier lieu, la suppression d'un non terminal ayant X comme label et son remplacement par le graphe D.

Les instructions de connexion sont de la forme (μ, δ) où μ et δ sont deux labels de nœuds terminaux ou non terminaux et impliquent, après l'étape de transformation décrite précédemment, l'introduction d'une arête entre chaque nœud du graphe fille ayant le label μ et chaque nœud voisin du nœud constituant le graphe mère et qui a comme label δ [6].

Une grammaire NLC est donc décrite par un quintuplet (AX, NT, T, P, C) où AX, NT, T, P représentent respectivement les champs classiques : l'axiome, l'ensemble des non terminaux, l'ensemble des terminaux et l'ensemble des productions de grammaire (de type NLC). C contient l'ensemble des instructions de connexion de la grammaire. Ces instructions sont communes à toutes les productions de grammaire : après chaque application d'une des productions appartenant à P, toutes les instructions de connexion de C qui sont applicables seront exécutées.

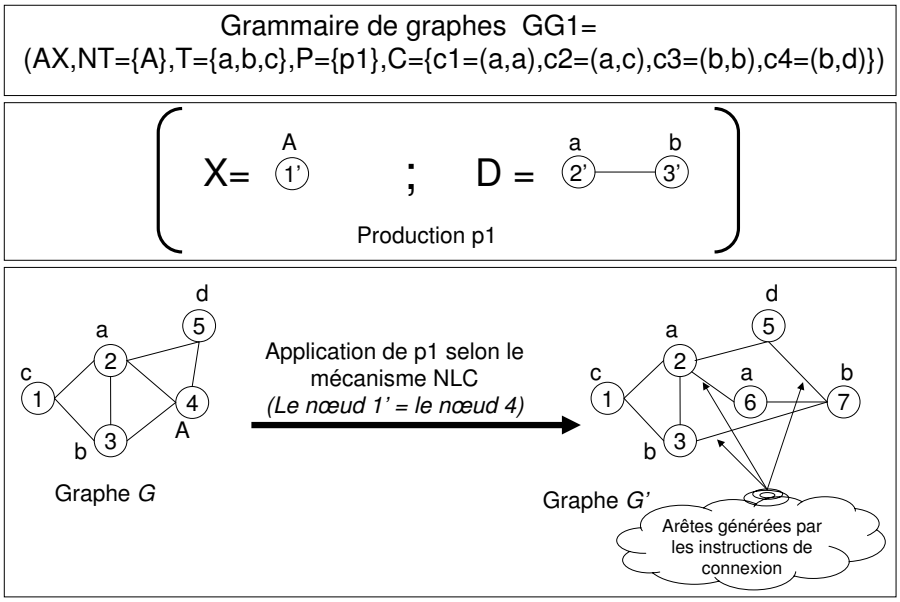


FIG. 1.15 – Exemple d'une grammaire qui utilise le mécanisme NLC

La figure 1.15 donne un exemple de grammaire NLC et de transformation de graphes en utilisant ce mécanisme. Le nœud 4 ayant le label non terminal A est remplacé par une copie du graphe D (i.e. les

[6] Notons que la présence de nœuds voisins du nœud appartenant au graphe mère avec un label δ n'est pas une condition de l'applicabilité de la production, et que dans le cas où ce type de nœuds est absent, l'instruction de connexion est tout simplement ignorée.

nœuds 6 et 7 et l'arête qui les relie). L'application de l'instruction *c1* a pour conséquence l'introduction d'une arête entre les nœuds 6 et 2, *c3* implique l'introduction d'une arête entre les nœuds 7 et 3 tandis que *c4* introduit l'arête entre 7 et 5. L'instruction de connexion *c2* est ignorée puisqu'aucun des voisins du nœud 4 n'a comme label 'c'.

Extensions et variations de l'approche NLC

L'approche NLC peut être étendue pour permettre de traiter d'autres types de graphes et d'apporter plus de puissance d'expression aux productions de grammaires. Nous présenterons, dans ce qui suit, quelques extensions permettant de prendre en compte des graphes orientés et offrant des spécifications plus fines des instructions de connexion.

dNLC

Dans le cadre de l'extension appelée dNLC (*d pour directed*), chaque instruction de connexion est décrite par un triplet (μ, δ, d) où $d \in \{in, out\}$ permet de prendre en compte le sens des arcs.

Ainsi, une instruction (μ, δ, in) implique l'introduction d'un arc entre tous les nœuds $n1$ appartenant au graphe fille et ayant comme label μ et tous les nœuds $n2$ qui sont des voisins entrants (*in-neighbours*) du graphe mère (les voisins entrants d'un nœud n sont tous les nœuds n' tels qu'il existe un arc allant de n' vers n) et qui ont comme label δ.

De la même manière, une instruction (μ, δ, out) implique l'introduction d'un arc entre tous les nœuds $n1$ appartenant au graphe fille et ayant comme label μ et tous les nœuds $n2$ qui sont des voisins sortants (*out-neighbours*) du graphe mère (les voisins sortants d'un nœud n sont tous les nœuds n' tels qu'il existe un arc allant de n vers n') et qui ont comme label δ.

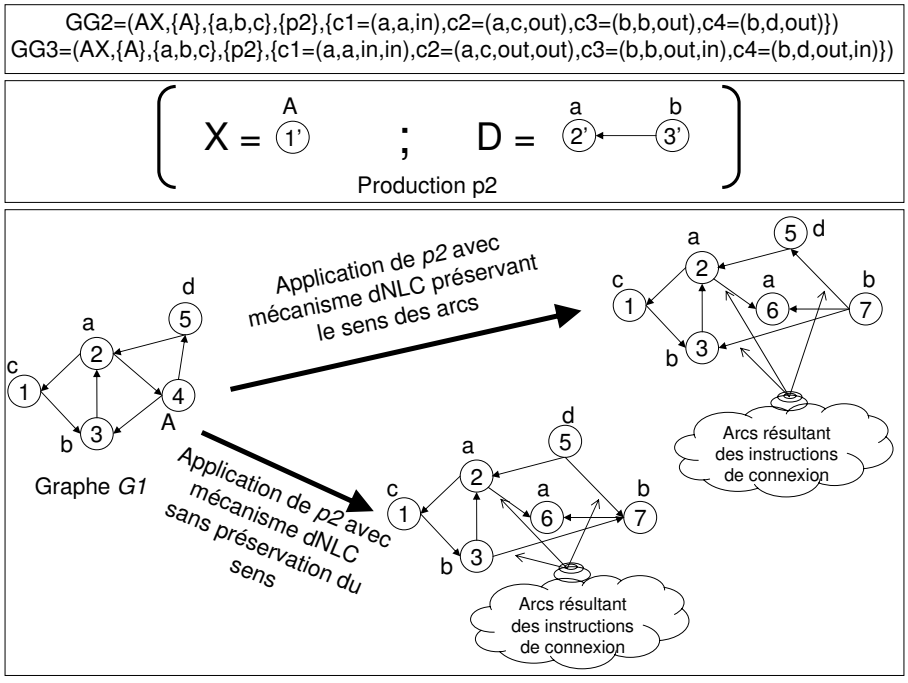


FIG. 1.16 – Mécanisme dNLC

Cette approche préserve la sens des arcs de manière à ce que les arcs résultant des instructions de type *out* produisent des arcs allant des nœuds du graphe fille vers les voisins sortants, et les instructions de type *in* produisent des arcs allant des voisins entrants vers les nœud du graphe fille.

Une autre approche permet de modifier le sens des arcs. Dans ce cas, les instructions sont spécifiées par un quadruplet (μ, δ, d, d') où μ, δ, et d gardent la même signification que dans l'approche précédente tandis que d' indique le sens de l'arc à introduire.

Ainsi, l'instruction (μ, δ, in, out) (respectivement (μ, δ, out, in)) implique l'introduction d'un arc entre tous les nœuds $n1$ appartenant au graphe fille et ayant comme label μ et tous les nœuds $n2$ qui sont des voisins entrants (respectivement voisins sortants) du graphe mère et qui ont comme label δ. Le sens de l'arc est cette fois-ci de $n1$ vers $n2$ (respectivement $n2$ vers $n1$) [7].

Un exemple de grammaire de graphes avec des productions de type dNLC sans et avec préservation du sens des arcs est donné dans la figure 1.16. On notera que le sens de l'arc reliant les nœuds 3 et 7 et celui reliant les nœuds 5 et 7 diffèrent selon l'approche choisie.

eNLC

On retrouve, dans la littérature, une autre extension de l'approche NLC dédiée au traitement des graphes possédant des arêtes étiquetées. Cette approche appelée eNLC (e pour *edge label*) permet, selon un mécanisme calqué sur celui de l'approche NLC, la connexion des nœuds du graphe fille aux voisins du nœud constituant le graphe mère. Il permet, en plus, de tenir compte des étiquettes des arêtes concernées et de leur mise-à-jour.

Les instructions de connexion sont de la forme $(\mu, p/q, \delta)$ où μ et δ sont des labels de nœuds qui désignent les mêmes concepts que dans le cas de l'approche NLC tandis que p et q sont des étiquettes d'arêtes. L'exécution de ce type d'instructions implique l'introduction d'une arête avec le label q entre tous les nœuds qui sont *p-voisins*[8] du nœud constituant le graphe mère et qui ont comme label δ.

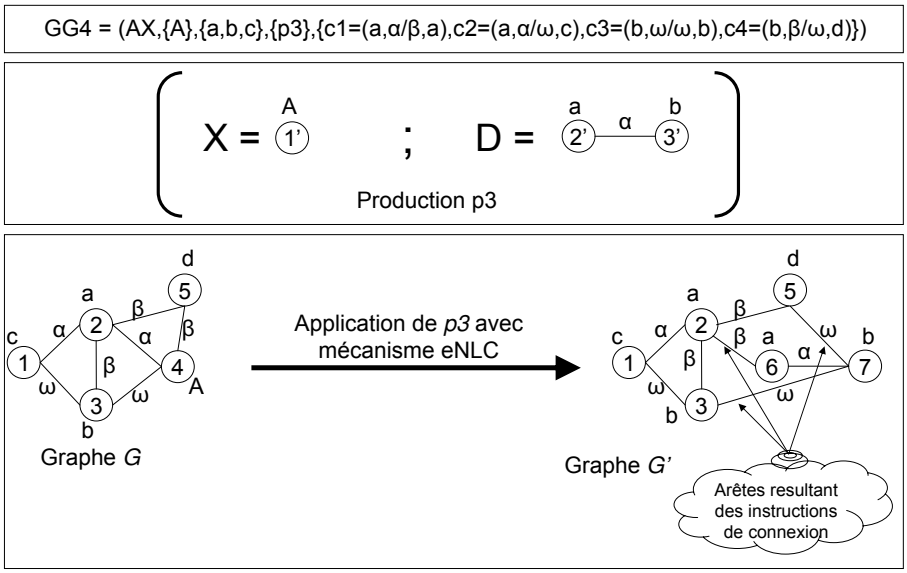


FIG. 1.17 – Exemple de l'application d'une production eNLC

Un exemple de l'application de productions eNLC est donné dans la figure 1.17. Les labels des arêtes résultant de l'exécution des instructions de connexion sont mis-à-jour selon la spécification donnée. Ainsi, le label de l'arête reliant les nœuds 6 et 2 passe de α à β, celui de l'arête reliant les nœuds 7 et 5 passe de β à ω tandis que le label reliant les nœuds 7 et 3 reste à ω comme stipulé dans l'instruction de connexion relative au cas ou les deux nœuds ont le même label b.

[7] Les instructions (μ, δ, in, in) et (μ, δ, out, out) sont respectivement équivalentes aux instructions (μ, δ, in) et (μ, δ, out) qui préservent le sens des arcs.

[8] Les nœuds p-voisins d'un nœud n sont tous les nœuds n' qui lui sont reliés avec des arêtes labelées par p.

Une combinaison des deux approches eNLC et dNLC produit l'approche edNLC. Dans cette approche, on prend en compte les deux mécanismes introduits par les grammaires eNLC et dNLC. Les instructions de connexion sont, dans le cas le plus général, de la forme $(\mu, p/q, \delta, d, d')$. Leur exécution implique l'introduction d'un arc dans le sens indiqué par d' entre tous les nœuds $n1$ du graphe fille qui ont comme label μ et entre tous les nœuds $n2$ qui sont p-voisins du nœud constituant le graphe mère tel que l'arc qui relie $n1$ et $n2$ est dans le sens d.

L'approche NCE

Un des défauts associés à l'approche NLC est le fait qu'on ne puisse faire la distinction entre les nœuds que par leurs labels. L'approche NCE (Neighbourhood Controlled Embedding) traite ce problème et permet de décrire des instructions de connexion en se référant directement aux nœuds du graphe fille au lieu de se référer à leurs labels.

Les instructions NCE sont de la forme (n, δ) où δ est un label de nœud terminal ou non et où n identifie un nœud du graphe fille. L'exécution de cette instruction de connexion implique l'introduction d'une arête entre le nœud n et tous les nœuds voisins du graphe mère et dont le label est δ. Une grammaire de graphes NCE est donc définie par le quadruplet (AX, NT, T, P) où AX, NT, et T gardent la même signification que dans les grammaires NLC. P Spécifie l'ensemble des productions de la grammaire qui sont de la forme $((X, D), C)$ où (X, D) est une production NLC et où C est un ensemble d'instructions de connexion impliquant les nœuds appartenant à D.

Il est à noter, par rapport à l'approche NLC, que les instructions de connexion ne sont plus globales (i.e. appliquées à toutes les productions de la grammaire), mais que chaque production possède des instructions de connexion qui lui sont propres.

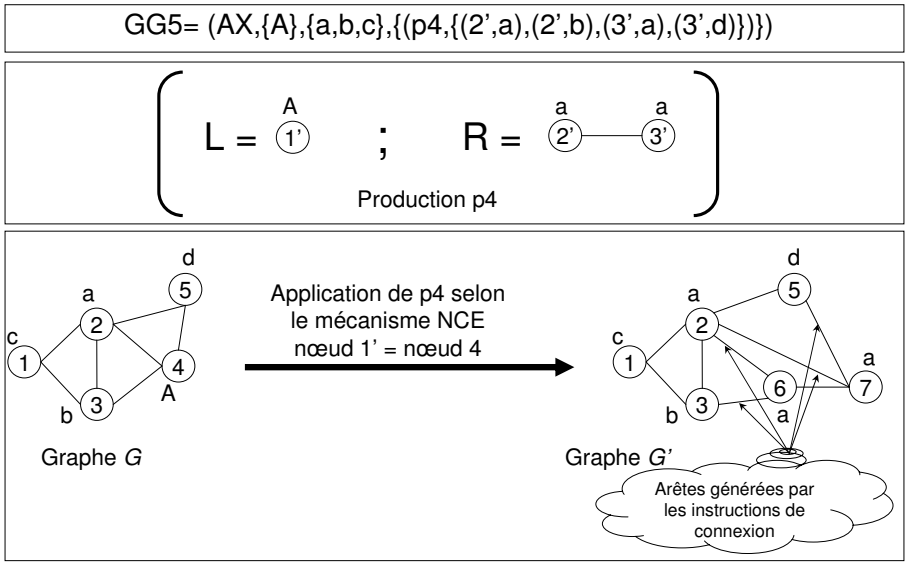


FIG. 1.18 – Exemple de l'application d'une production NCE

Un exemple de l'approche NCE est donné dans la figure 1.18. Dans cet exemple, l'approche NCE permet de connecter différemment les nœuds 6 et 7 (introduits par la copie du graphe fille D) aux voisins du nœud 4 (le nœud 6 est connecté aux nœuds 2 et 3 alors que le nœud 7 est connecté au nœud 5). Ceci n'aurait pas été possible en adoptant l'approche NLC puisque les nœuds 2' et 3' ont le même label a.

Les mêmes extensions eNLC, dNLC, et edNLC peuvent être appliquées pour obtenir les approches eNCE, dNCE, et edNCE. Dans le cas le plus général, on obtient pour les grammaires edNCE des productions de grammaire de la forme $((X, D), C)$ telles que C est un ensemble contenant des instructions de

connexion de la forme $(n, p/q, \delta, d, d')$. L'exécution des instructions de connexion edNCE implique l'introduction d'un arc dans le sens indiqué par d' entre le nœud n qui appartient au graphe fille et entre tous les nœuds n' qui sont p-voisins du nœud constituant le graphe mère. L'arc qui relie n et n' est introduit dans le sens d et possède q comme étiquette. Un exemple d'une grammaire edNCE est donné dans la figure 1.19.

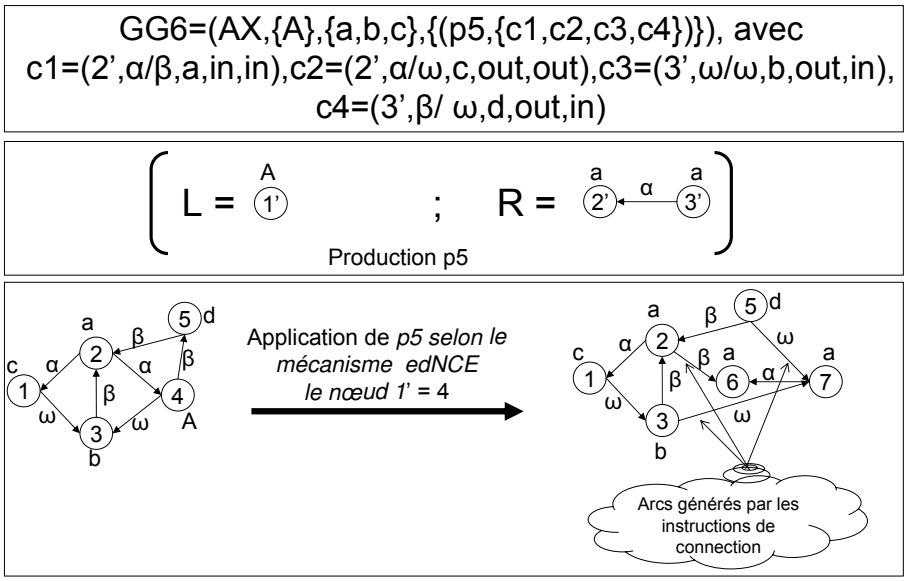


FIG. 1.19 – Mécanisme edNCE

1.5 Les algorithmes de recherche d'homomorphismes de graphes

La vérification de l'existence du graphe mère dans le graphe hôte est traduite mathématiquement par l'existence d'un homomorphisme entre ces deux graphes. Cette problématique est, donc, centrale dans toutes les approches impliquant des grammaires de graphes ou plus généralement la réécriture de graphes.

Nous présentons dans cette section, en premier lieu, les bases mathématiques pour la spécification des isomorphismes et homomorphismes de graphes. Ensuite, nous présentons les algorithmes les plus performants qui implémentent la recherche des homomorphismes et étudions leur complexité.

Deux graphes sont isomorphes s'il existe une correspondance bijective entre leurs ensembles de nœuds qui préserve la structure. C'est-à-dire que si deux nœuds d'un graphe sont reliés par une arête alors les deux nœuds qui leur correspondent par cette bijection sont aussi reliés par une arête. On obtient, donc, la définition formelle qui suit.

***Définition* 1 (Isomorphisme de graphes)** *Deux graphes $G1 = (N1, A1)$ et $G2 = (N2, A2)$ sont isomorphes s'il existe une bijection $F \subseteq N1 \times N2$ telle que pour chaque paire $n_i, n_j \in N1$ et $m_i, m_j \in N2$ avec $F(n_i) = m_i$ et $F(n_j) = m_j$, alors l'arête $\{n_i, n_j\} \in A1$, si et seulement si l'arête $\{m_i, m_j\} \in A2$. F est, dans ce cas, un isomorphisme de graphes entre $G1$ et $G2$.*

Une généralisation des isomorphismes de graphes est le concept d'homomorphisme de graphes appelé aussi isomorphisme de sous graphes (*subgraph isomorphism*). La problématique concerne le fait de déterminer si un graphe est isomorphe à un sous graphe d'un autre graphe. On parlera, dans ce qui suit, de morphismes pour désigner le cas général englobant les isomorphismes et les homomorphismes [9].

[9] La notion de morphismes est en fait plus large et englobe d'autre concepts tels que les monomorphismes, les epimorphismes, les endomorphimsmes, les automorphismes, etc... auxquels nous ne faisons pas référence dans ce manuscrit.

***Définition* 2 (Homomorphisme de graphes)** *Un homomorphisme d'un graphe* $G1 = (N1, A1)$ *vers un graphe* $G2 = (N2, A2)$ *est une injection* $F \subseteq N1 \times N2$ *telle que pour chaque paire* $n_i, n_j \in N1$*, si* $\{n_i, n_j\} \in A1$*, alors* $\{F(n_i), F(n_j)\} \in A2$*. F est, dans ce cas, un homomorphisme de graphes de* $G1$ *vers* $G2$*.*

Les définitions précédentes sont données pour le cas de base impliquant des graphes non orientés et non étiquetés. La prise en compte de graphes étiquetés ou labelés implique des contraintes supplémentaires pour la détection d'isomorphismes ou d'homomorphismes.

La préservation de la structure qui impliquait, dans le cas non orienté, l'existence d'une arête entre chaque paire de nœuds dans le graphe hôte dans le cas ou les deux correspondants dans le graphe mère étaient connectés par une arête, implique, dans le cas orienté, l'existence d'un arc entre deux nœuds n_{h1} et n_{h2} du graphe hôte si les deux nœuds n_{m1} et n_{m2} qui leur correspondent respectivement dans le graphe mère sont reliés par un arc. Les deux arcs doivent en plus respecter le sens. Ceci signifie que si l'arc est dans le sens n_{m1} vers n_{m2} alors on doit vérifier l'existence d'un arc de n_{h1} vers n_{h2} et vice-versa. On retrouve les deux définitions suivantes pour le cas orienté.

***Définition* 3 (Isomorphisme de graphes orientés)** *Deux graphes* $G1 = (N1, A1)$ *et* $G2 = (N2, A2)$ *sont isomorphes s'il existe une bijection* $F \subseteq N1 \times N2$ *telle que pour chaque paire* $n_i, n_j \in N1$ *et* $m_i, m_j \in N2$ *avec* $F(n_i) = m_i$ *et* $F(n_j) = m_j$*, l'arc* $(n_i, n_j) \in A1$*, si et seulement si l'arc* $(m_i, m_j) \in A2$*.*

***Définition* 4 (Homomorphisme de graphes orientés)** *Un homomorphisme de* $G1 = (N1, A1)$ *vers* $G2 = (N2, A2)$ *est une injection* $F \subseteq N1 \times N2$ *telle que pour chaque paire* $n_i, n_j \in N1$*, si* $(n_i, n_j) \in A1$*, alors* $(F(n_i), F(n_j)) \in A2$*.*

Dans le cas des graphes avec des nœuds labelés ou des arcs étiquetés, la contrainte sur la préservation de la structure reste nécessaire mais une contrainte supplémentaire aux niveaux des labels est aussi considérée. Dans le cas des graphes avec nœuds labelés, La fonction F (bijective dans le cas des isomorphismes et injective dans le cas des homomorphismes) est construite de telle manière que deux nœuds n_m et n_h appartenant respectivement au graphe mère et au graphe hôte avec $F(n_m) = n_h$ doivent avoir le même label.

***Définition* 5 (Morphisme de graphes orientés avec des nœuds labelés)** *Un isomorphisme (respectivement, un homomorphisme) d'un graphe* $G1 = (N1, A1, Lab1)$ *vers un graphe* $G2 = (N2, A2, Lab2)$ *est une bijection (respectivement, une injection)* $F \subseteq N1 \times N2$ *qui vérifie :*
1- pour chaque nœud $n \in N1$ *et chaque nœud* $m \in N2$ *tel que* $F(n_i) = m_i$ *alors* $Lab(n) = Lab(m)$*, et*
2- pour chaque paire $n_i, n_j \in N1$ *et* $m_i, m_j \in N2$ *avec* $F(n_i) = m_i$ *et* $F(n_j) = m_j$*, alors l'arc* $(m_i, m_j) \in A2$*, si et seulement si (respectivement, si) l'arc* $(n_i, n_j) \in A1$*.*

Dans le cas d'un graphe étiqueté, la contrainte supplémentaire implique, en plus de la préservation de la structure, la préservation des étiquettes des arêtes ou des arcs. C'est-à-dire que l'étiquette d'un arc reliant deux nœuds du graphe mère doit être égale à l'étiquette de l'arc reliant les deux nœuds correspondants dans le graphe hôte. Cela donne la définition suivante.

***Définition* 6 (Morphisme de graphes orientés avec arcs labelés)** *Un isomorphisme (repectivement, un homomorphisme) d'un graphe* $G1 = (N1, A1, Etiq1)$ *vers un graphe* $G2 = (N2, A2, Etiq2)$ *est une bijection (respectivement, une injection)* $F \subseteq N1 \times N2$ *qui vérifie pour chaque paire* $n_i, n_j \in N1$ *et* $m_i, m_j \in N2$ *avec* $F(n_i) = m_i$ *et* $F(n_j) = m_j$ *:*
1- l'arc $(m_i, m_j) \in A2$*, si et seulement si (respectivement, si) l'arc* $(n_i, n_j) \in A1$*, et*
2- $Etiq((m_i, m_j)) = Etiq((n_i, n_j))$*.*

La définition de morphismes pour les différents types de graphes (non orientés avec nœuds labelés, non orientés avec arcs étiquetés, avec nœuds labelés et arcs étiquetés etc ...) sont facilement déductibles en combinant les définitions précédentes. Mais, dans tous les cas, si on considère des graphes quelconques qui n'impliquent des restrictions sur aucun de paramètres du graphe, le problème de savoir si deux graphes

sont homomorphes est un problème NP-Complet[10] tandis qu'il est généralement admis dans la littérature que le problème de savoir si deux graphes sont isomorphes est un des rare problèmes NP[11] qui ne sont ni dans la classe P[12] ni dans la classe NP-complet [13].

Nous allons présenter dans ce qui suit les algorithmes les plus connus permettant la recherche d'homomorphismes entre les graphes. On commencera par introduire la technique générale de retour en arrière (*backtracking*) qui constitue l'approche de base pour de la majorité algorithmes qu'on trouve dans la littérature. Ensuite, nous présenterons l'algorithme d'*Ullmann* qui est la référence des algorithmes de recherche d'homomorphismes dans les graphes. Puis, nous présentons une approche basée sur une recherche des homomorphismes en largeur d'abord. Nous présenterons, en dernier lieu, l'algorithme de détection de cliques (*clique detection*) qui se base sur la transformation du problème de recherche d'homomorphismes entre deux graphes vers un problème de détection des cliques maximales dans un graphe associé.

1.5.1 Technique de retour en arrière (*backtracking*)

La technique classique de retour en arrière a été présentée pour la première fois dans un algorithme en 1957 par Ray et Kirsch [RK57] puis généralisée plus tard dans [GB65, BR75]. Cette technique se base sur une organisation d'une recherche exhaustive en profondeur d'abord sur l'arbre de recherche. L'objectif est de trouver une ou toutes les solutions d'un problème combinatoire en affectant des valeurs de manière consistante à un ensemble fini de variables.

L'approche consiste à partir d'une solution partielle puis l'étendre par étapes jusqu'à atteindre une solution complète. Chaque étape d'extension de la solution partielle intègre le traitement d'une variable supplémentaire. Si une solution partielle ne peut être étendue, on effectue un retour en arrière.

Cette technique est, par conséquent, destinée aux problèmes qui sont caractérisés par le principe *domino* (ce qui est le cas du problème de la recherche des homomorphismes). Ce principe stipule que si une solution partielle ne vérifie pas une propriété, alors toutes ses extensions ne vérifieront pas cette propriété. Ceci permet d'arrêter l'extension d'une solution partielle dès lors qu'on établit qu'elle viole une propriété que la solution doit vérifier.

Dans le cas des homomorphismes de graphes, le principe domino concerne le fait que si un graphe $G1$ n'est pas homomorphe à un graphe $G2$ alors tous les graphes $G3$ tels que $G1 \subset G3$ ne sont pas homomorphes à $G2$. L'extension d'une solution partielle pour la recherche d'homomorphismes de graphe se base sur le lemme suivant.

***Lemme* 1** *Soient $G_m = (N_m, A_m)$ et $G_h = (N_h, A_h)$ deux graphes. Soient $N_1 \subseteq N_m$ et $N_2 \subseteq N_h$ et soit $F \subseteq N_1 \times N_2$ un isomorphisme entre le sous graphe G_1 induit par l'ensemble des nœuds appartenant à N_1* [14] *et G_2 un sous graphe de G_h ayant comme ensemble de nœuds N_2. Pour tous les nœuds $n_m \in (N_m \setminus N_1)$ et $n_h \in (N_h \setminus N_2)$, $F \cup \{(n_m, n_h)\}$ est un isomorphisme entre le sous graphe de G_m induit par $N_1 \cup \{n_m\}$ et un sous graphe de G_h ayant comme ensemble de nœuds $N_2 \cup \{n_h\}$ si et seulement si pour tous les nœuds $n'_m \in N_1$ si $\{n_m, n'_m\} \in A_m$ alors $\{n_h, F(n'_h)\} \in A_h$.*

Considérons, dans ce qui suit, le problème de trouver tous les homomorphismes entre un graphe mère G_m et un graphe hôte G_h. Nous obtenons l'algorithme par retour arrière présenté dans la figure 1.20.

L'algorithme fait un parcours exhaustif *en profondeur d'abord* de tous les nœuds du graphe G_m et tous les nœuds du graphe G_h. Les nœuds déjà visités sont stockés dans deux ensembles respectivement

[10] Les problèmes *NP-complets* sont les plus difficiles de la classe NP et constituent les problèmes qui ont le plus de chances de ne pas être dans P. Un problème NP-complet est un problème complet pour NP dans le sens où il appartient à la classe NP et qu'il est *NP-difficile* (*NP-hard*) ce qui signifie que tout autre problème de NP est réductible à ce problème.

[11] NP (*Non-deterministic Polynomial time*) est la classe de complexité des problèmes de décision qui peuvent être **résolus** par une machine de *Turing* non déterministe dans un temps polynômial. Ceci est équivalent à dire que c'est l'ensemble des problèmes qui peuvent être **vérifiés** par une machine de *Turing* deterministe dans un temps polynômial.

[12] P (*Polynomial time*) est la classe de complexité contenant les problèmes de décision qui peuvent être résolus par une machine de *Turing* déterministe dans un temps polynômial.

[13] La question de savoir si $P = NP$ ou non est une question qui n'est toujours pas tranchée. Dans un panel informel de 100 scientifiques informatiques, 61 parmi eux ont déclarés qu'ils pensent que $P \neq NP$ et seulement 9 pensent que $P = NP$. Pour l'anecdote, le *Clay Mathematics Institute* de Cambridge offre un million de dollars pour celui qui fera une preuve formelle prouvant que $P = NP$ ou que $P \neq NP$.

[14] $G' = (N', A')$ est un sous graphe de $G = (N, A)$ induit par l'ensemble des nœuds N' si et seulement si G' est un sous graphe de G et que tous les arêtes ou arcs reliant deux nœuds de N dans le graphe G, appartiennent à A'.

pour les graphes G_m et G_h. Les solutions partielles sont étendues en se basant sur la propriété décrite dans le lemme 1 (lignes 1-2-2 et 1-2-3-x).

Si cette propriété n'est pas vérifiée, alors nous pouvons conclure, selon le principe domino, que toutes les configuration qui découlent de la solution partielle ne constituent pas un homomorphisme et nous effectuons donc un retour en arrière (ligne 1-2-4 : le matching (n,n') n'est pas pris en compte et le nœuds n' est marqué comme visité). Si tous les nœuds du graphe hôte ont été visités (i.e. ligne 1-3-1), alors l'algorithme exécute un retour en arrière pour tester les autres branches restantes (ligne 1-3-2).

Dans le cas où un homomorphisme est trouvé (ligne 1-3-3-1) la solution est stockée (ligne 1-3-3-2-1) et, dans la mesure où l'objectif de l'algorithme présenté ici est de trouver tous les homomorphismes de G_m vers G_h, un retour en arrière est exécuté pour chercher les solutions restantes (lignes 1-3-3-2-3 et 1-3-3-2-3). L'algorithme s'arrête une fois qu'il a visité tous les nœuds de G_m et G_h.

Algorithme Général avec retour en arrière

Soient $G_m = (N_m, A_m)$ et $G_h = (N_h, A_h)$. Soient $Sol = \emptyset$ et $hom = \emptyset$.
Soient $Visite_m = \emptyset$ et $Visite_h = \emptyset$.
1- Tant que $Visite_m \neq N_m$ ou $Visite_h \neq N_h$,
  1-1. Si $Visite_m \neq N_m$ et $Visite_h \neq N_h$
  1-2. Alors :
    1-2-1. Soient $n \in N_m \setminus Visite_m$ et $n' \in N_h \setminus Visite_h$
    1-2-2. Si $(\forall n_m \in Visite_m, \{n_m, n\} \in A_m \Rightarrow \{f(n_m), n'\} \in A_h)$
    1-2-3. Alors :
      1-2-3-1. $hom = hom \cup (n, n')$
      1-2-3-2. $Visite_m = Visite_m \cup \{n\}$
      1-2-3-3. $Visite_h = Visite_h \cup \{h\}$
    1-2-4. Sinon :
      1-2-4-1. $Visite_h = Visite_h \cup \{h\}$
  1-3. Sinon :
    1-3-1. Si $Visite_m \neq N_m$ et $Visite_h = N_h$
    1-3-2. Alors :
      1-3-2-1. $Visite_h = Image(hom)$
      1-3-2-2. $hom = hom \setminus dernier(hom)$
      1-3-2-3. $Visite_m = Visite_m \setminus dernier(Visite_m)$
    1-3-3. Sinon :
      1-3-3-1. Si $Visite_m = N_m$ et $Visite_h \neq N_h$
      1-3-3-2. Alors :
        1-3-3-2-1. $Solution = Solution \cup hom$
        1-3-3-2-2. $hom = hom \setminus dernier(hom)$
        1-3-3-2-3. $Visite_m = Visite_m \setminus dernier(Visite_m)$
2- *Retourner(Solution)*

FIG. 1.20 – Algorithme général avec retour en arrière pour la recherche d'homomorphismes de graphes non labelés et non orientés

1.5.2 Algorithme d'Ullmann

l'algorithme d'Ullmann [Ull76] est basé sur le principe de retour arrière en combinaison avec une technique de vérification en avant (*forward-checking*). Il traite la recherche d'homomorphismes entre des graphes orientés avec des nœuds labelés. Une description algorithmique est donnée dans la figure 1.21. Dans ce contexte, la spécification des homomorphismes de graphes est donc celle décrite dans la définition 5 et l'algorithme doit préserver la structure en tenant compte de la direction des arcs (condition (c1) de la ligne 1-2-2) et des labels des nœuds (condition (c2) de la ligne 1-2-2).

Le vrai apport de l'algorithme d'*Ullmann*, outre la considération des arcs orientés et des labels des nœuds, réside dans l'introduction d'une condition supplémentaire (appelée *forward-checking*) sur les

Algorithme d'Ullmann

Soient $G_m = (N_m, A_m, Lab_m)$ et $G_h = (N_h, A_h, Lab_h)$. Soient $Sol = \emptyset$ et $hom = \emptyset$.
Soient $Visite_m = \emptyset$ et $Visite_h = \emptyset$.
1- Tant que $Visite_m \neq N_m$ ou $Visite_h \neq N_h$,
1-1. Si $Visite_m \neq N_m$ et $Visite_h \neq N_h$
1-2. Alors :
1-2-1. Soient $n \in N_m \setminus Visite_m$ et $n' \in N_h \setminus Visite_h$
1-2-2. Si (c1) $(\forall n_m \in Visite_m, (n_m, n) \in A_m \Rightarrow (f(n_m), n') \in A_h)$
et si $(\forall n_m \in Visite_m, (n, n_m) \in A_m \Rightarrow (n', f(n_m)) \in A_h)$,
et si (c2) $Lab_m(n) = Lab_h(n')$
et si (c3) $\forall n_{m,i} \in (N_m \setminus (Visite_m \cup \{n\}))$ alors
$\exists n_{h,j} \in (N_h \setminus (Visite_h \cup \{n'\}))$ tel que :
(c3.1) $Lab_m(n_{m,i}) = Lab_h(n_{h,j})$, et
(c3.2.1) $(n, n_{m,i}) \in A_m \Rightarrow (n', n_{h,j}) \in A_h)$, et
(c3.2.2) $(n_{m,i}, n) \in A_m \Rightarrow (n_{h,j}, n') \in A_h)$, et
(c3.2.3) $(\forall n_m \in Visite_m, (n_m, n_{m,i}) \in A_m \Rightarrow (f(n_m), n_{h,j}) \in A_h)$, et
(c3.2.4) $(\forall n_m \in Visite_m, (n_{m,i}, n_m) \in A_m \Rightarrow (n_{h,j}, f(n_m)) \in A_h)$
1-2-3. Alors :
1-2-3-1. $hom = hom \cup (n, n')$
1-2-3-2. $Visite_m = Visite_m \cup \{n\}$
1-2-3-3. $Visite_h = Visite_h \cup \{h\}$
1-2-4. Sinon :
1-2-4-1. $Visite_h = Visite_h \cup \{h\}$
1-3. Sinon :
1-3-1. Si $Visite_m \neq N_m$ et $Visite_h = N_h$
1-3-2. Alors :
1-3-2-1. $Visite_h = Image(hom)$
1-3-2-2. $hom = hom \setminus dernier(hom)$
1-3-2-3. $Visite_m = Visit_m \setminus dernier(Visite_m)$
1-3-3. Sinon :
1-3-3-1. Si $Visite_m = N_m$ et $Visite_h \neq N_h$
1-3-3-2. Alors :
1-3-3-2-1. $Solution = Solution \cup hom$
1-3-3-2-2. $hom = hom \setminus dernier(hom)$
1-3-3-2-3. $Visite_m = Visite_m \setminus dernier(Visite_m)$
2- *Retourner(Solution)*

FIG. 1.21 – Algorithme d'Ullmann

nœuds qui ne sont pas encore pris en compte dans la solution partielle (conditions (c3) de la ligne 1-2-2).

Pour cette condition, l'algorithme vérifie que chaque nœud du graphe mère qui reste à intégrer dans la solution partielle pour les étapes à venir peut-être unifié avec au moins un nœud du graphe hôte parmi ceux qui n'ont pas encore été pris en compte dans cette solution partielle. Il vérifie pour chacun de ces nœuds $n_{m,i}$, qu'il existe un nœud du graphe hôte $n_{h,j}$ tel que les deux nœuds ont le même label (condition (c3.1)) et que leur unification est consistante par rapport à l'étape d'intégration courante (i.e. conditions (c3.2.1) et (c3.2.2)) et par rapport à la solution partielle (i.e. conditions (c3.2.3) et (c3.2.4)). Si, par exemple, pour un homomorphisme partiel $hom = \{(n_{m,1}, n_{h,i}), \ldots, (n_{m,k}, n_{h,j})\}$ il existe un nœud $n_{m,k+l}$ appartenant au graphe mère qui ne peut être unifié à aucun des nœuds appartenant à l'ensemble $N_h \setminus \{n_{h,i}, \ldots, n_{h,j}\}$ comprenant les nœuds appartenant au graphe hôte qui ne sont pas pris en compte par l'homomorphisme hom, alors l'algorithme effectue un retour en arrière permettant d'éviter le développement d'une branche de l'arbre de recherche qui va se révéler infructueuse.

1.5.3 Algorithme de Messmer & Bunke

Algorithme de Messmer

Soit $G_m = (N_m, A_m, Lab_m)$ avec $N_m = \{n_{m,1}, \ldots, n_{m,k}\}$.
Soit $G_h = (N_h, A_h, Lab_h)$ avec $N_h = \{n_{h,1}, \ldots, n_{h,l}\}$.
Soient $Sol = \{f_i = (n_{m,1}, n_{h,i}) \mid Lab_m(n_{m,1}) = Lab_h(n_{h,i})\}$ et $Sol' = \emptyset$.
1- Pour tout I tel que $1 < I \leq k$
  1-1. Si $Sol = \emptyset$
    1-1-1. Alors retourner($\emptyset$)
  1-2. Sinon :
    1-2-1. Pour tout f tels que $f \in Sol$
      1-2-1-1. Pour tous les nœuds $n_h \in (N_h \setminus Image(f))$
        1-2-1-1-1. Si $(\forall n'_m \in N_{temp}, \{n'_m, n_{m,I}\} \in A_m \Rightarrow \{f(n'_m), n_h\} \in A_h)$
          1-2-1-1-1-1. Alors $Sol' := Sol' \cup (f \cup (n_{m,I}, n_h))$
        1-2-1-1-2. Sinon Rien
    1-2-2. $Sol := Sol'$
    1-2-3. $Sol' := \emptyset$
2- retourner(Sol)

FIG. 1.22 – Algorithme de Messmer & Bunke

L'algorithme de *Messmer* traite le même type de graphes que l'algorithme d'*Ullmann* (i.e. graphes orientés avec nœuds labelés) mais se base sur une recherche en largeur d'abord.

Les premières solutions partielles concernant le nœud $n_{m,1}$ du graphe hôte G_h sont les couples $(n_{m,1}, n_{h,i})$ tels que ($n_{m,1}$ et $n_{h,i}$) ont le même label (cf. initialisation de la variable Sol dans la description de l'algorithme). L'algorithme intègre, ensuite, le nœud $n_{m,2}$ qui sera unifié avec un nœud $n_{h,j}$ tel que $n_{h,i} \neq n_{h,j}$ (car un homomorphisme est par définition injectif) et $Lab_n(n_{m,2}) = Lab_h(n_{h,j})$ et tel que s'il existe un arc $(n_{m,1}, n_{m,2}) \in A_m$ alors il existe un arc $(n_{h,i}, n_{h,j}) \in A_h$.

Ce processus est répété pour tous les autres nœuds $n_{m,j}$ du graphe mère G_m et l'algorithme produit, à chaque itération, tous les homomorphismes partiels entre le sous graphe de G_m induit par l'ensemble des nœuds $(n_{m,1}, \ldots, n_{m,I})$ et le graphe hôte G_h (cf. intégration du nœud $n_{m,I}$ à chaque itération I de la boucle de la ligne 1).

L'algorithme s'arrête avec succès et renvoie tous les homomorphismes existants après avoir intégré tous les nœuds de G_m. Grâce au principe domino, l'algorithme s'arrête en prouvant l'absence d'homomorphismes entre G_m et G_h si, à n'importe quelle itération, I l'ensemble des solutions partielles est vide (cf. ligne 1-1).

La complexité de l'algorithme de *Messmer* est équivalente à celle de l'algorithme d'*Ullmann*[15]. Cependant, *Messmer* a étendu son algorithme qui se montre plus performant dans le cas où l'objectif consiste à trouver tous les homomorphismes entre plusieurs graphes mères et le même graphe hôte [MB00].

L'algorithme introduit la notion de graphes communs entre les graphes mères. L'organisation de l'ordre d'introduction des nœuds pour l'extension de la solution partielle est réalisée de telle manière qu'on commence par les nœuds appartenant à ces graphes communs. L'unification est ainsi réalisée une fois pour toutes et cela pour tous les graphes contenant ce nœud. L'efficacité de cette approche reste, par contre, tributaire de la ressemblance des graphes mères et ne prend son sens que dans le cas ou ces graphes sont très proches. Dans le cas contraire, elle s'avère même handicapante puis qu'elle introduit des étapes de calcul supplémentaires pour la déduction des graphes communs.

1.5.4 Algorithme par détection de cliques

L'algorithme par détection de cliques a été introduit dans [BB76] pour la recherche des morphismes pour les graphes non orientés avec des nœuds labelés. Cet algorithme se base sur un représentation

[15] Dans le cas où nous cherchons **tous** les homomorphismes.

graphique du problème et défini pour cela les notions de *graphes d'association* et de *cliques* présentées respectivement dans les définitions 7 et 8.

Les nœuds du graphe d'association définissent les différentes possibilités d'unification entre les nœuds du graphe mère et du graphe hôte et sont labélés par les identifiants de ces deux nœuds. Un nœud appartenant à ce graphe d'association et labelé par (n_m, n_h) exprime le fait que les deux nœuds n_m et n_h sont unifiables (i.e. ont le même label).

Les arcs du graphe d'association expriment les compatibilités entre les différentes unifications de nœuds. Ainsi, un arc reliant deux nœuds labélés respectivement par $(n_{m,1}, n_{h,1})$ et par $(n_{m,2}, n_{h,2})$ exprime le fait que les deux unifications correspondant à ces deux nœuds sont compatibles i.e. $n_{m,1}$ (respectivement $n_{h,1}$) et $n_{m,2}$ (respectivement $n_{h,2}$) sont deux nœuds différents et que s'il existe dans le graphe mère un arc entre $n_{m,1}$ et entre $n_{m,2}$ alors il existe dans le graphe hôte un arc entre $n_{h,1}$ et $n_{h,2}$[16].

***Définition* 7 (Graphe d'association)** *Étant donnés deux graphes $G_m = (N_m, A_m, Lab_m)$ et $G_h = (N_h, A_h, Lab_h)$, le graphe d'association de G_h et G_m est un graphe $GA = (N_{GA}, A_{GA})$ avec $N_{GA} \subseteq N_m \times N_h$ et $A_{GA} \subseteq N_{GA} \times N_{GA}$ tels que :*
1- $N_{GA} = \{(n_m, n_h)$ avec $n_m \in N_m$ et $n_h \in N_h$ / $Lab_m(n_m) = Lab_h(n_h)\}$
2- $A_{GA} = \{(n_{GA,1}, n_{GA,2})$ avec $n_{GA,1} = (n_{m,1}, n_{h,1})$ et $n_{GA,2} = (n_{m,2}, n_{h,2})$ / $(n_{m,1}, n_{m,2}) \in A_m \Rightarrow (n_{h,1}, n_{h,2}) \in A_h$

***Définition* 8 (Clique)** *Soit G un graphe, une clique est un ensemble de nœuds N telle qu'il existe un arc connectant chaque couple de nœuds appartenant à N. Ceci est équivalent à dire que le graphe induit par N est un graphe complet.*

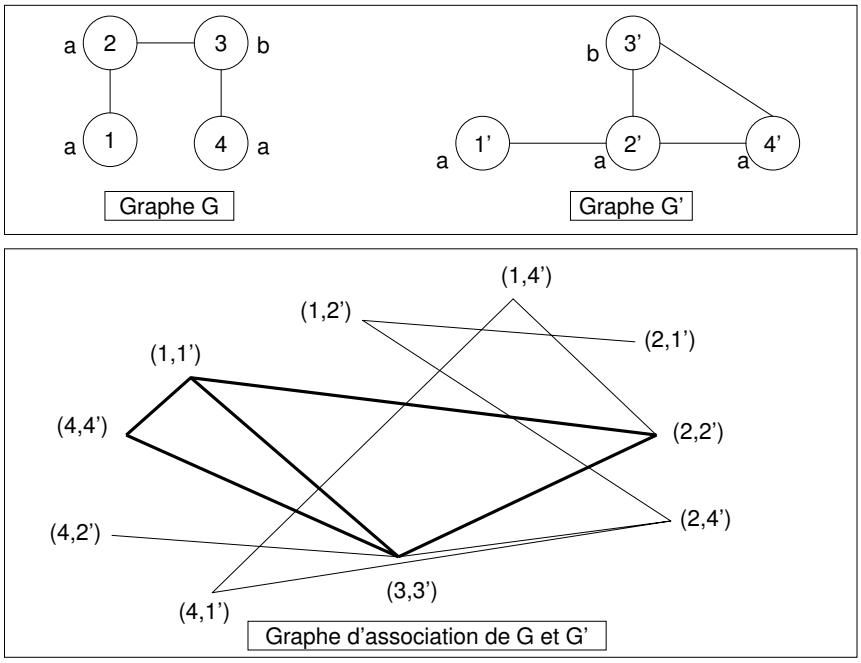


FIG. 1.23 – Recherche du plus grand isomorphisme de sous graphes par l'approche de détection de cliques.

La recherche d'homomorphismes entre les graphes est ramenée par cet algorithme à la recherche de cliques dans le graphe d'association puisque chaque clique de nœuds $n_{GA,1} = (n_{m,1}, n_{h,1}), \ldots, n_{GA,i} =$

[16] Dans le cas où on recherche des isomorphismes, une condition supplémentaire est introduite stipulant que s'il existe dans le graphe hôte un arc entre $n_{h,1}$ et $n_{h,2}$, alors il existe dans le graphe mère un arc entre $n_{m,1}$ et entre $n_{m,2}$.

$(n_{m,i}, n_{h,i})$ représente un isomorphisme entre le sous graphe du graphe mère induit par les nœuds $n_{m,1}, \ldots, n_{m,i}$ et un sous graphe du graphe hôte contenant les nœuds $n_{h,1}, \ldots, n_{h,i}$. La résolution du problème consiste, donc, à trouver une clique impliquant tous les nœuds du graphe mère (i.e. une clique de la même taille que N_m). De manière générale, l'algorithme de détection de cliques permet de trouver tous les isomorphismes entre G_m et G_h s'ils sont de même taille, et tous les homomorphismes s'ils sont de tailles différentes.

La complexité de l'algorithme reste cependant exponentielle puisque la détection de cliques avec une taille donnée dans un graphe est aussi un problème NP-Complet. L'algorithme est surtout utilisé dans le cas où on mise sur l'absence d'homomorphismes, ou dans le cas où l'on cherche à trouver le plus grand sous-graphe commun entre les deux graphes. Dans ce dernier cas, la recherche du plus grand sous-graphe commun peut être ramenée à la recherche de la clique maximale dans le graphe d'association.

Un exemple de la construction du graphe d'association et des cliques est donné dans la figure 1.23. Dans cet exemple les deux cliques maximales dans le graphe d'association représentent les deux plus grands sous-graphes communs entre G et G'.

1.6 Conclusion

Nous avons présenté, dans cette section les algorithmes traitant le problème de recherche d'homomorphismes dans les graphes. Ces algorithmes considèrent les graphes sans contraintes spécifiques sur leurs structures. Ils sont, par conséquent, caractérisés par une complexité exponentielle puisque le problème est NP-Complet.

Dans la littérature, on retrouve d'autres algorithmes spécialisés pour le traitement de graphes spécifiques qui ont des complexités inférieures. Ces algorithmes n'ont pas été pris en compte dans ce chapitre du fait que la problématique traitée dans ce manuscrit n'introduit aucune limitation ou contrainte ni sur la structure ni sur les labels des nœuds. Notre approche présente, par contre, une spécificité concernant l'enchaînement de l'application des règles. Nous verrons, donc, dans une version améliorée de notre algorithme comment nous avons profité de cette propriété pour implémenter un algorithme efficace pour la recherche des homomorphismes de graphes.

Nous avons présenté dans ce chapitre l'état de l'art des travaux relatifs aux différentes approches que nous présenterons dans la suite de ce manuscrit. Nous présenterons, dans le chapitre qui suit, notre approche basée sur les grammaires de graphes pour la description des architecture logicielles. Nous présenterons, dans ce même cadre, une approche pour le raffinement et l'abstraction de ces descriptions en nous basant sur les grammaires edNCE. Nous utiliserons aussi des combinaisons de règles de réécriture edNCE pour spécifier le protocole de reconfiguration architecturale.

Chapitre 2

Un Méta-Modèle pour les Architectures Dynamiques

2.1 Introduction

Dans ce chapitre, nous abordons la spécification d'un cadre formel pour la description des architectures logicielles orientées-composants et services. Une architecture est décrite par un graphe enrichi permettant d'intégrer les différents paramètres de ses composants et leurs interdépendances. Nous utilisons le concept de styles architecturaux pour décrire les architectures dynamiques et la spécification de leurs instances consistantes. Un style architectural est caractérisé par un modèle basé sur des grammaires de graphes étendues.

Dans un deuxième temps, nous introduisons une approche pour le raffinement et l'abstraction des architectures dynamiques. Nous montrons l'intérêt de distinguer plusieurs niveaux d'abstraction dans la description. Nous présentons, pour cela, une approche formelle permettant de décrire les systèmes de transformation pour traduire une description définie à un niveau d'abstraction donné vers un autre niveau d'abstraction. Cette approche se base aussi sur les grammaires de graphes. Nous verrons l'intérêt d'intégrer dans ce formalisme des mécanismes de type NCE.

Enfin, nous présentons un méta-modèle permettant de caractériser les événements et les règles ainsi que les protocoles de reconfiguration des architectures. Ces protocoles correspondent aux entités en charge de contrôler et de gérer l'évolution dynamique de l'architecture suite aux événements qu'ils reçoivent et en se basant sur des règles de reconfiguration génériques. Nous utiliserons pour la caractérisation des règles de reconfiguration des règles de transformation de graphes considérant des conditions d'application négatives et des instructions de connexion de type NCE.

Tout au long de ce chapitre, nous utilisons un exemple simple pour illustrer les différentes notations et concepts. Cet exemple se situe dans le contexte des architectures orientées services et considère trois types de composants correspondant aux consommateurs de services, aux services simples et aux services composites.

2.2 Description des architectures dynamiques

Le formalisme présenté ici a pour objectif la description des architectures dynamiques. Nous commençons par la spécification des entités élémentaires comprenant les composants de l'architecture et les interdépendances qui les relient. Nous définissons, ensuite, une notation et un formalisme permettant la description des styles architecturaux par la caractérisation de toutes les instances correctes de l'architecture.

Dans le chapitre précédent, nous avons souligné quelques insuffisances relatives aux approches relevant des ADLs classiques pour la description des architectures dynamiques. Une description des architectures basée sur les graphes présente, outre l'aspect formel, l'avantage d'offrir une description visuelle et faci-

lement compréhensible qui ne nécessite (au moins pour comprendre et communiquer) aucun pré-requis concernant les langages formels. Ceci présente un avantage certain dans la mesure où la description se situe en amont du cycle de développement de logiciel (i.e. la spécification) à un niveau qui implique des intervenants de différents horizons professionnels et avec différents domaines d'expertise.

2.2.1 Description d'une instance de l'architecture

Nous optons pour un cadre général basé sur les graphes et les grammaires de graphes où un graphe décrit une instance de l'architecture. Nous allons enrichir et étendre ces formalismes pour les adapter à notre problématique.

Les nœuds de ce graphe représentent les composants logiciels de l'application. Ces composants peuvent correspondre à des unités de calcul indépendantes telles que des composants EJB (Entreprise Java Beans), ou des Services Web. Ils peuvent aussi désigner d'autres entités logicielles telles que des registres UDDI ou des bases de données.

La description des composants logiciels inclut, en général, la spécification de plusieurs attributs. Ces attributs sont relatifs aux paramètres publiés par les composants pour faciliter leurs interconnexion et leur intégration dans l'application, ou pour aider à la gestion de leur cycle de vie. Ces paramètres peuvent se situer au niveau applicatif et représenter, par exemple, l'état courant du composant, son identité, les méthodes et services qu'il offre, ou sa localisation. D'autres paramètres de niveau coopération peuvent aussi être pris en compte tels que son rôle dans l'application, les protocoles d'interaction et les paramètres de niveau QdS.

Les paramètres nécessaires pour la description d'un composants peuvent varier d'un type de composants à un autre et d'un contexte de description à un autre. Les nœuds des graphes sont, donc, *multi-labélés* où chaque label correspond à un paramètre du composant.

Les interdépendances entre composants sont spécifiées par des arcs reliant les nœuds qui décrivent ces composants. Dans une application donnée, on peut retrouver plusieurs types d'interdépendances spécifiant, par exemple, des liens ou des canaux de communication de différents types (e.g. client-serveur, pair-à-pair, synchrone, asynchrone) ou des liens de contrôle et de coopération (e.g. composition, orchestration, chorégraphie).

Pour permettre la différenciation entre ces différents liens, les arcs du graphe sont étiquetés par le type et les propriétés des liens qu'ils représentent.

Le cadre de description correspond, donc, aux graphes étiquetés et labélés. Ces graphes admettent pour les deux entités concernant les nœuds et les arcs, des labels et des étiquettes multiples. Le nombre des labels pour les nœuds et des étiquettes pour les arcs peut varier d'un nœud à un autre ou d'un arc à un autre. La Définition 9 donne une caractérisation formelle du type de graphes considéré.

***Définition* 9** *Un graphe G est spécifié par le système :*
$G = (N, A, (Lab_1, \ldots, Lab_n), (L_1, \ldots, L_n), (Etiq_1, \ldots, Etiq_m), (E_1, \ldots, E_m))$. *Où, N et A correspondent respectivement à l'ensemble des nœuds et à l'ensemble des arcs du graphe. $L_1, \ldots, L_n$ et $Lab_1, \ldots, Lab_n$ correspondent respectivement aux domaines de définition des labels des nœuds et des fonctions permettant leur génération. $(E_1, \ldots, E_m)$ et $(Etiq_1, \ldots, Etiq_m)$ correspondent respectivement aux domaines de définition des étiquettes des arcs et des fonctions permettant leur génération.*

***Convention* 1** *Afin d'alléger les notations, nous prenons, dans la suite de ce manuscrit, les conventions de notation suivantes :*
(1) La description d'un nœud $n1$ sera donné sous la forme $n1(Lab_i(n1), \ldots, Lab_j(n1))$ où $Lab_i(n1), \ldots, Lab_j(n1)$ correspondent aux labels considérés pour $n1$.
(2) Un arc multi-étiqueté sera donné sous la forme $n1 \xrightarrow{(Etiq_k(n1,n2),\ldots,Etiq_l(n1,n2))} n2$ où $Etiq_k(n1,n2), \ldots, Etiq_l(n1,n2)$ correspondent aux étiquettes considérées pour l'arc reliant $n1$ à $n2$.
(3) Un graphe G sera décrit par la paire $G = (N, A)$ où N et A correspondent respectivement à l'ensemble des nœuds et à l'ensemble des arcs décrits selon les conventions définies dans les points (1) et (2).

Nous illustrons notre approche de description par un exemple simple dans le contexte des architectures orientées-services. Nous considérons trois types de composants correspondant aux consommateurs de services, aux services simples et aux services composites.

Pour chaque composant, nous considérons un premier paramètre correspondant à l'identifiant du composant. Le deuxième paramètre correspond au type du composant. Nous nous intéressons aussi à la localisation du composant en introduisant un paramètre relatif à la machine sur laquelle il est déployé. Pour les composants de type service simple nous exposons, en plus, un paramètre de QdS indiquant le temps de réponse affiché par le service.

Nous considérons deux types de liens entre les composants : 1) le premier concerne les liens de communication qui lient un consommateur à un service simple ou composite, alors que 2) le second concerne les liens de composition liant un service composite aux services simples ou composites qu'il contrôle.

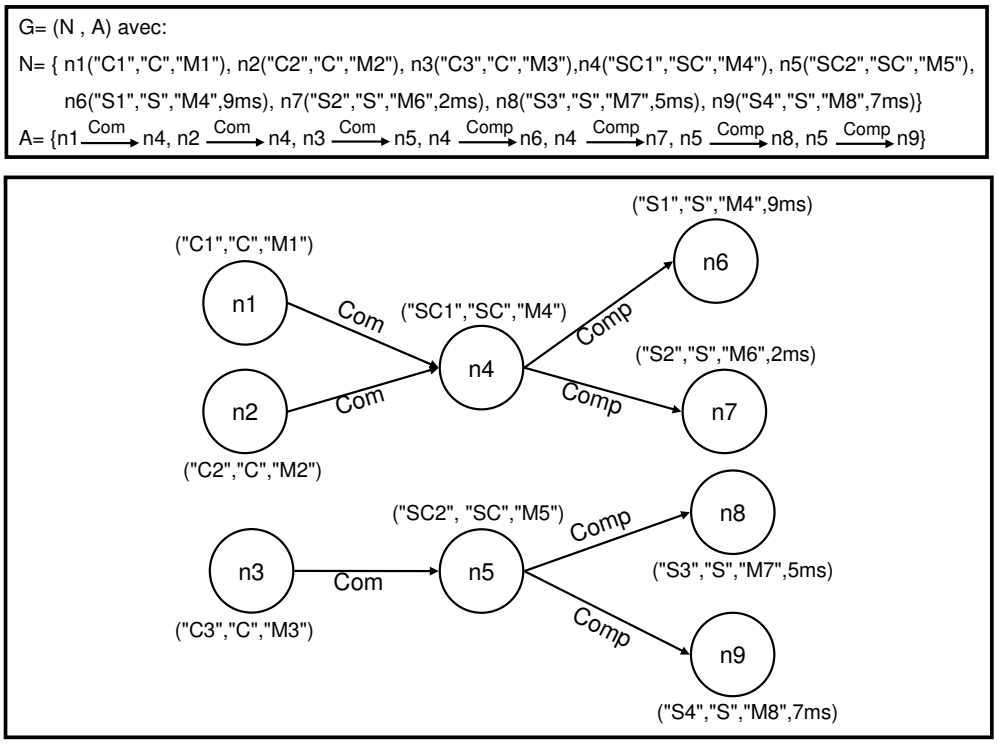


FIG. 2.1 – Descriptions textuelle et visuelle d'une architecture orientée-service.

La figure 2.1 présente une description d'une instance de cette architecture. Pour prendre en compte les paramètres introduits dans le paragraphe précédent, tous les nœuds possèdent trois labels qui désignent l'identité du composant, son type (i.e. "C" pour les consommateurs, "S" pour les services simples et "SC" pour les services composites), et sa localisation (i.e. la machine sur laquelle il est déployé). Les nœuds représentant des services simples possèdent un label supplémentaire permettant de spécifier le paramètre relatif à leur temps de réponse. Les arcs seront étiquetés par la nature du lien qu'ils représentent (i.e. "Com" pour les liens de communication et "Comp" pour les liens de composition).

La configuration de l'architecture considérée est constituée de trois consommateurs *C1*, *C2* et *C3* déployés sur trois machines distinctes M1, M2 et M3. Les consommateurs *C1* et *C2* communiquent avec le service composite *SC1* (liens de communication spécifiés par deux arcs étiquetés par *"Com"*) qui est déployé sur une machine M4. *C3* est l'unique consommateur du service composite *SC2* qui est déployé sur la machine M5. Le service *SC1* est constitué des services simples *S1* et *S2* (liens de composition spécifiés par deux arcs de type *"Comp"*). Le service simple *S1* est déployé sur la même machine M4 que *SC1* alors que *S2* est déployé sur une machine distincte M6. Les deux services exhibent respectivement un temps de réponse de 9ms et de 2ms. Le service composite *SC2* contient les services *S3* et *S4* qui offrent un temps de réponse respectivement de 5ms et de 7ms.

2.2.2 Description des styles architecturaux

La description donnée précédemment permet de décrire les éléments d'une instance de l'architecture en spécifiant les composants qui la constituent et les interdépendances qui les lient. Dans le cadre d'une application dont l'architecture est dynamique, il est nécessaire d'intégrer, dans la description de son style architectural, les contraintes considérées par le cahier des charges et par les choix de conception. Une solution simple serait de lister, une à une, toutes les instances consistantes, mais cette option n'est envisageable que si le nombre de ces instances est réduit ou du moins borné.

Nous allons utiliser les grammaires de graphes pour la description des styles architecturaux. L'idée est d'utiliser l'aspect génératif des grammaires pour caractériser rigoureusement les instances consistantes de l'architecture. Toute configuration correspondant à un graphe généré par la grammaire sera considérée comme consistante. D'autre part, toute configuration qui ne peut être générée par la grammaire est considérée comme inconsistante et ne faisant pas partie du style architectural de l'application.

Nous avons présenté, dans le chapitre qui traite de l'état de l'art, plusieurs approches et mécanismes pour la définition des grammaires de graphes. Dans la suite, nous introduisons le formalisme que nous considérons pour la caractérisation des styles architecturaux et nous décrivons les extensions introduites pour répondre à nos besoins de description.

Concernant la structure des productions de grammaire, nous optons pour une combinaison des approches Single PushOut et Double PushOut. Nous exploitons la structure de l'approche DPO permettant de désigner de manière explicite la partie du graphe mère qui sera maintenu après l'application de la production. Pour le traitement des arcs suspendus, nous optons pour l'approche SPO qui offre une puissance d'expression plus grande que l'approche DPO. Ceci implique la suppression de tous les arcs suspendus au lieu d'exiger leur absence comme précondition à l'application de la production.

Concernant le type de labels et des étiquettes considérés, le formalisme hérite des contraintes et des besoins introduits précédemment pour la description des instances de l'architecture : Les nœuds terminaux et les nœuds non-terminaux admettent plusieurs labels et les arcs considèrent plusieurs étiquettes.

Les différents paramètres considérés dans la description des composants logiciels appartiennent généralement à des domaines de définition infinis (i.e. des nombres entiers, des chaînes de caractères... etc.). La considération de labels (représentant ces différents paramètres) exclusivement constants n'est donc pas réaliste. En effet, ceci impliquerait de prendre en compte toutes les combinaisons possibles des paramètres des composants et des interdépendances, et produirait des modèles infinis ou de très grande taille.

Nous introduisons, donc, des labels variables et nous considérons des productions de grammaires paramétrées. Une grammaire de graphes est décrite par le quatruplet (AX, NT, T, P) où P est un ensemble de productions de grammaires de la forme $p[(X_1, ..., X_i), (D_1, \ldots, D_i)]$ où $X_1, \ldots, X_i$ sont des labels variables de nœuds (terminaux ou non terminaux) appartenant au graphe mère ou au graphe fille de la production p tandis que $D_1, \ldots, D_i$ correspondent respectivement aux domaines de définition de ces variables. L'affectation d'un paramètre X_j de cette production de grammaire par une valeur V_j appartenant au domaine D_j implique l'affectation de tous les labels des nœuds correspondant à X_j par V_j.

***Convention* 2** *Afin de distinguer les labels variables des labels de type chaînes de caractère, nous notons ces derniers en insérant des guillemets. Ainsi, XX correspond à la variable XX tandis que "XX" correspond à la constante de type chaîne de caractère XX.*

Un exemple d'une production de grammaire est donné dans la figure2.2. Dans cet exemple, nous présentons une production qui permet de générer un consommateur supplémentaire. La production générique requiert l'existence d'un service composite dont l'identifiant est représenté par la variable *X1* et qui est déployé sur une machine dont l'adresse est représentée par la variable *Y1*. L'application de cette production implique l'introduction d'un consommateur *X2* localisé sur la même machine *Y1*. De manière informelle, cette production peut-être traduite par la phrase suivante : "Si l'application contient au moins un service composite, alors il est possible de générer un consommateur supplémentaire déployé sur la même machine que ce service. Les deux composants seront connectés par un canal de communication".

L'affectation de cette production telle que cela est présenté dans l'exemple de la figure2.2, permet de spécifier sur quelle machine le service composite est déployé (i.e. M5) et de donner l'identifiant que nous allons affecter au consommateur à générer (i.e. C4). De manière informelle, cela revient à dire : "S'il

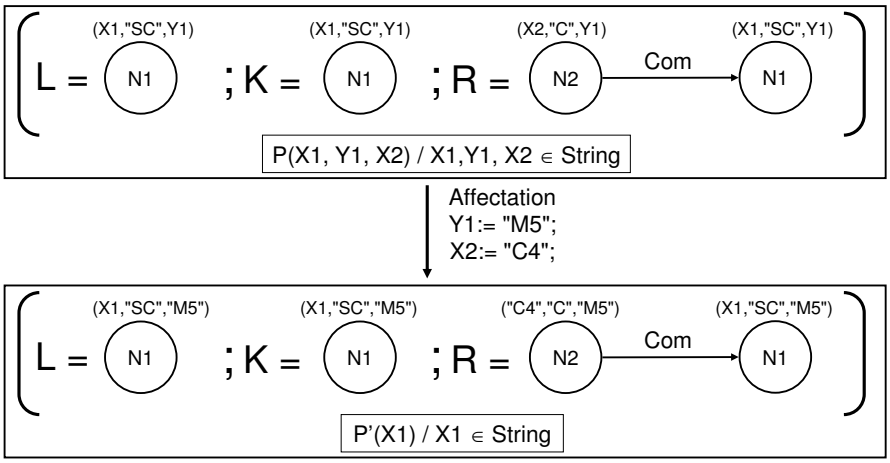


FIG. 2.2 – Exemple d'une production de grammaire paramétrée et de la production résultant d'une affectation partielle de ces paramètres.

existe au moins un service composite déployé sur la machine M5, alors générer un consommateur qu'on va nommer C4 et qui sera déployé sur la machine M5, et connecter les deux composants par un canal de communication".

En considérant l'introduction des variables et de leurs domaines dans la spécification des grammaires de graphes, nous admettons qu'un graphe G appartient à une grammaire de graphes GG si et seulement si G peut être généré à partir de l'axiome par une séquence des productions de GG. Les productions prises en compte dans cette séquence correspondent à toutes les productions p définies dans l'ensemble P ainsi qu'à toutes les productions obtenues en instanciant complètement ou partiellement les labels variables de ces productions par des constantes appartenant à leurs domaines de définition respectifs.

La possibilité de n'instancier que partiellement les productions de grammaires avant leur application nous amène à considérer des homomorphismes entre des graphes contenant des labels variables (e.g. les graphes mères et filles des productions) et entre des graphes contenant exclusivement des labels constants (i.e. les graphes intermédiaires produits par le processus de génération).

Nous introduisons dans la suite la caractérisation que nous considérons pour les homomorphismes de graphes en tenant compte de ce contexte. Nous donnons dans la définition 10 une définition formelle des graphes contenant des labels variables. Puis, nous introduisons dans la définition 11 la notion d'unification entre les nœuds et entre les arcs. Cette notion permet la prise en compte des labels variables et de la consistance de leur affectation par l'homomorphisme.

***Définition* 10 (Caractérisation des graphes avec labels variables)** *Un graphe considérant n types de nœuds et m types d'arcs différents et l'ensemble des variables $S_X = \{X_1, \ldots, X_i\}$ est défini par le système $G = (N, A, (Lab_1, \ldots, Lab_n), (L_1 \cup S1_X, \ldots, L_n \cup Sn_X), (Etiq_1, \ldots, Etiq_m), (E_1, \ldots, E_m))$. N et E correspondent respectivement à l'ensemble des nœuds et des arcs du graphe. $L_1, \ldots, L_n$ et $Lab_1, \ldots, Lab_n$ correspondent respectivement aux domaines de définition des labels des différents types de nœuds et des fonctions permettant leur génération et Si_X est un sous ensemble de S_X. De la même manière, $(E_1, \ldots, E_m)$ et $(Etiq_1, \ldots, Etiq_m)$ correspondent respectivement aux domaines de définition des étiquettes des différents types d'arcs et des fonctions permettant leur génération.*

***Définition* 11 (Unification de nœuds)** *Un nœuds $N1(lab_{N1,1}, \ldots, lab_{N1,n})$ avec $(lab_{N1,1}, \ldots, lab_{N1,n})$ contenant des constantes et des variables est unifiable avec un nœud $N2(lab_{N2,1}, \ldots, lab_{N2,m})$ avec $(lab_{N2,1}, \ldots, lab_{N2,m})$ constitué exclusivement par des constantes si et seulement si :*
1) $N1$ et $N2$ ont le même nombre et types de labels (i.e. $n = m$ et $L_{N1,i} = L_{N2,j}$), et
2) $(\forall 1 \leq i \leq n)$, (si $lab_{N1,i}$ est une constante, alors $lab_{N1,i} = lab_{N2,i}$), et

3) $(\forall 1 \leq i \leq n)$ *et* $(\forall 1 \leq j \leq n)$*, (si* $lab_{N1,i}$ *et* $lab_{N1,j}$ *sont deux variables telles que* $lab_{N1,i} = lab_{N1,j}$*, alors* $lab_{N2,i} = lab_{N2,j}$*)*

Nous obtenons, donc, la définition 4.6 pour les homomorphismes de graphes dans le contexte du formalisme présenté ici.

***Définition* 12 (Homomorphismes de graphes avec labels variables)** *Un graphe* $G1(N1, A1)$ *admettant des nœuds multi-labélés avec des constantes et des variables est homomorphe à* $G2(N2, A2)$ *considérant des nœuds multi-labélés avec des constantes si et seulement s'il existe une injection* $F \subseteq N1 \times N2$ *qui vérifie :*
1) pour chaque nœud $n \in N1$ *et chaque nœud* $m \in N2$ *tel que* $F(n_i) = m_i$ *alors* n_i *est unifiable avec* m_i*, et*
2) pour chaque paire $n_i, n_j \in N1$ *et* $m_i, m_j \in N2$ *avec* $F(n_i) = m_i$ *et* $F(n_j) = m_j$*, si* $(n_i, n_j) \in A1$ *alors* $(m_i, m_j) \in A2$ *et les deux arcs ont la même étiquette, et*
3) toutes les affectations résultant de l'unification sont consistantes.

La notion d'affectations consistantes sera traitée formellement dans le chapitre 4 relatif aux algorithmes de recherche d'homomorphismes. De manière informelle, cette condition de consistance exprime le fait qu'une même variable apparaissant dans les labels de deux nœuds ne peut être affectée par deux valeurs différentes.

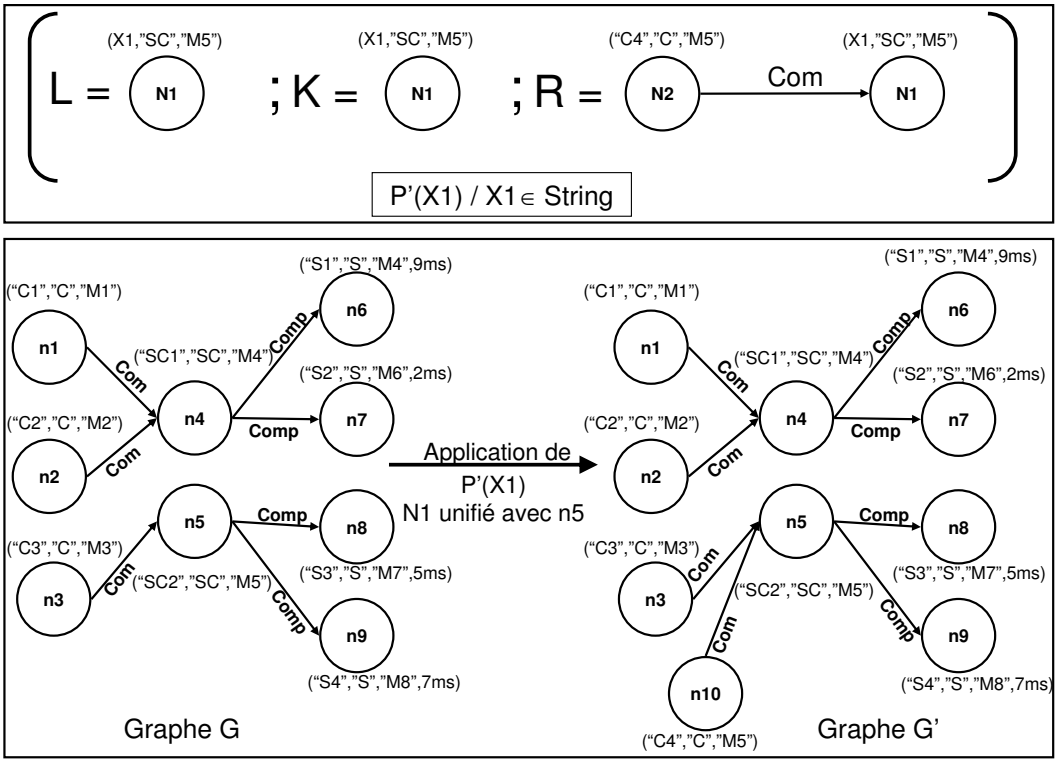


FIG. 2.3 – Exemple de l'application d'une production de grammaire paramétrée.

Dans la figure 2.3 nous présentons un exemple de l'application d'une production de grammaire. Dans cet exemple, l'application de la production P' (produite après affectation de la variable $Y1$ par la valeur $M5$ et de la variable $X2$ par la valeur $C4$ dans la production P de la figure 2.2), implique l'introduction d'un nouveau consommateur $C4$ déployé sur la machine $M5$. Cette production est applicable car le graphe

L est homomorphe au sous graphe constitué par le nœud $n5$: les nœuds N1 et n5 sont unifiables si l'on considère l'affectation $X1 := SC2$.

Exemple du producteur-consommateur

Dans ce qui suit nous donnons une description, selon notre modèle, de l'architecture dynamique pour les applications orientées services en considérant des services composites. Pour illustrer l'intérêt de cette spécification formelle, nous exposeront quelques propriétés topologiques que cette architecture doit respecter. L'objectif est de produire un modèle permettant de générer les configurations architecturales qui répondent aux contraintes exprimées pour ce style architectural.

Nous considérons, donc, les contraintes architecturales suivantes :
0) l'architecture vide n'est pas une architecture consistante.
1) Un consommateur peut communiquer avec un service simple ou un service composite.
2) Un consommateur communique au moins avec un service simple ou un service composite.
3) Un consommateur ne communique jamais avec un autre consommateur (i.e. on ne prend en compte que des consommateurs purs).
4) Les consommateurs peuvent communiquer en même temps avec plusieurs services simples et avec plusieurs services composites.
5) Les consommateurs peuvent partager des services simples ou des services composites.
6) Un service composite contient forcément au moins un service.
7) Un service composite peut être composé d'un ou de plusieurs services composites.
8) Les services composites peuvent partager des services simples ou des services composites.
9) Il n'y a aucune restriction sur le déploiement des composants.

La prise en compte de ces contraintes architecturales permet de définir la grammaire de graphes suivante :
$GG = (AX, NT, T, P)$ avec :
$NT = \{N(''Temp'')\}$, et
$T = \{N(X_C,'' C'', Y_C), N(X_{SC},'' SC'', Y_{SC}), (X_S,'' S'', Y_S, T_S)\}$, et
$P = \{p_1, \ldots, p_{10}\}$

$p_1 = (L = \{N1(AX)\}; K = \{\ \};$ $R = \{N2(X_C,'' C'', Y_C), N3(X_S,'' S'', Y_S, T_S), N4(''Temp''), (N2 \xrightarrow{Com} N3)\})$ $(X_C, Y_C, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_2 = (L = \{N1(AX)\}; K = \{\ \};$ $R = \{N2(X_C,'' C'', Y_C), N3(X_{SC},'' SC'', Y_{SC}), N4(X_S,'' S'', Y_S, T_S), N5(''Temp''),$ $N2 \xrightarrow{Com} N3, N3 \xrightarrow{Comp} N4\})$ $(X_C, Y_C, X_{SC}, Y_{SC}, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_3 = (L = \{N1(''Temp''), N2(X_C,'' C'', Y_C)\},$ $K = \{N1(''Temp''), N2(X_C,'' C'', Y_C)\}$ $R = \{N3(X_S,'' S'', Y_S, T_S), (N2 \xrightarrow{Com} N3)\})$ $(X_C, Y_C, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_4 = (L = \{N1(''Temp''), N2(X_C,'' C'', Y_C)\},$ $K = \{N1(''Temp''), N2(X_C,'' C'', Y_C)\}$ $R = \{N3(X_{SC},'' SC'', Y_{SC}), N4(X_S,'' S'', Y_S, T_S), (N2 \xrightarrow{Com} N3), (N3 \xrightarrow{Comp} N4)\})$ $(X_C, Y_C, X_{SC}, Y_{SC}, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_5 = (L = \{N1(''Temp''), N2(X_C,'' C'', Y_C), N3(X_S,'' S'', Y_S, T_S)\},$ $K = \{N1(''Temp''), N2(X_C,'' C'', Y_C), N3(X_S,'' S'', Y_S, T_S)\},$ $R = \{(N2 \xrightarrow{Com} N3)\})$ $(X_C, Y_C, X_S, Y_S \in String) \wedge (T_S \in Time)$

$p_6 = (L = \{N1(''Temp''), N2(X_C,'' C'', Y_C), N3(X_{SC},'' SC'', Y_{SC})\},$ $K = \{N1(''Temp''), N2(X_C,'' C'', Y_C), N3(X_{SC},'' SC'', Y_{SC})\},$ $R = \{(N2 \xrightarrow{Com} N3)\})$ $(X_C, Y_C, X_{SC}, Y_{SC} \in String)$
$p_7 = (L = \{N1(''Temp''), N2(X_{SC},'' SC'', Y_{SC})\},$ $K = \{N1(''Temp''), N2(X_{SC},'' SC'', Y_{SC})\}$ $R = \{N3(X_S,'' S'', Y_S, T_S), (N2 \xrightarrow{Comp} N3)\})$ $(X_{SC}, Y_{SC}, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_8 = (L = \{N1(''Temp''), N2(X_{SC1},'' SC'', Y_{SC1})\},$ $K = \{N1(''Temp''), N2(X_{SC1},'' SC'', Y_{SC1})\}$ $R = \{N3(X_{SC2},'' SC'', Y_{SC2}), N4(X_S,'' S'', Y_S, T_S), (N2 \xrightarrow{Comp} N3), (N3 \xrightarrow{Comp} N4)\})$ $(X_{SC1}, Y_{SC1}, X_{SC2}, Y_{SC2}, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_9 = (L = \{N1(''Temp''), N2(X_{SC},'' SC'', Y_{SC}), N3(X_S,'' S'', Y_S, T_S)\},$ $K = \{N1(''Temp''), N2(X_{SC},'' SC'', Y_{SC}), N3(X_S,'' S'', Y_S, T_S)\}$ $R = \{(N2 \xrightarrow{Comp} N3)\})$ $(X_{SC}, Y_{SC}, X_S, Y_S \in String) \wedge (T_S \in Time)$
$p_{10} = (L = \{N1(''Temp''), N2(X_{SC1},'' SC'', Y_{SC1}), N3(X_{SC2},'' SC'', Y_{SC2})\},$ $K = \{N1(''Temp''), N2(X_{SC1},'' SC'', Y_{SC1}), N3(X_{SC2},'' SC'', Y_{SC2})\},$ $R = \{(N2 \xrightarrow{Comp} N3)\})$ $(X_{SC1}, Y_{SC1}, X_{SC2}, Y_{SC2} \in String)$
$p_{11} = (L = \{N1(''Temp'')\}, K = \{\ \}, R = \{\ \})$

FIG. 2.4 – Les productions de grammaire pour la description de l'architecture dynamique de l'exemple.

Les productions $p1$ et $p2$ permettent de prendre en compte la structure de connexion des différents types de composants : Un consommateur peut communiquer avec un service simple ou un service composite et un service composite peut contenir des composants de type service simple.

Les productions p_3 et p_4 permettent de considérer qu'un consommateur puisse communiquer respectivement avec plusieurs services simples et plusieurs services composites (propriété 4).

p_5 et p_6 spécifient qu'un consommateur peut partager des services composites avec d'autres consommateurs et partager des services simples avec d'autres consommateurs ou services composites (propriété 5). Cette assertion est aussi valable dans le sens inverse assurant qu'un service simple ou composite peut être connecté à des consommateurs qui communiquent avec d'autres services.

Les productions p_7 et p_8 spécifient respectivement qu'un composite peut contenir plusieurs services simples et plusieurs services composites (propriété 6).

Les productions p_9 et p_{10} caractérisent des propriétés similaires à celles relatives aux productions p_5 et p_6. Ces productions impliquent qu'un composite peut partager des services simples ou composites avec des clients ou avec d'autres composites (propriété 7).

La production p_{11} termine le processus de génération par l'élimination du nœud non terminal labélé par "*Temp*".

La grammaire de graphes GG respecte la contrainte 1 puisque les productions qui consomment l'axiome de la grammaire génèrent au moins un composant et que toutes les autres productions susceptibles d'être appliquées après cela ne font qu'augmenter la taille des graphes intermédiaires[17].

La propriété 2 est garantie par le fait que les deux productions (i.e. p_2 et p_3) qui génèrent des consommateurs génèrent aussi un premier service auquel le consommateur produit est connecté. L'absence de production de grammaire qui génère une lien entre un consommateur et un autre consommateur assure le respect de la propriété 3. La propriété 6 est garantie par le fait que toutes les productions de grammaire

[17] Dans le cas d'une spécification qui considère l'architecture vide comme consistante, on introduirait, en plus des productions de grammaire GG, une production de grammaire supplémentaire caractérisée par : $p = (L = \{N(AX)\}; K = \{\ \}; R = \{\ \})$.

(i.e. p3, p5 et p9) qui génèrent un service composite supplémentaire (i.e. où la partie R contient un nœud de type SC), génèrent aussi un service simple qu'il contient. L'absence de contraintes sur le déploiement des composants (propriété 9) est garantie par le fait que, dans toutes les productions, les variables (i.e. Y_C, Y_S, Y_{SC}) correspondant aux machines qui hébergent les composants sont toutes des variables libres.

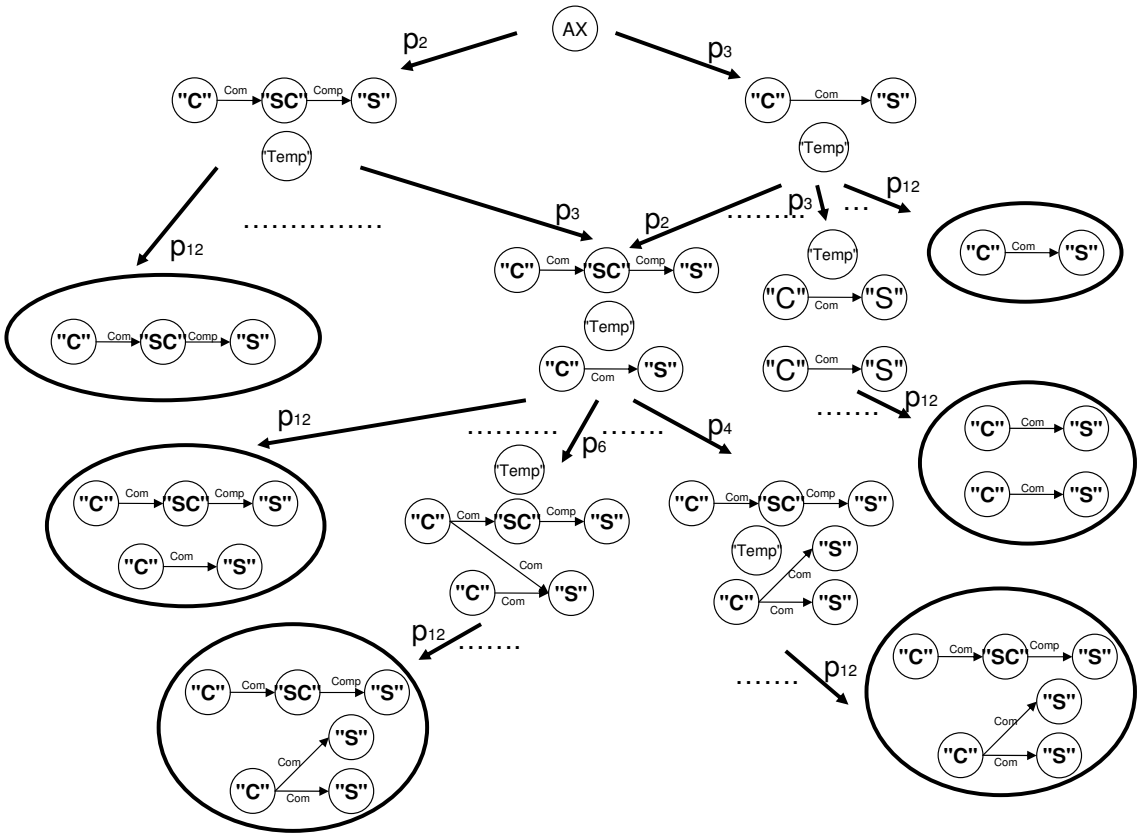


FIG. 2.5 – Un arbre de génération partiel pour le style architectural de l'exemple.

L'exemple présenté dans la figure 2.5 présente un arbre incomplet des instances générées par la grammaire de graphes GG. Pour ne pas surcharger la figure, nous présentons les nœuds avec uniquement le label correspondant au type du composant (i.e. $''C''$ pour les consommateurs, $''S''$ pour les services simples et $''SC''$ pour les services composites)[18]. Les instances consistantes de l'architectures sont celles qui appartiennent au langage défini par la grammaire contenant exclusivement des nœuds terminaux. Nous retrouvons une partie de ces instances au niveau des feuilles de cet arbre partiel.

2.3 Raffinement et abstraction des architectures

Pour comprendre, vérifier ou gérer les architectures dynamiques, plusieurs niveaux d'abstraction peuvent être pris en compte. Ces différents niveaux peuvent considérer des entités logicielles et des interdépendances architecturales différentes appartenant à des styles architecturaux différents. Nous considérons, ici, deux problématiques relatives à la transformation des descriptions d'architectures entre différents niveaux d'abstraction.

La première problématique concerne le raffinement des architectures permettant d'aller d'un modèle de description abstrait vers des modèles de description plus concrets. Ces niveaux d'abstraction peuvent,

[18] Les autres paramètres peuvent correspondre à n'importe quelle valeur appartenant au domaine de leur définition.

par exemple, correspondre aux différentes étapes du processus de développement logiciel. Dans une démarche de raffinement, on peut commencer par générer une spécification de haut niveau impliquant seulement les composants abstraits et leurs interdépendances. Puis ensuite, décrire la structure interne de ses composants pour obtenir des spécifications plus complexes et plus détaillées en se rapprochant de plus en plus de l'implémentation. Par exemple, dans une architecture orientées services, le niveau le plus haut peut correspondre aux services directement accessibles aux clients. Un raffinement de cette architecture décrirait les mécanismes d'orchestration et de chorégraphie permettant de composer et de coordonner les services de plus bas niveau.

La deuxième problématique concerne l'abstraction des descriptions permettant d'aller d'un modèle de description de bas niveau vers un modèle de plus haut niveau. L'objectif est d'alléger le modèle de description pour faciliter la compréhension de l'architecture et focaliser le modèle sur des aspects de l'architecture selon des points de vues spécifiques. Généralement cela permet de réduire la taille du modèle pour la validation et la vérification des propriétés architecturales relatives au niveau d'abstraction cible.

Nous introduisons des systèmes de transformation permettant le passage automatique d'un niveau d'abstraction vers un autre. Ces systèmes de transformation sont basés sur les grammaires de graphes et génèrent, en considérant le style architectural de départ appartenant à un niveau d'abstraction donné, pour toutes les instances de l'architecture, l'instance ou les instances qui lui correspondent dans le niveau d'abstraction cible. Dans ce qui suit, nous utiliserons le terme *transformation verticale* pour parler indistinctement du raffinement et de l'abstraction.

2.3.1 Le cadre formel

Les systèmes de transformation verticale peuvent considérer des composants et des relations d'interdépendance appartenant à différents niveaux d'abstraction. Ces systèmes doivent, par conséquent, permettre de traduire les composants ainsi que les liens appartenant au niveau d'abstraction de départ vers les composants et les liens considérés au niveau d'abstraction cible.

Nous étendons notre formalisme pour permettre cette traduction. Nous utilisons des grammaires de graphes de type edNCE qui se révèlent bien adaptées à ce type de problématique. Les nœuds non-terminaux (qu'il faut consommer) coïncident avec les nœuds correspondant aux composants du niveau d'abstraction de départ et les nœuds terminaux (que nous produisons) coïncident avec les nœuds considérés au niveau d'abstraction cible. Les instructions de connexion edNCE permettent de déduire les interdépendances au niveau d'abstraction cible à partir des interdépendances considérées dans le niveau de départ.

Nous étendons l'approche edNCE définie dans le chapitre 1 pour l'adapter au contexte de notre problématique. Nous considérons une structure de productions de grammaires similaire à celle définie pour la description des styles d'architecture et cela pour les mêmes raisons introduites précédemment (i.e. les productions de grammaire combinent la structure de l'approche DPO et intègrent la logique d'application SPO pour le traitement des arcs suspendus. Nous considérons aussi des règles paramétrées et des nœuds avec des labels variables. Ces productions de grammaire sont, en plus, étendues pour considérer des instructions de connexion qui génèrent les relations d'interdépendance pour le niveau d'abstraction cible.

Une dernière extension concerne l'introduction de conditions d'application négatives (CAN). Ceci augmente l'expressivité des systèmes de transformation verticale en permettant la spécification de motifs dont l'absence est requise pour l'application des productions de grammaire.

Nous obtenons, par conséquent, des productions de grammaire de type (L, K, R, N, C) où (L, K, R) correspond à la structure d'une production de type DPO, où N est le motif impliqué dans la définition de la CAN et où C contient les instructions de connexion.

Les instructions appartenant à C sont de type edNCE et sont spécifiées par un quintuplet $(n, \delta, p/q, d, d')$ avec n un nœud appartenant au graphe fille R, p et q des labels des arcs, δ est un label de nœud, et d et d' appartiennent à l'ensemble $\{in, out\}$.

Considérons une production de grammaire de ce type définie par le n-uplet $(L, K, R, N, \{(n, \delta, p/q, d, d')$ $\})$. Une telle production est applicable à un graphe G s'il contient une occurrence du graphe mère L. La condition d'application négative implique qu'il n'existe pas d'occurrence de N dans G.

L'application de cette production consiste à transformer le graphe G en supprimant l'image du sous graphe $Del = L \setminus K$ et en introduisant une copie homomorphe au sous graphe $Add = R \setminus K$ tandis que l'image du sous graphe K reste inchangée.

L'exécution de l'instruction de connexion implique l'introduction d'un arc entre le nœud n qui appartient au graphe fille $K \cup R$ et entre tous les nœuds n' qui sont p-voisins[19] et d-voisins du graphe mère $L \setminus K$. L'arc qui relie n et n' est introduit dans le sens d'. Cet arc est étiqueté par q.

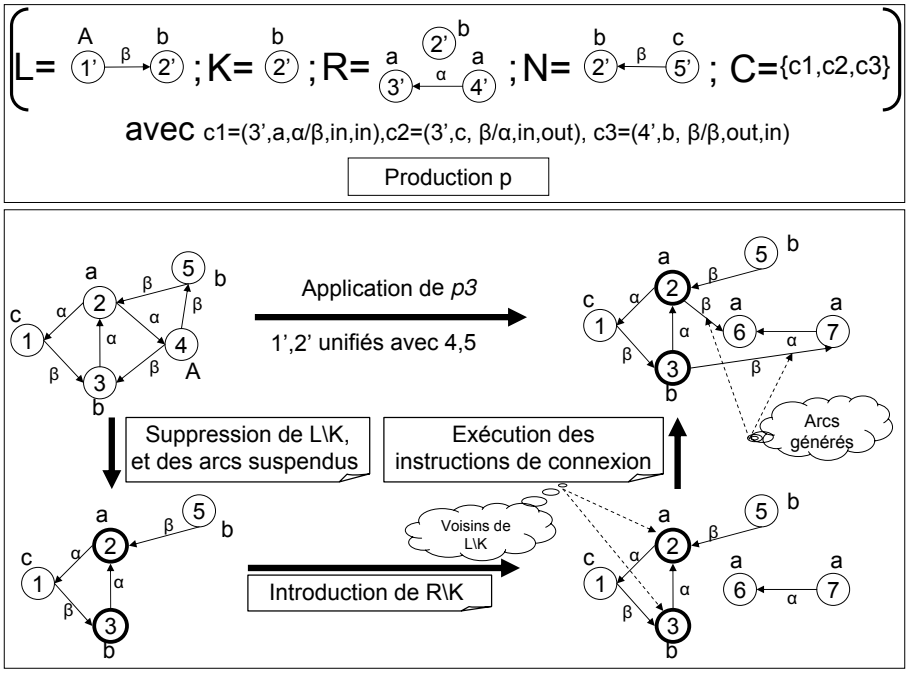


FIG. 2.6 – Exemple de l'application d'une production de grammaire appartenant au modèle de transformation verticale.

La figure 2.6 donne un exemple de l'application d'une production appartenant à notre modèle de transformation verticale. On note l'existence de deux occurrences de L dans le graphe G : le sous graphe contenant les nœuds 4 et 3 et le sous graphe contenant les nœuds 4 et 5. La condition d'application négative implique que le nœud du graphe correspondant au nœud $2'$ n'est connecté, avec un arc étiqueté par β, à aucun nœud dont le label est c. La première occurrence (i.e. qui considère les nœuds 4 et 3) n'est pas valide à cause du nœud 1 qui a comme label c et qui est connecté au nœud 3. La deuxième occurrence est, par contre, valide puisqu'aucun des voisins du nœud 5 n'est labélé par c.

Si nous considérons cette deuxième occurrence, l'application de la production implique, en plus des actions classiques comprenant la suppression de $L \setminus K$ et les arcs suspendus, et l'introduction de $R \setminus K$, la génération d'arcs selon les instructions de connexion spécifiées. L'instruction $c1$ concerne tous les α-voisins entrants du nœud 4 qui sont labélé par a. L'exécution de cette instruction introduit, donc, un arc labélé par β connectant le nœud 2 au nœud 6. L'instruction $c3$ concerne les β-voisins sortants du nœud 4 qui sont labélés par b. L'exécution de l'instruction $c3$ introduit un arc labélé par β entre les nœuds 3 et 7. L'instruction $c2$ est tout simplement ignorée parce qu'aucun voisin de $L \setminus K$ (voisins du nœud 4) n'est labélé par le label c.

[19] Pour rappel, les nœuds p-voisins d'un nœud n sont tous les nœuds n' tels qu'il existe une arête labélée p qui relie n et n'.

2.3.2 Exemple de transformation verticale, raffinement de la composition des Services Web

Dans cette sous section, nous illustrons les approches de raffinement et d'abstraction pour l'exemple générique de l'architecture orientée service. Nous considérons ici un raffinement dans le contexte des Services Web [ACKM04b, CDK+02, WS06]. Les consommateurs de services sont raffinés par des clients JAVA (type noté JAVA) alors que les services simples sont raffinés par des Services Web (type noté WS).

Nous envisageons deux raffinements pour les services composites [ACKM04a, MM04, LCG04] en prenant en compte l'orchestration et la chorégraphie [Pel03, Cho06b, Cho06a]. Par exemple, un service composite constitué de deux services sera raffiné par : 1) un processus BPEL [BPE06] qui orchestre deux Services Web, ou 2) deux Services Web respectant un protocole de chorégraphie pour leurs interactions.

Les liens de composition (notés *Comp* précédemment) sont raffinés pour obtenir le lien *Orch* dans le cas de l'orchestration et le lien *Chor* dans le cas de la chorégraphie. Les liens de communication sont raffinés par des canaux pour l'envoie de messages SOAP [SOA06] (type de liens noté SOAP).

La figure 2.7 donne les deux raffinements possibles (i.e. orchestration ou chorégraphie) pour un service composite avec deux services simples.

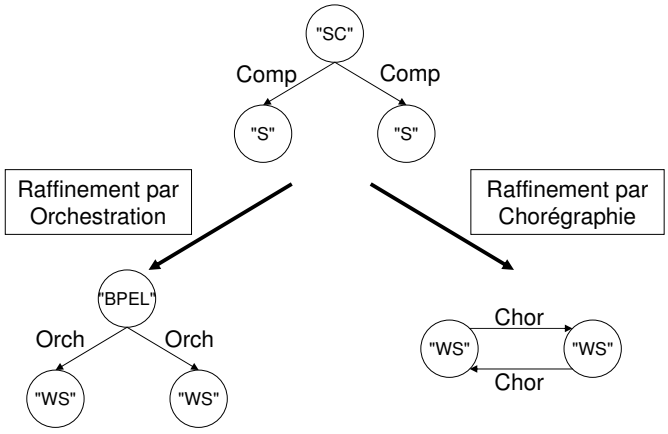


FIG. 2.7 – Raffinements d'un service composite.

Le raffinement consiste à définir un système permettant de traduire toutes les instances appartenant au style d'architecture défini par la grammaire de graphes GG (cf. section 2.2.2) vers les instances correspondantes au niveau d'abstraction considérant la composition des Services Web par l'orchestration et la chorégraphie.

Ce système de raffinement est caractérisé par la grammaire de graphes GG_r. Cette grammaire intègre les extensions introduites dans cette section et considère donc les conditions d'application négatives et les instructions de connexion.

L'axiome correspond au graphe de l'architecture G qui se situe au niveau d'abstraction le plus haut.

Pour notre contexte, l'ensemble des nœuds non-terminaux est constitué par les nœuds appartenant au niveau le plus haut (i.e. de types S, SC et C).

L'ensemble des nœuds terminaux est constitué par les nœuds du niveau d'abstraction le plus bas (i.e. WS, BPEL et JAVA).

Les instructions de connexion permettent de générer correctement les liens de communication, d'orchestration et de chorégraphie reliant les différents composants de l'application.

Convention 3 *Nous utiliserons, pour simplifier la présentation de la grammaire de graphes, la notation "$*$" pour désigner une valeur indéterminée. Une instruction de connexion de type $(n, *, p/q, d, d')$ concernera tous les voisins du graphe mère quelque soit leur label. Un nœud $N(*)$ désignera un nœud avec un nombre et un contenu de labels quelconques (i.e. ce nœud peut être unifié avec n'importe quel nœud).*

$GG_r = (G, NT, T, P)$ avec :
$NT = \{N(X_C,'' C'', Y_C), N(X_{SC},'' SC'', Y_{SC}), (X_S,'' S'', Y_S, T_S)\}$, et
$T = \{N(X_C,'' JAVA'', Y_C), N(X_B,'' BPEL'', Y_B), (X_S,'' WS'', Y_S, T_S)\}$, et
$P = \{p'_1, \ldots, p'_5\}$

$p'1 = (L = \{N1(X_C,'' C'', Y_C)\}; K = \{\ \}; R = \{N2(X_C,'' JAVA'', Y_C)\}; N = \{\ \}; C = \{i11\})$ $(X_C, Y_C \in String)$, et $i11 = (N2, *, Com/SOAP, out, out)$
$p'2 = (L = \{N1(X_{SC},'' SC'', Y_{SC})\}; K = \{\ \}; R = \{N2(X_{SC},'' BPEL'', Y_{SC})\};$ $N = \{\ \}; C = \{i21, \ldots, i27\})$ $(X_{SC}, Y_{SC} \in String)$, et $i21 = (N2, (X,'' C'', Y), Com/Com, in, in)$ $i22 = (N2, (X,'' JAVA'', Y), SOAP/SOAP, in, in)$ $i23 = (N2, (X,'' BPEL'', Y), Orch/Orch, in, in)$ $i24 = (N2, (X,'' SC'', Y), Contient/Contient, in, in)$ $i25 = (N2, *, Comp/Orch, out, out)$ $i26 = (N2, *, Chor/Chor, in, in)$ $i27 = (N2, *, Chor/Chor, out, out)$
$p'3 = (L = \{N1(X_{SC},'' SC'', Y_{SC}), N2(*), N3(*), N1 \xrightarrow{Comp} N2, N1 \xrightarrow{Comp} N3\};$ $K = \{N2(*), N3(*)\}; R = \{N2(*), N3(*), N2 \xrightarrow{Chor} N3, N3 \xrightarrow{Chor} N2\};$ $N = \{N1(X_{SC},'' SC'', Y_{SC}), N4(*), N4 \xrightarrow{Comp} N1\}; C = \{i31, \ldots, i34\})$ $(X_{SC}, Y_{SC} \in String)$, et $i31 = (N2, (X,'' C'', Y), Com/Com, in, in)$ $i33 = (N2, (X,'' JAVA'', Y), SOAP/SOAP, in, in)$ $i34 = (N2, *, Comp/Chor, out, out)$ $i35 = (N2, *, Comp/Chor, out, in)$
$p'4 = (L = \{N1(X_{SC1},'' SC'', Y_{SC1}), N2(X_{SC2},'' SC'', Y_{SC2}), N3(*), N4(*),$ $N1 \xrightarrow{Comp} N2, N2 \xrightarrow{Comp} N3, N2 \xrightarrow{Comp} N4\};$ $K = \{N1(X_{SC1},'' SC'', Y_{SC1}), N3(*), N4(*)\};$ $R = \{N1(X_{SC1},'' SC'', Y_{SC1}), N3(*), N4(*), N1 \xrightarrow{Comp} N3,$ $N1 \xrightarrow{Comp} N4, N3 \xrightarrow{Chor} N4, N4 \xrightarrow{Chor} N3\};$ $N = \{\ \};$ $C = \{i41, \ldots, i45\})$ $(X_{SC1}, Y_{SC1}, X_{SC2}, Y_{SC2} \in String)$, et $i41 = (N3, (X,'' C'', Y), Com/Com, in, in)$ $i42 = (N3, (X,'' JAVA'', Y), SOAP/SOAP, in, in)$ $i43 = (N3, *, Comp/Chor, out, out)$ $i44 = (N3, *, Comp/Chor, out, in)$ $i45 = (N1, *, Comp/Comp, out, out)$
$p'5 = (L = \{N1(X_S,'' S'', Y_S, T_S)\}; K = \{\ \}; R = \{N2(X_S,'' WS'', Y_S, T_S)\};$ $N = \{\ \}; C = \{i51, i52\})$ $(X_S, Y_S \in String)$, et $i51 = (N2, *, */*, in, in)$ $i52 = (N2, *, */*, out, out)$

FIG. 2.8 – Les productions de grammaire pour le raffinement des descriptions.

La production $p'1$ transforme un nœud de type consommateur (i.e. $''C''$) en un nœud de type client Java (i.e. $''JAVA''$). L'instruction de connexion i11 permet de traduire correctement les liens de communication vers le niveau le plus bas. Ainsi, tous les composants (services simples ou composites) connectés

par un lien de communication (i.e. de type $''Com''$) avec un consommateur doivent être connectés au nœud de type $''JAVA''$ par un lien de type $''SOAP''$.

La production $p'2$ permet d'introduire le raffinement par l'orchestration de la composition des Services Web. Le nœud représentant le service composite est raffiné par un nœud décrivant un processus BPEL qui implémente l'orchestration des services (composites ou simples) qu'il contient. Les instructions de connexion i21 et i22 traduisent le fait que les consommateurs du service composite (de type $''JAVA''$ ou les nœuds non terminaux de type $''C''$) seront connectés à l'orchestrateur BPEL. Les instructions i23 et i24 permettent de traduire le lien de composition entre les éventuels services composites ou orchestrateurs qui peuvent contenir le processus BPEL. L'instruction i25 permet de traduire le lien de composition vers les liens d'orchestration pour les services orchestrés par le processus BPEL. Dans le cas où le service composite à raffiner intervient dans une chorégraphie, les instructions i26 et i27, spécifient que c'est le processus BPEL qui prend son rôle dans cette chorégraphie.

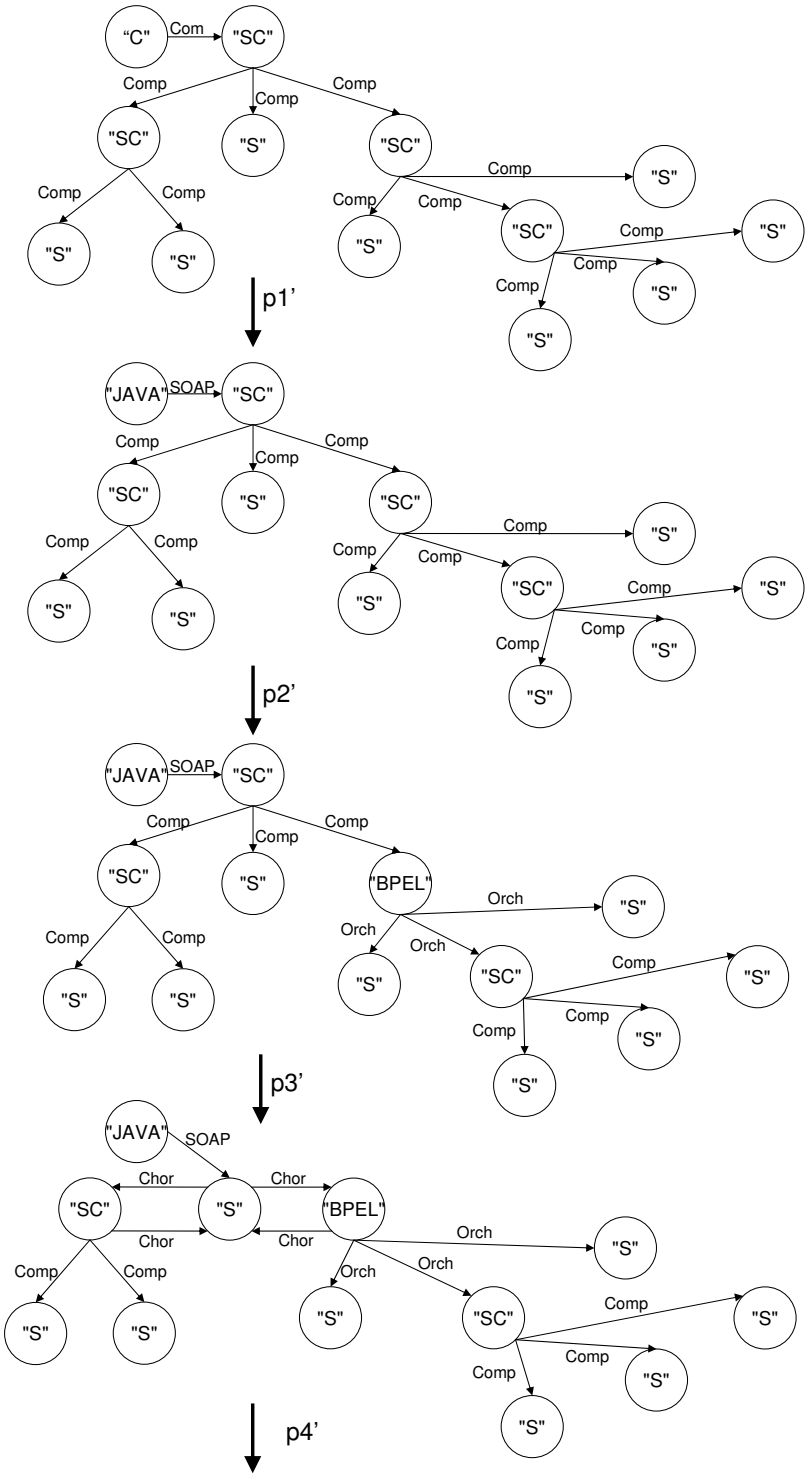
Les productions $p'3$ et $p'4$ traitent le raffinement de la composition par la chorégraphie. Grâce à l'utilisation de la condition d'application négative, nous distinguons deux cas.

Le premier cas (traité par $p'3$) concerne la situation où le service composite appartient au niveau de composition le plus haut (i.e. il n'est pas contenu dans un autre service composite). Dans ce cas, la composition est traduite par l'introduction d'une chorégraphie entre tous les éléments composant ce service. Ces liens sont donc bi-directionnels et sont représentés par deux arcs étiquetés par $''Chor''$. La spécification présentée ici implique une chorégraphie structurée en étoile (ici c'est le service représenté par le nœud N2 qui joue le rôle du centre de l'étoile). Les instructions i31 et i32 impliquent que les consommateurs du services composite envoient leurs requêtes vers le service situé au centre de l'étoile. Les instructions i33 et i34 permettent d'établir les liens représentant la chorégraphie entre les différents composants appartenant au service composite.

Le deuxième cas (traité par $p'4$) concerne la situation où le service composite appartient lui même à un autre service composite. Dans ce cas de figure, l'instruction i45 introduit, pour ce service composite de plus haut niveau, des liens de composition vers tous les composants de plus bas niveau (i.e. si C1 contient C2 qui contient deux services S1 et S2, alors cela pourrait être raffiné par C1 qui contient S1 et S2 en plus de l'introduction de la chorégraphie entre S1 et S2). Les instructions i41 jusqu'à i44 sont exactement identiques aux instructions i31 jusqu'à i34 et ont, par conséquence, la même incidence sur le système de transformation.

La production $p'5$ permet de traduire les service simples vers des nœuds représentant des services web. Les instructions de connexion préservent les liens de communication, d'orchestration et de chorégraphie (instructions $i51$ et $i52$).

La figure 2.9 présente un exemple de raffinement basé sur le système défini par la grammaire de graphes GG_r. Le graphe de départ appartient au style architectural correspondant au niveau d'abstraction le plus haut (en utilisant la grammaire de graphes GG et le chemin de génération : $p3; p9; p8; p9; p8; p9; p8; p8$). Le chemin de raffinement utilisé dans cette figure concerne la séquence d'application des productions de la grammaire GG_r : $p'1; p'2; p'3; p'4; p'2$ et ensuite p'5 appliquée huit fois.


"C"
Com
"SC"
Comp
"S"
p1'
"JAVA"
SOAP
p2'
"BPEL"
Orch
p3'
Chor
p4'

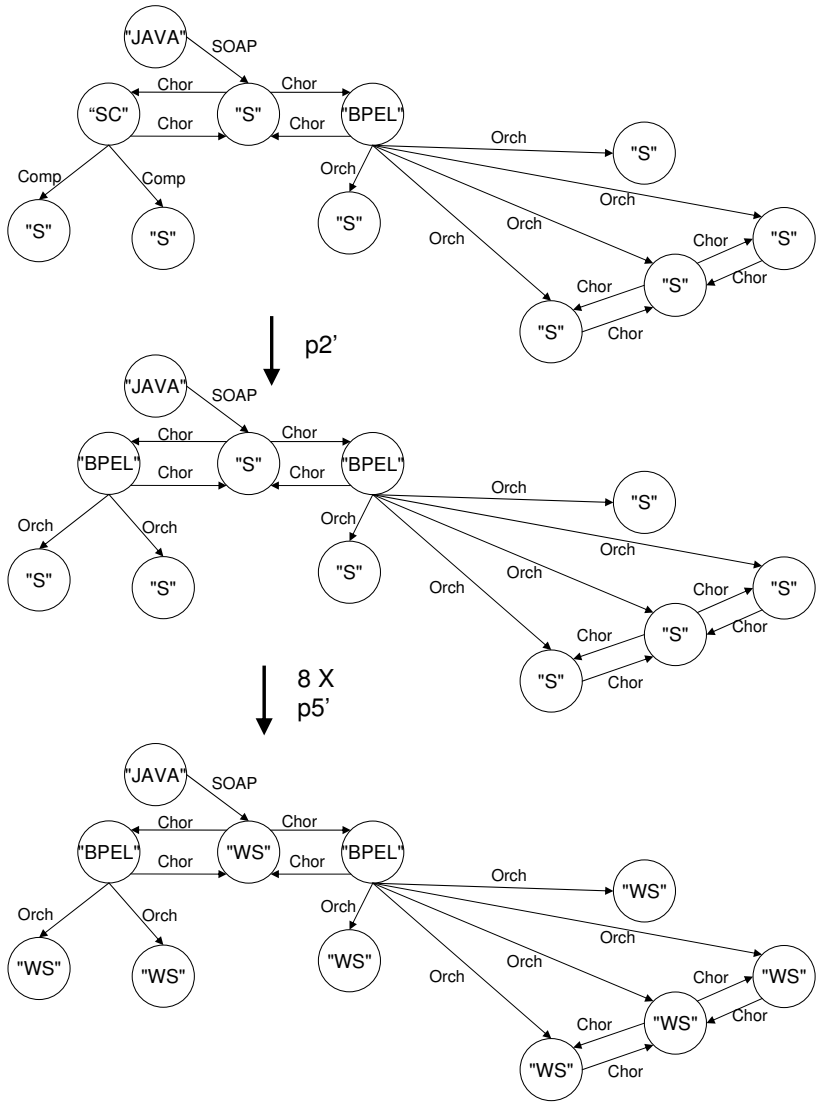


FIG. 2.9 – Exemple d'un chemin de raffinement en utilisant la grammaire GG_r.

2.4 Évolution dynamique de l'architecture

Dans les deux sections précédentes, nous avons abordé la problématique de la description des styles architecturaux permettant de spécifier l'ensemble des instances consistantes de l'architecture d'un système. Nous avons aussi défini les systèmes de transformation verticale pour raffiner et abstraire une description considérant un niveau d'abstraction donné vers les descriptions correspondantes dans un autre niveau d'abstraction. Dans les deux cas, il s'agit de définir des systèmes génératifs (i.e. des grammaires de graphes). Le premier cas correspond à produire l'ensemble des instances (i.e. le style architectural) de l'architecture tandis que le deuxième cas correspond à produire toutes les configurations qui raffinement ou qui abstraient une configuration appartenant à un niveau d'abstraction donné.

Dans cette section nous considérons une autre problématique. Il s'agit de spécifier les actions de reconfiguration à exécuter pour faire passer l'architecture d'une configuration donnée à une autre. Dans ce cas, il s'agit de définir un système permettant de transformer l'instance courante de l'architecture en produisant une nouvelle instance afin de répondre à la production d'un événement de reconfiguration.

Pour résumer, dans les deux premiers cas nous caractérisons un ensemble d'instances consistantes et dans le dernier cas nous caractérisons les transformations à réaliser pour passer d'une instance consistante vers une autre instance consistante.

La reconfiguration de l'architecture (que nous appelons aussi transformation horizontale) correspond à une transformation de la structure de l'application. Cette transformation peut impliquer l'introduction ou la suppression de composants et/ou la modification des interdépendances entre ces composants. L'objectif est d'adapter l'application pour répondre à un changement de son contexte (e.g. alarmes de sécurité, changements au niveau des ressources disponibles, pannes réseau), à de nouveaux objectifs applicatifs (e.g. passage d'une phase d'exécution vers une autre), ou à une modification du nombre ou du rôle applicatif de ses composants (e.g. intégration ou exclusion de participants dans une application coopérative).

Les règles de reconfiguration doivent maintenir l'application dans un état consistant. Contraindre la reconfiguration à suivre un protocole prédéfini la préserve de se retrouver dans un état inconsistant. Les règles de reconfiguration, exécutées à la suite de la réception d'un événement donné, doivent prendre en compte deux aspects principaux :

- Le premier aspect concerne la consistance de la reconfiguration elle même. Ceci revient à l'autoriser ou non selon qu'elle implique ou non une inconsistance de l'architecture. Les règles de reconfiguration doivent, donc, définir le contexte correct de leur application.
- Le deuxième aspect concerne la définition du processus de reconfiguration en spécifiant la combinaison d'actions élémentaires à exécuter (i.e. les composants et liens à introduire ou supprimer).

2.4.1 Cadre formel

Une instance de l'architecture est décrite par un graphe. Nous décrivons, donc, les règles de reconfiguration de l'architecture par une combinaison de règles de transformation de graphes. Ces règles de transformation correspondent au cadre formel défini pour la description. Elles traitent des nœuds mutilabélés et des arcs multi-étiquetés. Pour les mêmes besoins de généricité introduits pour les productions de grammaires, les règles de reconfiguration peuvent être paramétrées et considérer des labels variables.

La différence entre les productions de grammaire et les règles de transformation réside dans la prise en compte des notions de terminaux et de non-terminaux. Contrairement aux productions de grammaires, nous considérons pour la reconfiguration de l'architecture exclusivement des nœuds terminaux. Ceci est dû au fait que la configuration initiale et la configuration finale représentent toutes les deux une instance de l'architecture qui, par définition, ne contient que des terminaux.

À part la prise en compte des nœuds non terminaux, les règles de transformation de graphes correspondent en tous points (structure et sémantique) aux productions de grammaire présentées dans la section 2.3.1. Une règle de transformation est, donc, spécifiée par un quintuplet $(L; K; R; N; C)$.

Nous profitons, ainsi, de la possibilité d'introduire des conditions d'application négatives permettant de lier la reconfiguration de l'architecture à l'absence d'une sous configuration. Les instructions de connexion permettent de rediriger les interdépendances impliquant les composants à supprimer vers les composants à introduire. Ce dispositif se révèle très utile dans le cas d'un composant pouvant avoir un nombre indéterminé de composants voisins.

Nous donnons, dans la figure 2.10, une règle de transformation pour l'exemple des architectures orientées services. Cette règle correspond à une reconfiguration de l'architecture suite à une dégradation du temps de réponse d'un service composite $SC1$ par rapport aux requêtes reçues de la part d'un client $C1$. L'adaptation de l'architecture correspond à rediriger $C1$ vers un autre service composite $SC2$. Étant donné que la spécification de l'architecture impose qu'un service composite possède au moins un client, cette règle n'est applicable que si $SC1$ est connecté à au moins un autre client (client représenté par le nœud N2). D'un autre côté, cette redirection n'a de sens que si $C1$ n'est pas déjà connecté à SC2 (lien matérialisé par l'arc reliant N1 et N4).

La règle de transformation instanciée $R1(''C2'', X2,'' SC1'','' SC2'', M1, M2, M3, M4)$ est appliquée sur une instance de l'architecture dans la figure 2.11. Cette reconfiguration est consistante puisque : 1) Il existe au moins un homomorphisme entre la partie L de la règle et le graphe initial G (i.e. fonction injective affectant au nœuds N1,N2,N3,N4 respectivement les nœuds n2,n1,n4,n5) et 2) il n'existe pas d'arcs reliant les images des nœuds N1 et N4 (i.e. entre les nœuds n2 et n5). La reconfiguration de l'architecture implique, donc, la rupture du lien de communication entre le client $C2$ et le service composite $SC1$ (suppression

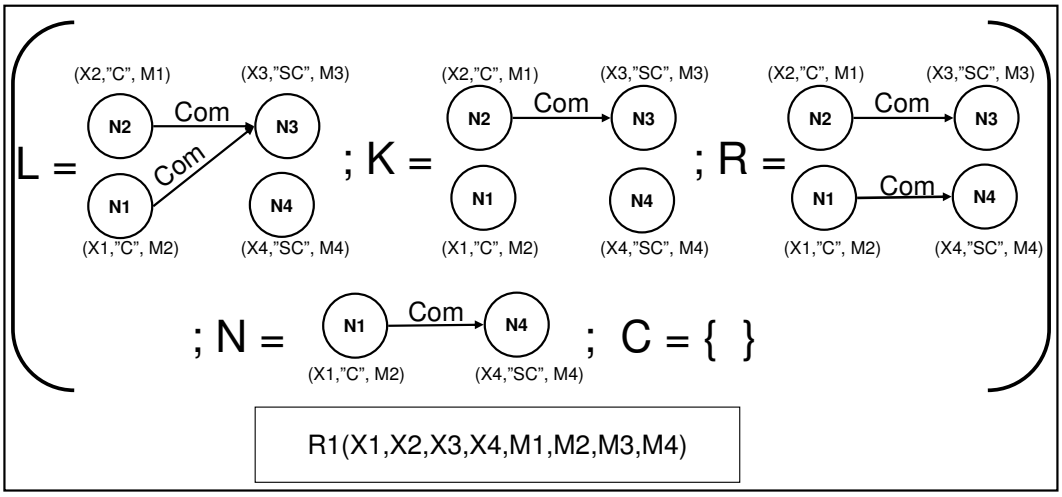


FIG. 2.10 – Règle de reconfiguration pour la redirection d'un client.

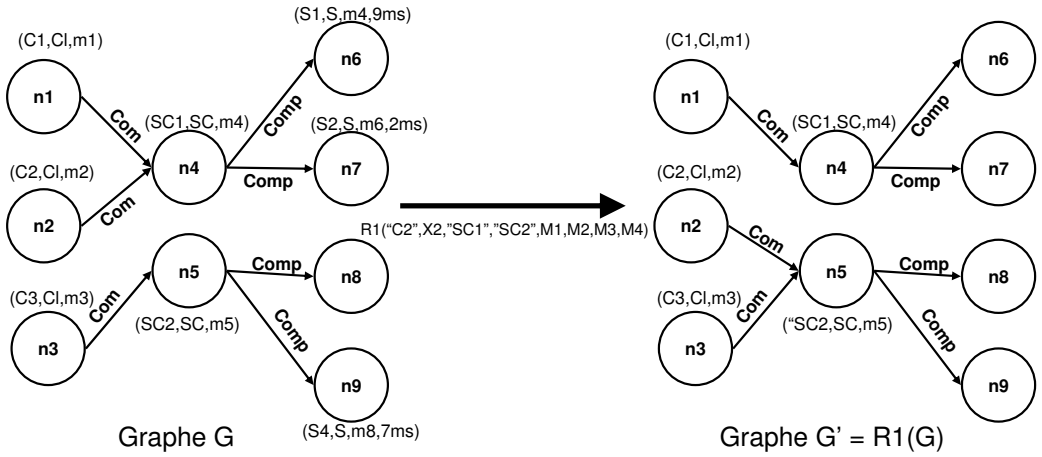


FIG. 2.11 – Application de la règle R1.

de l'arc reliant n2 et n4) et l'établissement d'un lien de communication entre le consommateur $C2$ et le service composite $SC2$ (introduction de l'arc reliant n2 et n5).

La deuxième règle présentée dans la figure 2.12 correspond à la suppression d'un service composite $X1$ qui est détecté en panne. Cette règle a pour but d'introduire un nouveau service composite équivalent $X2$ qui se substituera à $X1$ dans l'application. Le service initial $X1$ peut être connecté à un nombre indéterminé de consommateurs, peut contenir un nombre indéterminé de services simples ou composites, et peut composer un nombre indéterminé de services composites. Nous utilisons les instructions de connexion ($i1$ pour les consommateurs, $i2$ et $i3$ pour les services simples et composites contenus dans $X1$, et $i4$ pour les services composites auxquels il appartient) pour substituer $X1$ en gardant la même topologie de l'architecture.

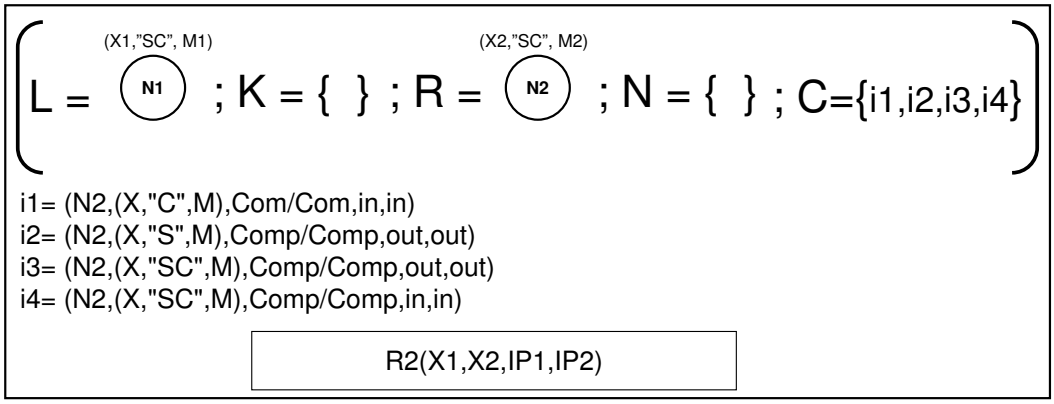


FIG. 2.12 – Règle de transformation pour la substitution d'un service.

La règle $R2(''SC2'',''SC3'',''M1'',''IP9'')$ est applicable à G' puisque les nœuds $N1$ et $n5$ sont unifiables. L'application de cette règle implique la suppression du nœud $n5$ et l'introduction d'une copie du nœud $N2$ (i.e. le nœud $n10$). Le fait que l'on considère l'approche SPO pour le traitement des arcs suspendus implique la suppression des arcs $n2 \xrightarrow{Com} n5$, $n3 \xrightarrow{Com} n5$, $n5 \xrightarrow{Comp} n8$ et $n5 \xrightarrow{Comp} n9$ alors que 1) l'instruction de connexion $i1$ introduit les liens de communication $n2 \xrightarrow{Com} n10$ et $n3 \xrightarrow{Com} n10$, et 2) l'instruction de communication $i2$ introduit les liens de composition $n10 \xrightarrow{Comp} n8$ et $n10 \xrightarrow{Comp} n9$.

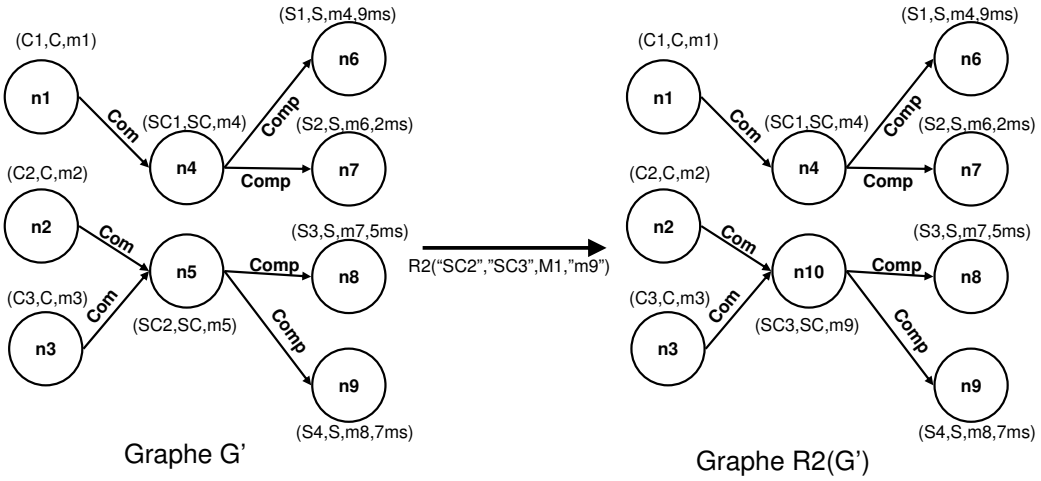


FIG. 2.13 – Application de la règle $R2$.

2.4.2 Le protocole de reconfiguration

Nous appelons, dans ce qui suit, protocole de reconfiguration de l'architecture le dispositif global permettant de gérer l'évolution dynamique de l'architecture. Ce protocole doit identifier les événements pertinents qui impliquent une reconfiguration et indiquer les règles de reconfiguration correspondantes.

La reconfiguration de l'architecture peut avoir différentes origines et poursuivre différents objectifs. L'élément déclencheur de l'action de reconfiguration, selon le point de vue du protocole de reconfiguration,

est toujours la production d'un événement qu'on appellera événement de reconfiguration.

Ces événements peuvent être générés par les différents composants du système ou par son environnement. Ils sont décrits par un type et peuvent contenir des paramètres qui amène une spécialisation des actions à réaliser. Ces paramètres peuvent, par exemple, préciser le type ou l'identité des composants concernés. Un événement est, donc, décrit par un couple *T(Params)* où T correspond au type de l'événement et où *Params* est l'ensemble des paramètres transportés par l'événement.

La règle de reconfiguration est définie par une combinaison de règles de transformation. Pour la spécification de ces combinaisons, nous considérons les opérateurs suivants :

- L'opérateur d'application séquentielle (noté ";") où une combinaison *c=r1;r2* implique : 1) si *r1* est applicable alors appliquer *r1* puis *r2*, 2) sinon la combinaison c n'est pas applicable.
- L'opérateur de disjonction (noté "||") où une combinaison *c=r1*||r2 implique : 1) si *r1* (respectivement *r2*) est applicable alors appliquer *r1* (respectivement *r2*) sinon, 2) appliquer *r2* (respectivement *r1*) si cette règle est applicable. Si ni *r1* ni *r2* ne sont applicables alors c n'est pas applicable.
- L'opérateur d'application multiple (noté "*") où l'application d'une combinaison *c=r** implique d'appliquer la règle r tant que cette règle est applicable[20]. La combinaison c est applicable si et seulement si r est applicable une fois.

La correspondance entre événements et combinaisons de règles de reconfiguration est caractérisée par un système d'actions de reconfiguration (T, C, I) où T est un type d'événement, C une combinaison de règles de transformation et I un ensemble de règles d'instanciation permettant d'instancier une partie des variables considérées dans les règles de transormation de C par les paramètres transportés par les événements de type T.

L'instanciation des règles de transformation correspond à une spécialisation des actions de reconfiguration. Une règle de reconfiguration relative à la suppression d'un service peut, par exemple, être spécialisée pour indiquer l'identifiant du service visé.

Une règle d'instanciation I est définie par un ensemble de paires I= $\{(v_1, v'_1), \ldots, (v_n, v'_n)\}$ où v_i correspond à une variable représentant un paramètre transporté par l'événement, et où v'_i est une variable considérée dans les labels des nœuds de la règle de reconfiguration.

Si nous considérons l'action de reconfiguration (T= $E(x1)$, C= $r1(X1, X2)$;$r2(X3)$, I= $\{(x1, X2); (x1, X3)\}$, la réception d'un événement $E(p1)$ implique l'instanciation de la combinaison C par l'affectation des variables $X1$ et $X3$ par la valeur $p1$ respectivement dans les règles r1 et r2. Si la règle instantiée $r1(X1, p1)$ est applicable, alors le protocole de reconfiguration transforme l'architecture en appliquant en séquence les règles $r1(X1, p1)$ et $r2(p1)$ instanciées. Si r1 n'est pas applicable, alors l'architecture n'est pas reconfigurée.

Le protocole de reconfiguration prend en compte les événements de reconfiguration, les règles de reconfiguration et les règles d'instanciation. Il est décrit formellement par un ensemble PR contenant un ensemble de triplets de la forme (T, C, I). À la réception d'un événement $E1 = T1(p1, ..., pn)$, le protocole de reconfiguration parcourt l'ensemble PR à la recherche du triplet correspondant au type $T1$. Si un tel triplet $(T1, C1, I1)$ existe, le protocole instancie la combinaison $C1$ par l'ensemble de paramètres $\{p1, ..., pn\}$ en suivant la règle d'instanciation $I1$ pour obtenir la combinaison instanciée $C1_i$. C'est cette combinaison qui sera appliquée pour la reconfiguration de l'architecture.

La figure 2.14 présente un protocole de reconfiguration partiel considérant les deux règles de reconfiguration $R1$ et $R2$. Cette figure donne un scénario de l'évolution de l'architecture suite à la réception d'événements de reconfiguration.

Les protocole de reconfiguration lie les événements de type *Redirect* à la règle de reconfiguration R1 et les événements de type *Substitute* à la règle de reconfiguration R2. Il indique, pour chaque règle, la procédure permettant d'affecter les paramètres reçus avec l'événement aux variables prises en compte dans la spécification des règles.

L'événement *Redirect*($''C3''$,$''SC2''$,$''SC1''$) ne peut conduire à la reconfiguration de l'architecture puisque la règle $R1$ instanciée selon le protocole de reconfiguration par les paramètres $''C3''$, $''SC2''$ et $''SC1''$ n'est pas applicable (i.e. puisque le service composite $''SC2''$ n'est pas relié à un consommateur

[20] L'opérateur $*$ est très utile dans la spécification des actions de reconfiguration. Cependant, à la différence des deux autres opérateurs, il peut impliquer des problèmes de non terminaison dans le cas où une règle est applicable à l'infini.

autre que $''C3''$).

L'événement *Redirect*($''C2''$,$''SC1''$,$''SC2''$) (R1 est applicable en considérant les paramètres transportés par l'événement) implique, par contre, une reconfiguration de l'architecture en supprimant le lien de communication entre le consommateur $''C2''$ et le service composite $''SC1''$ et en le remplaçant par le lien de communication entre $''C2''$ et le service composite $''SC2''$.

Les événements Substitute($''SC2''$,$''SC3''$,$''m9''$) et *Substitute*($''SC1''$,$''SC5''$,$''m1''$) entraînent respectivement la substitution des services composites $''SC2''$ et $''SC1''$ par les services $''SC3''$ et $''SC5''$ déployés respectivement sur les machines $m9$ et $m1$.

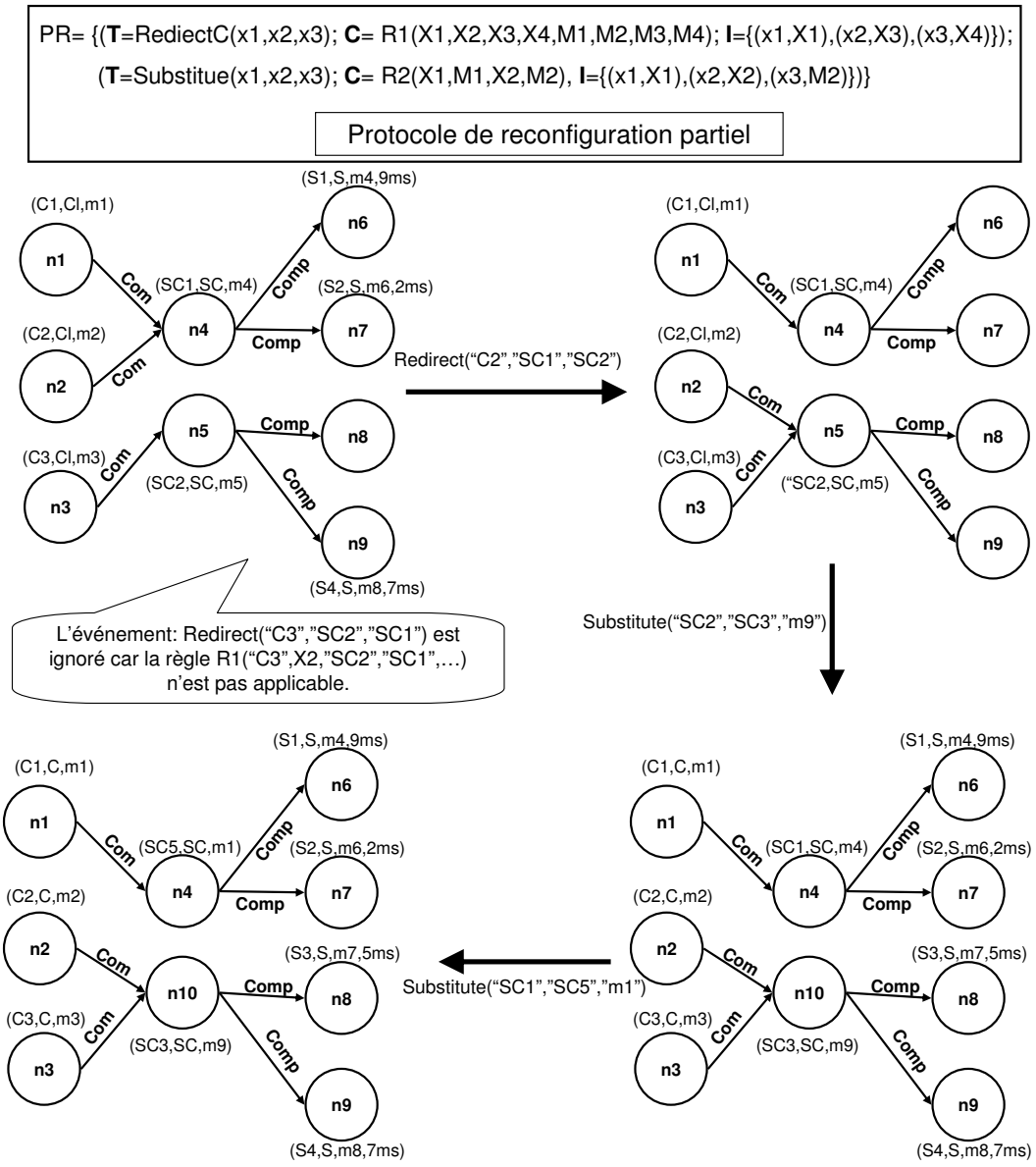


FIG. 2.14 – Scénario de traitement d'événements de reconfiguration.

2.5 Caractérisation des propriétés architecturales

Dans cette section, nous présentons un modèle permettant de caractériser des propriétés architecturales et de valider leur respect par une instance de l'architecture donnée. Nous utilisons, pour cela, deux concepts.

Le premier concept concerne ce que nous appelons les *propriétés positives*. Ce type de propriétés correspond à des contraintes qui impliquent l'existence d'une sous configuration pendant toute la durée de l'exécution : la configuration initiale ainsi que toutes les configurations résultantes de l'évolution de l'architecture doivent respecter cette contrainte. Un exemple de propriété positive, pour le cas du producteur-consommateur, est l'existence d'au moins un consommateur (i.e. l'architecture vide est non consistante) qui correspond à la contrainte numéro 1.

Le second concept concerne ce que nous appelons les *propriétés négatives* qui impliquent une condition qui stipule l'absence d'une sous configuration tout au long de l'exécution de l'application : cette sous configuration n'appartient pas à la configuration initiale et ne doit jamais apparaître suite à une reconfiguration de l'architecture. Pour l'exemple du producteur-consommateur, la contrainte numéro 2 qui impose qu'un consommateur communique avec au moins un service simple ou composite peut être caractérisée par l'absence d'une sous configuration correspondant à un consommateur isolé.

Les propriétés architecturales sont décrites sous la forme de règles de réécriture réduites où la vérification de leur validité dépend de l'applicabilité de ces règles.

Une propriété positive est violée si la règle correspondante n'est pas applicable. D'autre part, si une règle d'une propriété négative n'est applicable, alors cette dernière est vérifiée.

Nous gardons, pour la définition des propriétés, le modèle le plus étendu des règles correspondant à celui défini dans la section 2.3.1. Sachant que l'utilisation de ces règles est limitée à la vérification de leur applicabilité, nous nous passons des instructions de connexion [21]. La partie R est aussi ignorée et les zones L et K sont toujours égales pour préserver la neutralité des règles (i.e. elle ne transforme pas le graphe).

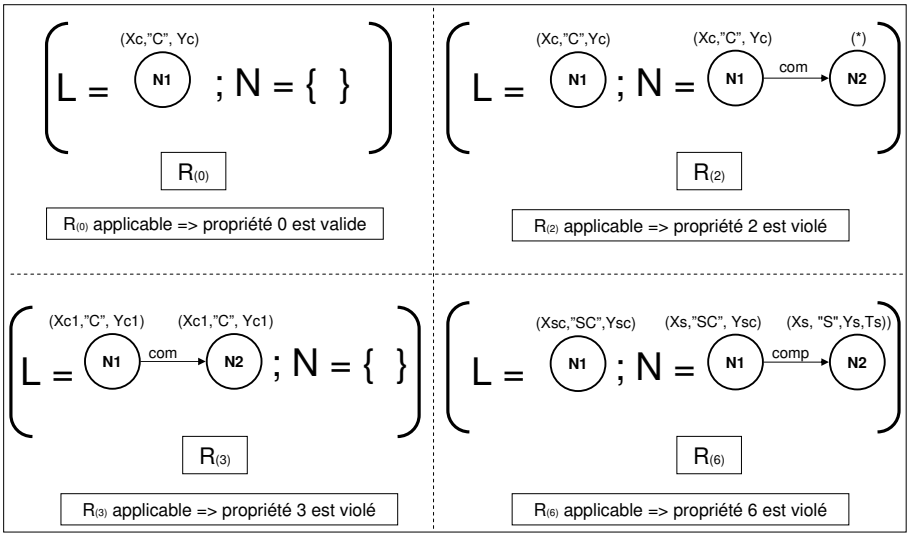


FIG. 2.15 – Règles positives et négatives caractérisant les propriétés architecturales de l'exemple du producteur consommateur

Nous obtenons des propriétés définies par des règles de réécriture $(L, K = L, R = \{\}, N)$. Sachant que la zone R est toujours vide et que L et K sont toujours égales, nous réduisons la description des propriétés à des règles de type (L, N).

[21] De toute manière, elles n'interviennent pas à l'étape de vérification de l'applicabilité mais seulement à l'application de la règle.

La vérification de ces propriétés peut être spécialisée en instanciant partiellement ou complètement leurs labels variables. e.g. vérifier que le consommateur $C2$ respecte la propriété 2 implique d'instancier la règle $R_{(2)}$ en affectant la valeur "$C2$" à la variable X_C.

Nous donnons, dans la figure 2.15, une caractérisation d'une partie des propriétés architecturales énoncés pour l'exemple du producteur-consommateur[22] dans la section 2.2.2.0.

La règle $R_{(0)}$ est la seul règle qui correspond à une propriété positive et son applicabilité assure la validité de la contrainte (0) qui considère qu'une architecture vide est non consistante.

La règle $R_{(2)}$ correspond à une propriété négative qui concerne la contrainte (2) stipulant qu'un consommateur ne peut être isolé et qu'il doit forcément communiquer avec un autre composant.

La règle $R_{(3)}$ correspond à la propriété négative qui concerne la contrainte (3). Son applicabilité viole cette contrainte qui implique que deux consommateurs ne peuvent être directement reliés par un canal de communication.

L'applicabilité de la règle $R_{(6)}$ implique l'existence d'un service composite qui ne contient aucun service simple ce qui violerait la contrainte (6) de la spécification de l'architecture.

2.6 Problématiques de validation et de vérification

Plusieurs approches peuvent être adoptées pour vérifier et valider nos modèles. La figure 2.16 présente une synthèse de ces approches où nous distinguons deux niveaux.

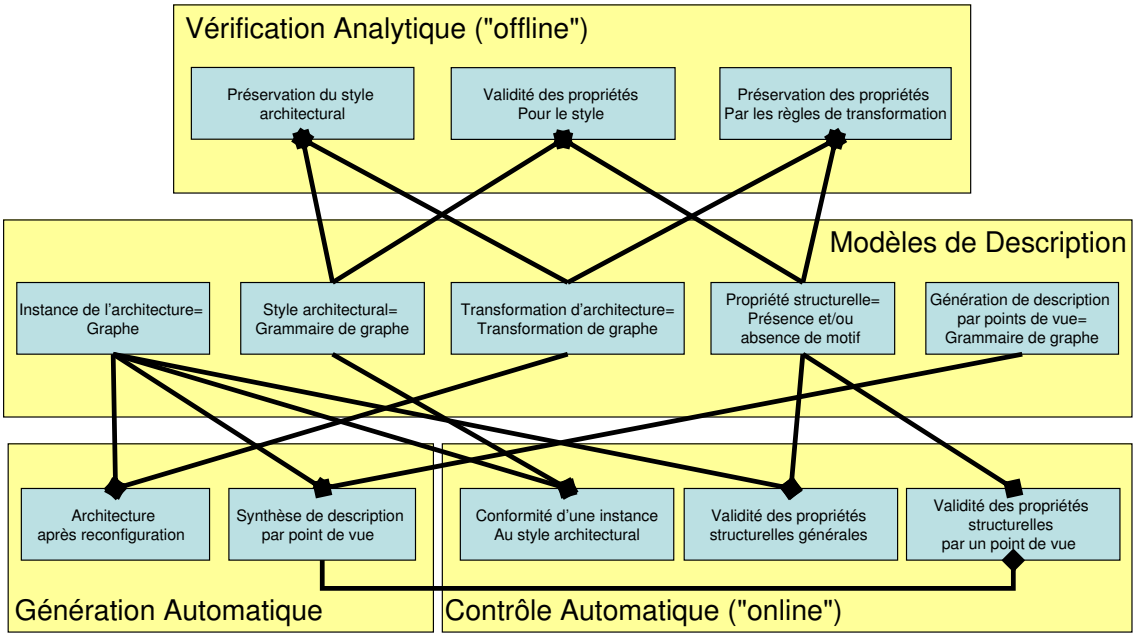


FIG. 2.16 – Problématiques de description et de vérification des architectures dynamiques en utilisant le méta-modèle.

[22]Les propriétés impliquant la possibilité d'une *configuration* (e.g. Un consommateur **peut** communiquer avec un service simple ou un service composite) ne peuvent être caractérisé par notre approche. Elles sont garanties par les productions de grammaire du style.

Le premier niveau correspond à la production de preuves analytiques réalisées à priori. L'objectif est de prouver la conformité du protocole de reconfiguration avec le style architectural. Cela revient à prouver qu'en appliquant n'importe quelle combinaison de reconfiguration à n'importe quelle instance appartenant au style architectural, nous obtenons une instance qui appartient toujours au style architectural. Nous avons développé dans [GDD06] une approche similaire en s'inspirant des travaux de LeMetayer [Met98]. Cette approche générique s'adresse à des modèles simplifiés appartenant à notre méta-modèle qui considèrent des règles de réécriture sans instructions de connexion, des combinaison réduite à une règle de réécriture mais avec des labels variables et des conditions d'application négatives. Pour des modèles qui considèrent toutes les extensions du méta-modèle, l'implémentation d'une telle approche générique se révèle très complexe. Par contre, une preuve au cas par cas est toujours envisageable.

À ce même niveau, une autre problématique concerne la vérification de la préservation des propriétés architecturales par le style d'architecture et par le protocole de reconfiguration. Le premier aspect implique de vérifier que toutes les instances produites par le style préservent les propriétés définies (les preuves informelles que nous avons présenté pour le style producteur-consommateur et les propriétés (1)...(9) peuvent être classées dans cette catégorie). L'autre aspect est de vérifier que toutes les instances produites par le protocole respectent les propriétés de l'architecture.

L'autre niveau concerne la vérification automatique et le contrôle de l'évolution de l'architecture pendant l'exécution en examinant la validité des instances produites par le protocole de reconfiguration. Ceci correspond à 1) vérifier l'appartenance de la configuration produite au style architectural (i.e. appartenance du graphe décrivant l'instance à la grammaire décrivant le style), et 2) vérifier en utilisant des outils de simulation (tels que ceux que nous présentons dans le chapitre 5) la validité des propriétés architecturales en vérifiant l'applicabilité des règles caractérisant les différentes propriétés positives et négatives (i.e. applicabilité des propriétés positives et non applicabilité des propriétés négatives au graphe décrivant l'instance produite).

Sachant que ce deuxième niveau concerne des vérifications réalisée pendant l'exécution des problématiques liées à la performance et au passage à l'échelle sont capitales et doivent être prises en compte pour la définition des outils de simulation et de contrôle.

2.7 Conclusion

Nous avons présenté, dans ce chapitre, un cadre formel pour le traitement des architectures dynamiques. Nous avons abordé trois problématiques principales et proposé des méta-modèles pour les considérer.

Dans un premier lieu, nous avons présenté un formalisme basé sur les grammaires de graphes pour la description des styles architecturaux permettant de caractériser, à travers un processus génératif, les instances considérées comme consistantes par la spécification de l'application.

Ensuite, nous avons évoqué la prise en compte de différents niveaux d'abstraction. Dans ce cadre, nous avons présenté un formalisme basé sur les grammaires de graphes edNCE permettant de spécifier les transformations verticales impliquant le raffinement et l'abstraction des descriptions.

Pour finir, nous avons introduit notre approche pour la gestion de l'évolution dynamique de l'architecture. Nous avons présenté les notions d'événements et de règles de reconfiguration. En combinant ces deux notions et en introduisant des mécanismes d'instanciation, nous avons présenté le protocole de reconfiguration permettant de contrôler et de gérer l'évolution de l'architecture.

Pour chaque problématique traitée, nous avons illustré nos approches en utilisant un exemple simple dans le cadre des applications orientées-services et en considérant la composition des services web par l'orchestration et la chorégraphie des services.

Nous présentons, dans le chapitre suivant, un cas d'étude plus élaboré auquel nous appliquons nos approches pour la description et la spécification des transformations verticales et horizontales. Ce cas d'étude se situe dans le cadre plus général des applications coopératives hiérarchiques et concerne plus spécifiquement les opérations d'intervention d'urgence. Il implique des besoins d'adaptabilité liés aux changements des objectifs applicatifs, à la panne des composants, et au provisionnement de la QdS.

Chapitre 3

Cas d'Étude, Activités Coopératives pour les Opérations d'Intervention d'Urgence

3.1 Introduction

Nous avons éprouvé notre approche pour les applications à architectures dynamiques pour différents cas d'étude. Nous avons, défini des modèles de description et de reconfiguration pour des applications coopératives telles que l'édition partagée [GDD05] et la revue coopérative [Del06a] en considérant des actions de reconfiguration pour la gestion des groupes de coopération et de la QdS. Nous avons aussi traiter le cas des systèmes coopératifs hiérarchiques pour la gestion de crise [CGD+06a, CGD+06b].

Dans ce chapitre, nous considérons une version étendue du dernier cas d'étude en prenant en compte un niveau hiérarchique supplémentaire et des actions de reconfiguration consécutives à l'occurrence de pannes et au provisionnement de la QdS. Nous nous situons dans le contexte des activités d'opération d'intervention d'urgence (OIU). L'application implique des groupes structurés de robots ou de personnel militaire qui coopèrent pour la réalisation d'une mission commune. Les éléments de l'architecture possèdent différents rôles et disposent de ressources inégales en capacités de communication, en CPU et en énergie. Ils sont déployés sur des machines fixes et mobiles et communiquent via des réseaux filaires et sans fil.

L'activité admet deux phases d'exécution correspondant à une phase d'exploration du champ d'investigation et à une phase d'action faisant suite à la découverte d'une situation critique. Les rôles dans l'application et la structure des interactions entre participants évoluent d'une étape d'exécution à l'autre pour s'adapter à l'évolution des objectifs applicatifs et du contexte d'exécution.

Trois niveaux d'abstraction sont identifiés en se référant aux trois couches application, middleware et réseau relevant des couches hautes et basses du modèle OSI. Cependant, le principe d'indépendance des couches est remis en cause en autorisant la prise en compte, au niveau considéré, d'informations dont la sémantique est d'un niveau différent ; c'est le principe de base du *cross layering* où une réorganisation affectant un niveau de l'architecture peut se traduire par une réorganisation à un autre niveau architectural.

Les composants logiciels, qu'ils relèvent des couches application, middleware ou réseau auront à prendre en compte des besoins multiples et évolutifs dans le temps liés à l'activité ciblée, à la mobilité des utilisateurs, aux flux de données échangés (audio, vidéo et texte) et aux contraintes de l'environnement de communication. De plus, l'évolution de la mission induira inévitablement des changements dans la hiérarchie et dans la structure des coopérations entre intervenants : par exemple, suite aux informations acquises par les investigateurs sur le terrain ou par les décisions du contrôleur et des coordinateurs de la mission.

Dans ce qui suit, nous décrivons le style architectural correspondant à chaque niveau d'abstraction

pour chaque phase d'exécution. Nous définissons, aussi, les systèmes de raffinement et d'abstraction permettant le passage automatique d'une description appartenant à un niveau d'abstraction vers la ou les descriptions correspondantes à un autre niveau d'abstraction. D'autre part, nous décrivons le protocole de reconfiguration qui gère l'évolution dynamique de l'architecture et nous identifions et caractérisons des propriétés architecturales que l'application doit observer en considérant les différents niveaux d'abstraction .

3.2 Description de l'architecture au niveau application

Le niveau application (A) constitue le plus haut niveau d'abstraction considéré dans notre étude. Il décrit les rôles des participants et les types de données de haut niveau échangées entre les participants ainsi que les exigences et contraintes relatives à la structure dynamique de l'architecture globale.

Pour notre exemple, la coopération est basée sur l'échange de données entre des participants, notamment des données d'observation et des données d'analyse, produites périodiquement ou immédiatement après un événement particulier.

Une équipe d'intervention d'urgence est ainsi constituée de participants ayant différents rôles : un contrôleur de la mission, plusieurs coordinateurs, et plusieurs sections d'investigateurs. Le contrôleur de la mission gère l'ensemble des coordinateurs et chaque coordinateur dirige une section d'investigateurs. À chaque rôle correspondent les fonctions suivantes :

- Un contrôleur a pour fonction de diriger et d'autoriser les actions qui sont déléguées aux coordinateurs. Le contrôleur est l'entité qui supervise toute l'application, il attend des rapports de tous ses coordinateurs qui synthétisent le contexte courant de l'application et l'informent du déroulement de la mission. Le coordinateur est déployé sur une machine fixe, il dispose d'un accès à l'énergie permanent et de capacités de communication et de CPU conséquentes.
- Selon les actions et les objectifs assignés par le contrôleur, un coordinateur doit diriger ses investigateurs en leur assignant des tâches à exécuter. Il doit également collecter, interpréter et synthétiser les informations reçues des investigateurs et les diffuser vers le contrôleur. Les coordinateurs sont eux aussi déployés sur des machines fixes.
- Les investigateurs ont pour fonction d'explorer le champ opérationnel, d'observer, d'analyser et de faire un rapport décrivant la situation aux coordinateurs qui les contrôlent. Ils sont déployés sur des machines mobiles et disposent, donc, de ressources limitées en énergie et en CPU.

Les fonctions assignées aux participants impliquent d'Observer (O) le champ d'investigation et de Rapporter (R) sur ce qui est observé. Les données de retour O sont des données descriptives tandis que les données de retour R sont des données produites et expriment l'analyse de la situation par un investigateur ou un coordinateur.

Le contrôleur supervise l'ensemble de la mission, en décidant des actions à exécuter en fonction des objectifs opérationnels et de l'analyse des retours R transmis par les coordinateurs. Il possède sous ses ordres, au moins un coordinateur, et chaque coordinateur possède au moins un investigateur. Un coordinateur est en charge de la partie de la mission qui lui a été assignée par le contrôleur. Il décide localement des actions à exécuter en fonction de l'observation et de l'analyse des données O transmises par les investigateurs. Pour prendre cette décision, le coordinateur peut également utiliser les données R transmises par les investigateurs. Les coordinateurs rapportent l'évolution de la sous mission au contrôleur en utilisant des données de retour de type R.

3.2.1 Le style architectural correspondant à la phase d'exploration

En phase d'exploration, l'exécution de la mission implique les échanges de données suivants :

- Les investigateurs envoient de manière continue des données O au coordinateur. Ils envoient également des données R périodiquement.
- Les coordinateurs envoient périodiquement des rapports de type R exprimant la situation courante de l'investigation.

Le style architectural induit par ces spécifications, implique la prise en compte des composants décrits par les nœuds et les labels suivants :

- Le composant contrôleur est décrit par un nœud *N("Co","Cont","M")* où *"Co"* correspond à l'identifiant de ce composant, *"Cont"* spécifie que le nœud décrit un composant de type contrôleur et "M" correspond à la machine fixe sur laquelle le composant est déployé.
- Un coordinateur est décrit par un nœud *N("C","Coord","φ_1","M","R")*, où *"C"* correspond à l'identifiant du composant, où *"Coord"* correspond au type coordinateur et où *"M"* correspond à la machine du coordinateur décrit. L'attribut *"φ_1"* indique que le coordinateur est en phase d'exploration alors que l'attribut *"R"* spécifie que le coordinateur produit des données de type R.
- Un composant investigateur est décrit par un nœud *N("I","Inv","M","O+R")* où *"I"* correspond à l'identifiant du composant, l'attribut *"Inv"* indique qu'il s'agit d'un investigateur, *"M"* correspond à la machine sur laquelle il est déployé. L'attribut *"O+R"* indique qu'il produit des données de type O et de type R.

Un exemple d'une instance de l'architecture est donné dans la figure 3.1. Cette instance comprend, en plus du contrôleur, deux coordinateurs qui coordonnent respectivement trois investigateurs et deux investigateurs. Les arcs de ce graphe sont étiquetés par le type de données échangées.

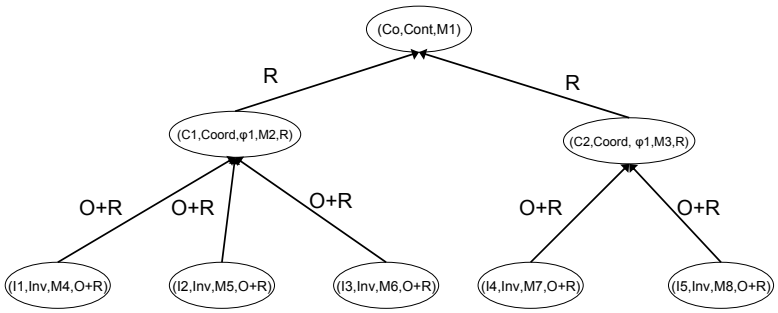


FIG. 3.1 – Exemple de l'architecture dans la phase d'exploration

Tous les paramètres considérés dans le cas d'étude sont de type *String*. Pour alléger les descriptions, nous omettons d'y spécifier le type des paramètres qui correspond, donc, par défaut au type *String*. En se basant sur les spécifications et les conventions de notation introduites précédemment, la grammaire de graphe $GG_{A,e}$ décrit le style architectural correspondant au niveau d'abstraction application (A) et à la phase d'exploration (e).

$GG_{A,e}$=(AX,NT,T,P) avec :
T={N(X_I,"Inv",M_I,"O+R"), N(X_{Coord},"Coord","φ_1",M_{Coord},"R"), (X_{Cont},"Cont",M_{Cont})}, et
NT={N("Temp")}, et
P={p_1,...,p_4}

p_1= (L={AX};K={ }; R={N1(X_{Cont},"Cont",M1), N2("Temp"), N3(X_{Coord},"Coord",φ_1,M2,"R"), N3$\xrightarrow{R}$N2, N4(X_I,"Inv",M3,"O+R"), N4$\xrightarrow{O+R}$N3})
p_2= (L={N1(X_{Cont},"Cont",M1) , N2("Temp")}; K={N1(X_{Cont},"Cont",M1) , N2("Temp")}; R={N3(X_{Coord},"Coord",φ_1,M2,"R"), N3$\xrightarrow{R}$N2, N4(X_I,"Inv",M3,"O+R"),N4$\xrightarrow{O+R}$N3})

p_3= (L={N1(X_{Coord},"Coord",φ_1,M1,"R") , N2("Temp")} ; K={N1(X_{Coord},"Coord",φ_1,M2,"R") , N2("Temp")} ; R={N3(X_I,"Inv",M3,"O+R"),N3$\xrightarrow{O+R}$N2})
p_4= (L={N("Temp")} ;K={ } ;R={ })

FIG. 3.2 – Les productions de grammaire pour la description de l'architecture au niveau application et en phase d'exploration.

La production p_1 permet de générer le composant représentant le participant qui joue le rôle de contrôleur de la mission (nœud N1). Ce participant dont l'identifiant est X_{Cont} est déployé sur une machine M1. Étant donné, dans la spécification, que l'application contient au moins un coordinateur et un investigateur, la production p_1 génère un coordinateur (nœud N3) qui est dans la phase φ_1 et qui est déployé sur une machine M2 en produisant des informations de type rapport (type "R"), et un investigateur (nœud N4) déployé sur une machine M4 et produisant des informations observées et rapportées (de type "O+R"). Les différents composants sont connectés selon la spécification de l'application (i.e. de l'investigateur vers le coordinateur, et du coordinateur vers le contrôleur).

La production p_2 permet, à chaque fois qu'elle est appliquée, de générer un coordinateur supplémentaire. L'application de cette production exige l'existence préalable d'un contrôleur[23] (le nœud N1 dans le sous graphe L). Elle implique l'introduction d'un contrôleur dans la phase φ_1 (nœuds N2) déployé sur une machine M2. Du fait qu'un coordinateur possède au moins un investigateur, on génère aussi un investigateur (nœud N4) déployé sur une machine M3 qui sera connecté au coordinateur introduit.

La production p_3 permet de générer, à chaque fois qu'elle est appliquée, un investigateur supplémentaire. Cet investigateur est contrôlé par le coordinateur introduit dans la partie L de la production (nœud N1). Le lien reliant les deux services correspond au type de données $O + R$.

La production p_4 permet de terminer le processus de génération en éliminant le seul nœud non terminal N("Temp").

L'instance de l'architecture dynamique, décrite dans la figure 3.1, peut [24] être obtenue par le chemin de génération : p_1 permettant de générer le non terminal *Temp*, le contrôleur Co, le coordinateur $C1$, l'investigateur $I1$ et les liens de communication qui les relient ; p_2 permettant de générer le coordinateur $C2$ et l'investigateur $I4$ et les liens de communication qui les relient ; p_3 permettant de générer l'investigateur $I2$ et de le relier au coordinateur $C1$; p_3 permettant de générer l'investigateur $I3$ et de le relier à $C1$; p_3 qui génère l'investigateur $I5$ et le connecte au coordinateur $C2$; puis finalement la production p_4 pour éliminer le non terminal *Temp* et obtenir une instance terminale. L'existence de ce chemin de génération prouve l'appartenance de cette instance au style architectural décrit.

Propriétés architecturales

Nous présentons, ici, la caractérisation des propriétés architecturales au niveau application et en phase d'exploration. Nous utilisons les modèles définis dans le chapitre précédent selon la structure des règles de transformation réduites de type $(L; N)$.

$pr_{A,1}$= (L={N(X_{Cont},"Cont",M)} ; N={ }), Pos
$pr_{A,2}$= (L={N1($X1_{Cont}$,"Cont",M1), N2($X2_{Cont}$,"Cont",M2)} ; N={ }), Neg
$pr_{A,3}$= (L={N1(X_I,"Inv",M1,"O+R")} ; N={N2(X_{Coord},"Coord",φx,M2,"R"), N1$\xrightarrow{O+R}$N2}), Neg
$pr_{A,4}$= (L={N1(X_I,"Inv",M1,"O+R"), N2(X_{Coord},"Coord",φx,M2,"R"), N1$\xrightarrow{O+R}$N2 N3(X_{Coord},"Coord",φx,M3,"R"), N1$\xrightarrow{O+R}$N3} ; N={ }), Neg

[23] Ce qui implique que p_2 ne peut être exécutée avant p_1.
[24] Il existe plusieurs chemins de génération pour une même instance.

$pr_{A,5}$= (L={N1(X_{Coord},"Coord",φx,M1,"R")} ; N={N2(X_{Cont},"Cont",M2), N1$\xrightarrow{R}$N3}), Neg
$pr_{A,6}$= (L={N1(X_{Coord},"Coord",$\varphi 1$,M1,"R")} ; N={N2(X_I,"Inv",M2,"O+R"), N2$\xrightarrow{O+R}$N1}, Neg

FIG. 3.3 – Les propriétés architecturales au niveau application et en phase d'exploration.

La figure 3.3 caractérise les propriétés positives et négatives de l'application dont la sémantique est donnée dans ce qui suit : 1) la propriété positive $pr_{A,1}$ exprime le fait que l'application possède toujours au moins un contrôleur, 2) la propriété négative $pr_{A,2}$ spécifie que l'application contient au plus un contrôleur. La vérification de la validité des deux propriétés (applicabilité de $pr_{A,1}$ et non applicabilité de $pr_{A,2)}$ assure la propriété d'unicité du contrôleur. 3) La propriété négative $pr_{A,3}$ implique qu'un investigateur est forcément relié à un coordinateur, alors que 4) la propriété $pr_{A,4}$ vérifie qu'un investigateur ne peut être coordonné par deux coordinateurs[25], 5) la propriété $pr_{A,5}$ implique que tous les coordinateurs sont toujours reliés au contrôleur, et finalement 6) la propriété $pr_{A,6}$ caractérise le fait que chaque coordinateur possède au moins un investigateur.

3.2.2 Le style architectural correspondant à la phase d'action

La phase d'exploration se termine quand une situation critique est découverte par un investigateur $I1$. Cette situation critique implique une reconfiguration de l'architecture et l'application bascule dans une autre phase d'exécution appelée phase d'action[26]. La reconfiguration touche spécifiquement le coordinateur *Coord1* qui contrôle l'investigateur $I1$ ainsi que tous les investigateurs de *Coord1*. Les autres coordinateurs et les investigateurs qui leurs sont rattachés ne sont pas affectés par cette reconfiguration. Dans ce qui suit, nous désignerons l'investigateur critique par le terme investigateur α et les autres investigateurs appartenant à la même section par le terme investigateurs β. Nous exigeons, pour le passage vers la phase d'action, l'existence d'au moins deux investigateurs : l'un jouera le rôle α alors que le second jouera le rôle β.

Lors du passage de la phase d'exploration vers la phase d'action, le coordinateur *Coord1* ainsi que ses investigateurs réagissent de la manière suivante :

- Après avoir découvert la situation critique, $I1$ continue de remplir les mêmes fonctions que celles qu'il réalisait durant l'étape d'exploration (Observer et Rapporter), mais envoie aussi ses observations O aux autres investigateurs. Il continue d'envoyer les deux types de données (R et O) au coordinateur.
- Les autres investigateurs rapportent maintenant seulement les données R au coordinateur ; elles correspondent à des analyses complémentaires des données O transmis par l'investigateur $I1$;

Le style architectural au niveau application pendant la phase d'action est donné par la grammaire de graphe $GG_{A,a}$. Cette grammaire considère les nœuds et les labels suivants :

- Les composants correspondant au contrôleur et aux coordinateurs dans la phase d'exploration sont respectivement décrits, tels que pour la grammaire $GG_{A,e}$, par un nœud *N("C","Cont","M")* et un nœud *N("Co","Coord","φ_1","M","R")*.
- Un coordinateur en phase d'action est décrit par un nœud *N("Co","Coord","φ_2","M","R")*, où tous les labels gardent la même signification que la description du coordinateur en phase d'exploration sauf pour le label indiquant "φ_2" qui exprime le fait que le coordinateur est en phase d'action.
- L'investigateur α est décrit par un nœud *N("I","Inv_α","M","O+R")* où "I" correspond à l'identifiant du composant, l'attribut "Inv_α" indique qu'il s'agit d'un investigateur α, "M" correspond à la machine sur laquelle il est déployé, et l'attribut "$O+R$" indique qu'il produit des données de type O et de type R.
- Un investigateur β est décrit par un nœud *N("I","Inv","M","R")* où "I" correspond à l'identifiant du composant, l'attribut "Inv" indique qu'il s'agit d'un investigateur, "M" correspond à la machine

[25] La combinaison des deux propriétés $pr_{A,3}$ et $pr_{A,4}$ implique que chaque investigateur est coordonné par un coordinateur unique.

[26] L'application est considérée dans une phase d'action dès qu'il y'a un coordinateur qui est dans cette phase. Ceci n'exclut pas qu'il y aie d'autres coordinateurs qui sont encore en phase d'exploration.

sur laquelle il est déployé, et l'attribut "R" indique que ce composant produit exclusivement des données de type R.

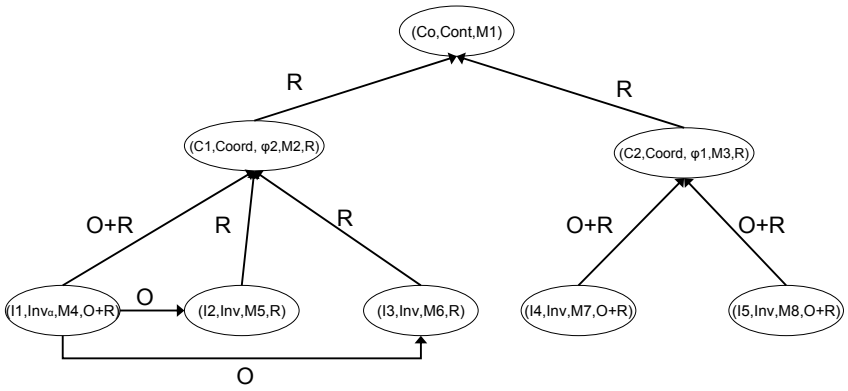


FIG. 3.4 – Exemple de l'architecture dans la phase d'action

La Figure 3.2.2 présente une instance de l'architecture correspondant au niveau application et à la phase d'action. Le coordinateur $C1$ est en phase d'action tandis que $C2$ est en phase d'exploration. Le rôle d'investigateur α est joué par $I1$ alors que $I2$ et $I3$ sont des investigateurs β.

$GG_{A,a}$=(AX,NT,T,P) avec :
T={N(X_I,"Inv_α",M_I,"O+R"), N(X_I,"Inv",M_I,"R"), N(X_I,"Inv",M_I,"O+R") ,
N(X_{Coord},"Coord","φ_1",M_{Coord},"R"), N(X_{Coord},"Coord","φ_2",M_{Coord},"R"),
N(X_{Cont},"Cont",M_{Cont})}, et
NT={N("Temp")}, et
P={p_1,...,p_6}

p_1= (L={AX} ;K={ } ; R={N1(X_{Cont},"Cont",M1), N2(X_{Coord},"Coord","φ_2",M2,"R") , N3(X_{I1},"Inv_α",M3,"O+R"), N4(X_{I2},"Inv",M4,"R"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{O+R}$N2, N4$\xrightarrow{R}$N2, N3$\xrightarrow{O}$N4, N5("Temp")})
p_2= (L={N1(X_{Cont},"Cont",M1) , N2("Temp")} ; K={N1(X_{Cont},"Cont",M1) , N2("Temp")} ; R={N3(X_{Coord},"Coord","φ_2",M2,"R"), N4(X_{I1},"Inv_α",M3,"O+R"), N5(X_{I2},"Inv",M4,"R"), N3$\xrightarrow{R}$N1 , N4$\xrightarrow{O+R}$N3, N5$\xrightarrow{R}$N3 , N4$\xrightarrow{O}$N5})
p_3= (L={N1(X_{Cont},"Cont",M1) , N2("Temp")} ; K={N1(X_{Cont},"Cont",M1) , N2("Temp")} ; R={N3(X_{Coord},"Coord","φ_1",M2,"R"), N3$\xrightarrow{R}$N2}), N4(X_I,"Inv",M3,"O+R"), N4$\xrightarrow{O+R}$N3})
p_4= (L={N1(X_{Coord},"Coord","φ_2",M1,"R"), N2(X_{I1},"Inv_α",M2,"O+R"),N2$\xrightarrow{O+R}$N1 , N3("Temp")} ; K={N1(X_{Coord},"Coord","φ_2",M1,"R"), N2(X_{I1},"Inv_α",M2,"O+R"),N2$\xrightarrow{O+R}$N1 , N3("Temp")} ; R={N4(X_{I2},"Inv",M3,"R") , N2$\xrightarrow{O}$N4, N4$\xrightarrow{R}$N1} ;

p_5= (L={N1(X_{Coord},"Coord","φ_1",M1,"R") , N2("Temp")} ; K={N1(X_{Coord},"Coord","φ_1",M1,"R") , N2("Temp")} ; R={N3(X_I,"Inv",M2,"O+R"),N3$\xrightarrow{O+R}$N2})
p_6= (L={N("Temp")};K={ };R={ })

FIG. 3.5 – Les productions de grammaire pour la description de l'architecture au niveau application et en phase d'action

La production p_1 génère, à partir de l'axiome, le participant qui joue le rôle de contrôleur (nœud N1), un coordinateur dans la phase d'action φ_2 (le nœud N2), l'investigateur (nœud N3) qui correspond à celui qui a découvert la situation critique, et le premier investigateur β (nœud N4). Ces composants sont reliés par des liens exprimant le type de données échangées (i.e. de type rapport entre le coordinateur et le contrôleur, de type observation et rapport entre l'investigateur α et son coordinateur, de type rapport entre l'investigateur β et le coordinateur, et de type observation entre l'investigateur α et l'investigateur β). Cette production génère le seul non terminal *Temp* qui doit être consommé à la fin du processus de génération.

Les productions p_2 et p_3 permettent de prendre en compte des coordinateurs supplémentaires. p_2 génère un coordinateur (nœud N3) en phase d'action ($\varphi 2$) et ses investigateurs α (nœud N4) et β (nœud N5) tandis que p_3 permet de générer, à chaque application, un coordinateur en phase d'exploration ($\varphi 1$) ainsi que son premier investigateur.

p_4 permet de générer, pour un coordinateur en phase d'action (nœud N1), un investigateur β supplémentaire. Cet investigateur, selon la spécification donnée précédemment, est connecté au coordinateur par un lien de type R, et à l'investigateur α par un lien de type O.

p_5 génère, à chaque application, un investigateur supplémentaire pour un coordinateur en phase d'exploration. Cette investigateur est connecté au coordinateur par un lien supportant les données de type O et R.

La production p_6 élimine le non terminal *Temp* et termine le processus de génération en obtenant une instance qui contient exclusivement des nœuds terminaux.

L'instance de l'architecture introduite par la figure peut être produite par la grammaire $GG_{A,a}$ selon le chemin de génération : 1) p_1 pour générer les participants Co, $C1$ et $I1$; 2) p_3 pour générer le coordinateur $C2$ et l'investigateur $I5$; 3) p_4 pour générer $I2$ l'investigateur β ; 4) p_4 pour générer $I3$ l'autre investigateur β ; 5) p_5 pour générer l'investigateur $I4$ et finalement ; 6) la production p_6 pour terminer le processus de génération.

Propriétés architecturales

Les propriétés architecturales de notre système au niveau application et en phase d'exploration sont présentées dans la figure 3.6.

$pr_{A,6'}$= (L={N1(X_{Coord},"Coord",φ2,M1,"R")} ; N={N2(X_I,"Inv",M2,"O+R"), N2$\xrightarrow{O+R}$N1}), Neg
$pr_{A,7'}$= (L={N1(X_{Coord},"Coord",φ2,M1,"R")} ; N={N2(X_I,"Inv",M2,"R"), N2$\xrightarrow{R}$N1}), Neg
$pr_{A,8'}$= (L={N1(X_{Coord},"Coord",φ2,M1,"R"), N2(X_I,"Inv",M2,"O+R"), N2$\xrightarrow{O+R}$N1, N3(X_I,"Inv",M3,"O+R"), N3$\xrightarrow{O+R}$N1} ; N={ }), Neg

FIG. 3.6 – Les propriétés architecturales au niveau application et en phase d'action.

Les propriétés $pr_{A,1}, \ldots, pr_{A,5}$ définies dans la section relative aux propriétés de l'application en phase d'exploration restent valables même en phase d'action. À ces propriétés viennent se rajouter : 1) la propriété négative $pr_{A,6'}$ qui stipule qu'un coordinateur $\varphi 2$ coordonne au moins un investigateur α, 2) la

propriété négative $pr_{A,7'}$ impliquant qu'il coordonne aussi au moins un investigateur β, et finalement 3) la propriété négative $pr_{A,8'}$ dont la non applicabilité assure l'existence d'un investigateur α unique pour chaque section de l'application qui se trouve en phase d'action.

3.3 Description de l'architecture au niveau middleware

Le niveau middleware concerne les communications de composant à composant qui raffinent les liens de niveau application. A ce niveau, trois rôles sont distingués : producteur d'événements (P), consommateur d'événements (C) et gestionnaire de canal (G). Plusieurs producteurs et consommateurs peuvent être reliés par un même canal.

Les actions d'adaptation consistent à instancier et déployer correctement les entités gestionnaires de canaux (G) entre les différents participants de l'application. Clairement, la localisation d'un gestionnaire sur une machine consommera de l'espace mémoire et engendrera une utilisation supplémentaire de CPU et donc d'énergie. Le choix de déploiement (ou de redéploiement pendant l'exécution) doit donc être guidé par la disponibilité de ces ressources sur les machines qui hébergent les différents participants.

3.3.1 La phase d'exploration

Durant l'étape d'exploration, chaque investigateur est constitué de deux composants producteurs de données et correspondant respectivement aux données de type observation (O) et rapport (R). Le coordinateur est constitué de deux consommateurs de données permettant de traiter les données d'observation et de rapport de ses investigateurs et d'un producteur de données permettant de générer les rapports envoyés vers le contrôleur. Le contrôleur est constitué d'un consommateur pour les données envoyées par ses différents coordinateurs.

Nous supposons qu'un investigateur est toujours autorisé à héberger un gestionnaire mais étant donné ses ressources limitées, il ne peut héberger les deux gestionnaires de sa section en même temps. Le coordinateur, par contre, peut hébérger les deux gestionnaires étant donné qu'il s'agit d'une machine fixe bénéficiant d'une source d'énergie permanente. Le gestionnaire du canal traitant les messages entre le contrôleur et les coordinateurs est toujours déployé sur la machine du contrôleur du fait que cette machine est celle qui dispose des ressources les plus larges. L'autre argument concerne la sensibilité de l'application à ce gestionnaire puisque sa perte serait catastrophique et impliquerait l'arrêt de l'application en coupant toutes les communications vers contrôleur.

Le style architectural au niveau middleware pendant la phase d'exploration est donné par la grammaire de graphe $GG_{M,e}$. Cette grammaire considère les nœuds et les labels suivants :

- Le nœud *N("C","C_{cont}","M","R")* décrit le consommateur d'événements du contrôleur où C donne l'identifiant du composant, C_{cont} correspond au type indiquant qu'il s'agit d'un consommateur d'événements rattaché au contrôleur, M correspond à la machine du contrôleur et R indique que ce consommateur reçoit des événements de type R.
- Les composants correspondant aux producteurs d'événements envoyés du coordinateur vers le contrôleur sont décrits par un nœud *N("P1","P_{coord}","M","R")* où *P1* est l'identifiant du composant, P_{coord} exprime le fait qu'il correspond à un producteur d'événements rattaché à un coordinateur, M correspond à la machine sur laquelle le composant est déployé et R indique que ce composant produit des événements de type R.
- Les composants correspondant aux consommateurs d'événements rattachés au coordinateur sont décrits par un nœud *N("C1","C_{coord}","M_C",X)* où *C1* est l'identifiant du composant, C_{coord} correspond au type décrivant les consommateurs d'événements du coordinateur, M_C indique la machine de déploiement. Selon que X corresponde à la valeur R ou O, le nœud correspondra à un composant consommateur d'événements de type R ou O.
- De manière analogue, les nœuds *N1("P","P_{Inv}","M","R")* et *N2("P","P_{Inv}","M","O")* décrivent des producteurs d'événements dont l'identifiant est P, le type P_{Inv} indique qu'ils constituent des producteurs rattachés à des investigateurs, M correspond à la machine de déploiement tandis que O et R indiquent le type d'événements qu'ils produisent.

– Les nœuds *N1("G1","G","M","R")* et *N2("G1","G","M","O")* correspondent à des gestionnaires de canaux de communication. Le label *G1* donne l'identifiant du composant, *G* indique qu'il s'agit d'un composant de type gestionnaire de canal, *M* correspond à la machine de déploiement alors que *R* et *O* donnent le type d'événements qui transitent par le canal de communication concerné.

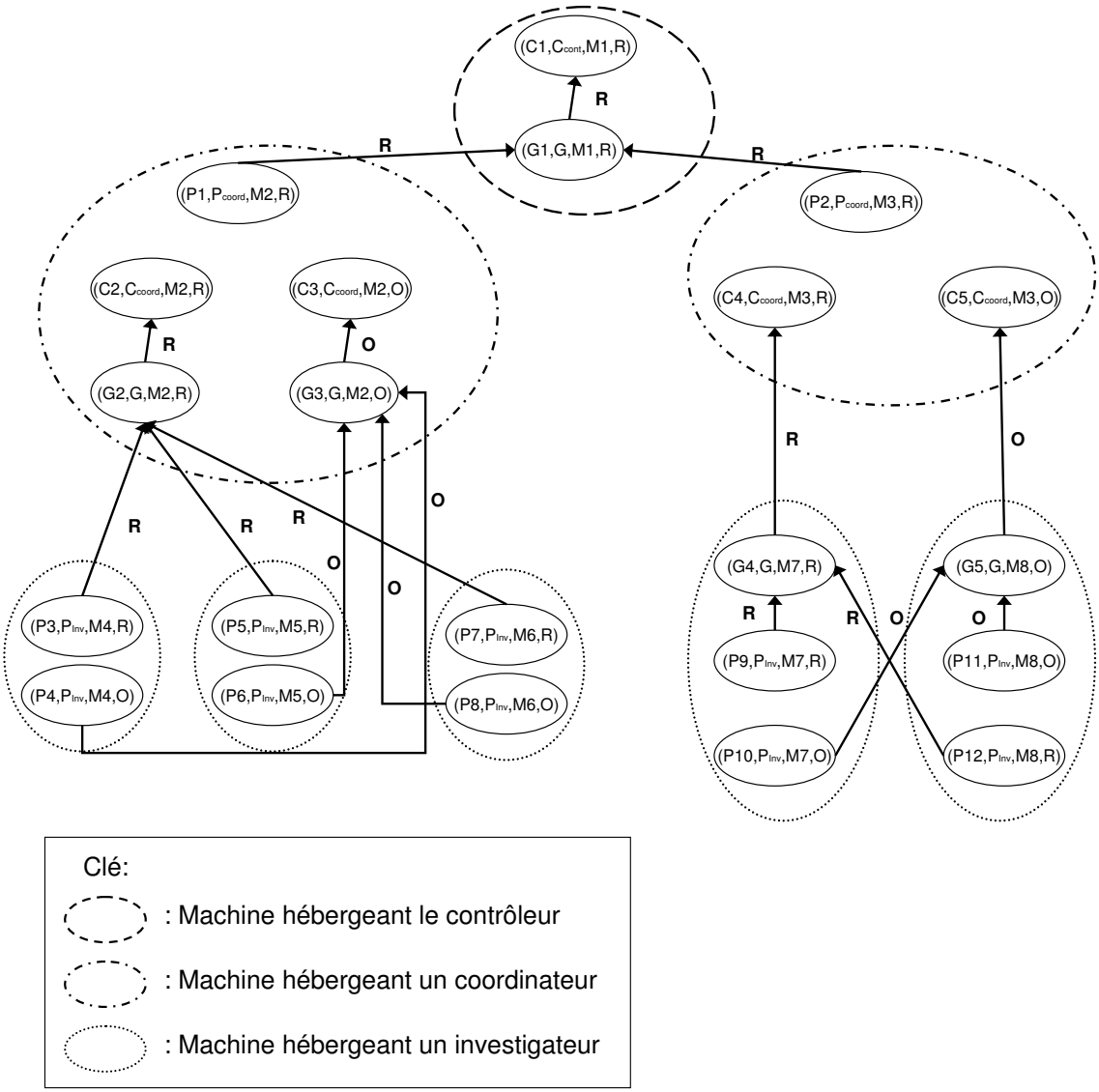


FIG. 3.7 – Exemple de l'architecture dans la phase d'exploration.

La figure 3.7 présente une configuration de l'architecture au niveau middleware et en phase d'exploration. Cette configuration inclut, pour le contrôleur, le composant *C1* consommateur des rapports provenant des coordinateurs et le composant correspondant au gestionnaire *G1* dédié à ce type de données. *G1* est déployé sur la machine du contrôleur *M1*. L'application contient deux coordinateurs correspondant aux machines *M2* et *M3*.

La machine *M2* héberge *P1* le producteur des rapports vers le contrôleur, et *C2* et *C3* les consommateurs des données de type *O* et *R* provenant des investigateurs. Pour le coordinateur de la machine *M2*, le choix est de déployer les deux gestionnaires *G2* et *G3* sur cette même machine *M2*. Ce coordinateur gère trois investigateurs sur les machines *M4*, *M5*, *M6*. Chaque machine héberge un producteur de données de

type R relié au gestionnaire *G2* et un producteur de données de type O relié au gestionnaire *G3*.

La machine *M3* héberge le deuxième coordinateur et contient, par conséquent, le producteur de données de type R relié au gestionnaire *G1*, et *C4* et *C5* les deux consommateurs de données de types R et O. Les gestionnaires de canaux pour les données entre les investigateurs et ce coordinateur sont déployés sur les machines d'investigateur M7 et M8. M7 contient, en plus du gestionnaire de type O, les producteurs de données de types O et R (i.e. respectivement *P9* et *P10*) alors que *M8* héberge le gestionnaire de type R et les producteurs de données *P11* et *P12*. Les liens de communication sont établis pour permettre d'acheminer correctement les données.

Nous obtenons, donc, le style architectural suivant pour les architectures niveau middleware en phase d'exploration.

$GG_{M,e}$=(AX,NT,T,P) avec :
T={N(X_P,"P_{coord}",M_P,"R"), N(X_C,"C_{coord}",M_C,"R"), N(X_C,"C_{coord}",M_C,"O"),
N(X_P,"P_{Inv}",M_P,"R"), N(X_P,"P_{Inv}",M_P,"O"), N(X_C,"C_{cont}",M_C,"R"),
N(X_G,"G",M_G,"R"), N(X_G,"G",M_G,"O")}, et
NT={N("Temp")}, et
P={p_1,...,p_{10}}, et

p_1= (L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M2,"R"), N7(X_{G3},"G",M2,"O"), N6$\xrightarrow{R}$N4, N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N7, N10("Temp")})
p_2=(L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M2,"R"), N7(X_{G3},"G",M3,"O"), N6$\xrightarrow{R}$N4, N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N7, N("Temp")})
p_3=(L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M3,"R"), N7(X_{G3},"G",M2,"O"), N6$\xrightarrow{R}$N4, N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N("Temp")})
p_4=(L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O") , N6(X_{G2},"G",M3,"R"), N7(X_{G3},"G",M4,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N10(X_{P4},"P_{Inv}",M4,"R") N11(X_{P5},"P_{Inv}",M4,"O"), N8$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N7, N10$\xrightarrow{R}$N6, N11$\xrightarrow{O}$N7, N("Temp")})

p_5= (L={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R") , N3$\xrightarrow{R}$ $N2$, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M2,"R"), N7(X_{G3},"G",M2,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N7})
p_6= (L={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M2,"R"), N7(X_{G3},"G",M3,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7})
p_7= (L={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M3,"R"), N7(X_{G3},"G",M2,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$ $N7$})
p_8= (L={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M3,"R"), N7(X_{G3},"G",M4,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N10(X_{P4},"P_{Inv}",M4,"R"), N11(X_{P5},"P_{Inv}",M4,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10$\xrightarrow{R}$N6 , N11$\xrightarrow{O}$N7})
p_9= (L={N1(X_{C1},"C_{coord}",M1,"R"), N2(X_{C2},"C_{coord}",M1,"O"), N("Temp"), N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M3,"O"), N3$\xrightarrow{R}$N1 , N2$\xrightarrow{O}$N4} K={N1(X_{C1},"C_{coord}",M1,"R") , N2(X_{C2},"C_{coord}",M1,"O") , N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M3,"O"), N3$\xrightarrow{R}$N1 , N4$\xrightarrow{O}$N2 , N("Temp")}; R={N5(X_{P1},"P_{Inv}",M4,"R"), N6(X_{P2},"P_{Inv}",M4,"O"), N5$\xrightarrow{R}$N3 , N6$\xrightarrow{O}$N4})
p_{10}= (L={N("Temp")};K={ };R={ })

FIG. 3.8 – Les productions de la grammaire pour le style architectural niveau middleware en phase d'exploration

La production p_1 génère les différents composants constituant le contrôleur de l'application : i.e. le consommateur des rapports envoyés par les coordinateurs (le nœud N1) et le gestionnaire du canal de communication qui gère l'envoie de ces données (nœud N2). Ces deux composants sont hébergés sur la machine du contrôleur (la machine M1). p_1 génère aussi : les composants d'un coordinateur comprenant le producteur d'événements de type R pour le contrôleur (nœud N4), les consommateurs d'événements O et R (respectivement les nœuds N4 et N5) en provenance des investigateurs, et les composants d'un premier investigateur incluant les producteurs de données O et R (respectivement les nœuds N8 et N9) destinées au coordinateur. Cette production exprime un choix de déployer les deux gestionnaires de canaux O et R sur la machine du coordinateur (i.e. machine M2).

Les productions p_2, p_3 et p_4 génèrent exactement les mêmes composants que p_1 mais impliquent des

choix différents au niveau du déploiement des gestionnaires de canaux. Elles correspondent respectivement aux alternatives : 1) déployer le gestionnaire R sur le coordinateur et le gestionnaire O sur un investigateur, 2) déployer le gestionnaire O sur le coordinateur et le gestionnaire R sur un investigateur, et 3) déployer les deux gestionnaires sur deux investigateurs différents.

La production p_5 permet de générer les composants des coordinateurs additionnels. Elle génère le composant producteur de rapports pour le contrôleur (nœud N3) et les consommateurs de données de type O et R envoyées par les investigateurs. Cette production génère aussi les producteurs de données O et R correspondant à un premier investigateur[27].

p_5 correspond au choix de déployer les deux gestionnaires de données O et R sur la machine du coordinateur (i.e. M2). Les productions p_6, p_7 et p_8 correspondent aussi à l'introduction de coordinateurs supplémentaires mais impliquent des choix différents de déploiement des gestionnaires. p_6 correspond au déploiement du gestionnaire R sur la machine du coordinateur (i.e. machine M2) et du gestionnaire O sur la machine d'un investigateur (M3). p_7 correspond au choix de déploiement inverse alors que p_8 correspond au déploiement des gestionnaires sur les machines de deux investigateurs différents (machines M3 et M4).

La production p_9 permet de générer, pour un coordinateur donné, les composants appartenant à un investigateur additionnel. Ceci implique l'introduction des producteurs de données R (nœud N5) et de données O (nœud N6). Les deux producteurs sont connectés aux gestionnaires correspondants (respectivement nœuds N3 et N4). Cette production prend en compte tous les cas de figure concernant le déploiement des gestionnaires puisque les variables M2 et M3 peuvent correspondre à la même valeur dans le graphe terminal produit.

La production p_{10} termine le processus de génération.

L'instance de l'architecture présentée dans la figure 3.7 peut être obtenue par application de la grammaire $GG_{M,e}$ selon le chemin de génération suivant : 1) p_1 pour obtenir le consommateur *C1*, le gestionnaire *G1*, le producteur *P1*, les consommateurs *C2* et *C3*, les gestionnaires *G2* et *G3* déployés sur la machine *M2*, et les producteurs *P3* et *P4* ; 2) la production p_8 pour générer le producteur *P2*, les consommateurs *C4* et *C5*, les gestionnaires *G4* et *G5* déployés sur les deux machines *M7* et *M8*, les producteurs *P9* et *P10*, et les producteurs *P11* et *P12* ; 3) p_9 qui génère les producteurs *P5* et *P6* ; 4) p_9 qui génère *P7* et *P8* ; et finalement 5) p_{10} pour terminer.

Propriétés architecturales

La figure 3.9 présente les propriétés considérées pour le niveau middleware en phase d'exploration.

$pr_{M,1}$= (L={N1(X_P,"P_{coord}",M1,"R"), N2(X_G,"G",M1,"R"), N1$\xrightarrow{R}$N2} ; N={ }), Neg
$pr_{M,2}$= (L={N1(X_P,"P_{Inv}",M1,"R"),N2(X_P,"P_{coord}",M2,"R"), N3(X_G,"G",M1,"R"), N2$\xrightarrow{R}$N3} ; N={ }), Neg
$pr_{M,3}$= (L={N1(X_P,"P_{Inv}",M1,"R"), N2(X_{G1},"G",M1,T), N3(X_{G2},"G",M1,T')} ; N={ }), Neg
$pr_{M,4}$= (L={N1(X_P,"P_{Inv}",M1,"R"), N2(X_{G1},"G",M1,"R")} ; N={N1$\xrightarrow{R}$N2}), Neg
$pr_{M,5}$= (L={N1(X_P,"P_{Inv}",M1,"O"), N2(X_{G1},"G",M1,"O")} ; N={N1$\xrightarrow{O}$N2}), Neg

FIG. 3.9 – Les propriétés architecturales au niveau middleware et en phase d'exploration.

La propriété négative $pr_{M,1}$ caractérise le fait que le gestionnaire en charge de l'acheminement des événements vers le contrôleur ne peut être hébergé par une machine d'un coordinateur. La propriété négative $pr_{M,2}$ spécifie que ce gestionnaire ne peut pas non plus être hébergé par la machine d'un investigateur.

[27] Un coordinateur possède au moins un investigateur.

l'applicabilité de la règle $pr_{M,3}$ implique que la propriété instaurant qu'un investigateur ne peut héberger deux gestionnaires est violée. Les propriétés négatives $pr_{M,4}$ et $pr_{M,5}$ correspondent (respectivement pour les données de type "R" et "O") à l'exigence qu'un investigateur ne puisse héberger des gestionnaires qui ne font pas partie de sa propre section.

3.3.2 La phase d'action

En ce qui concerne la partie de l'architecture relevant des coordinateurs qui sont en phase d'exploration, la composition des coordinateurs et des investigateurs qui leurs sont rattachés ainsi que leurs interconnexions restent les mêmes que dans la phase d'exploration. Les coordinateurs qui sont passés en phase d'action, les structures internes de la section ainsi que les interactions sont réorganisées pour s'adapter au nouveau contexte d'exécution selon les contraintes introduites au niveau application.

Dans ce contexte, le coordinateur déploie, en plus du producteur d'événements de type R connecté au gestionnaire du contrôleur, trois consommateurs d'événements. Les deux premiers concernent les événements de type R et O provenant de l'investigateur α. Le troisième consommateur est dédié aux événements de type R envoyés par les investigateurs β. Les deux gestionnaires (un pour les événements R et l'autre pour les événements O) en charge des communications entre l'investigateur α et le coordinateur sont extrêmement critiques. Pour des raisons de sécurité et de disponibilité de ressources, ils sont forcément déployés sur la machine du coordinateur.

L'investigateur α correspond à deux composants producteurs d'événements (de types R et O) pour le coordinateur et à un producteur d'événements de type O pour les investigateurs β. Les investigateurs β possèdent un consommateur d'événements de type O pour les événements envoyés par l'investigateur α, et un producteur d'événements de type R produisant les rapports pour le coordinateur.

Le gestionnaire du canal correspondant aux événements envoyés par l'investigateur α vers les autres investigateurs (de type O) et le gestionnaire correspondant aux événements envoyés par les investigateurs β vers le coordinateur sont moins critiques que les autres gestionnaires et peuvent être déployés sur des investigateurs β [28]. Par contre, Ils ne peuvent être déployés sur le même investigateur ni sur l'investigateur α dont les ressources doivent être préservées.

Nous considérons les nœuds suivants pour la description des différents composants au niveau middleware et en phase d'action :

- Pour les composants relevant de la partie de l'architecture qui implique des coordinateurs et leurs investigateurs en phase d'exploration, nous retrouvons naturellement les mêmes nœuds utilisés pour la description de l'architecture dans cette phase ; i.e. les nœuds $N("C","C_{cont}","M","R")$, $N("P","P_{coord}","M_P","R")$, $N("C","C_{coord}","M_C","R")$, $N("C","C_{coord}","M_C","O")$, $N("P","P_{Inv}","M","R")$, $N("P","P_{Inv}","M","O")$, $N("G1","G","M","R")$, et $N("G1","G","M","O")$ pour respectivement le consommateur R du contrôleur, le producteur R du coordinateur, les consommateurs R et O du coordinateur, les producteurs R et O des investigateurs, et les gestionnaires O et R des canaux de communication.
- Pour les composants relevant de coordinateurs en phase d'action et de leurs investigateurs, nous introduisons les nœuds suivants :
  - Les nœuds $N("G","G_\alpha","M","R")$ et $N("G","G_\alpha","M","O")$ correspondent aux gestionnaires de canaux responsables des communications entre les producteurs d'événements R et O de l'investigateur α et les consommateurs d'événements R et O du coordinateur.
  - Le nœud $N("G","G_\beta","M","O")$ correspond aux gestionnaires en charge des communications entre un producteur d'événements O de l'investigateur α et les consommateurs d'événements O appartenant aux investigateurs β.
  - Le nœud $N("G","G_\beta","M","R")$ correspond aux gestionnaires en charge des communications entre les producteurs d'événements R des investigateurs β et le consommateur d'événements R appartenant au coordinateur.
  - Les nœuds $N("P","P_{Inv,\alpha}","M","R")$ et $N("P","P_{Inv,\alpha}","M","O")$ correspondent respectivement aux producteurs de données R et O d'un investigateur α.

[28] Ceci est justifié par le fait que la section peut être très éloignée du coordinateur et que les communications entre investigateurs peuvent être plus performantes si le gestionnaire est hébergé par un investigateur au lieu d'être déployé sur la machine du coordinateur.

– Pour un investigateur β, les nœuds $N("P","P_{Inv}","M","R")$ et $N("C","C_{Inv}","M","O")$ correspondent respectivement au producteur des données R pour le coordinateur et du consommateur de données envoyées par l'investigateur α.

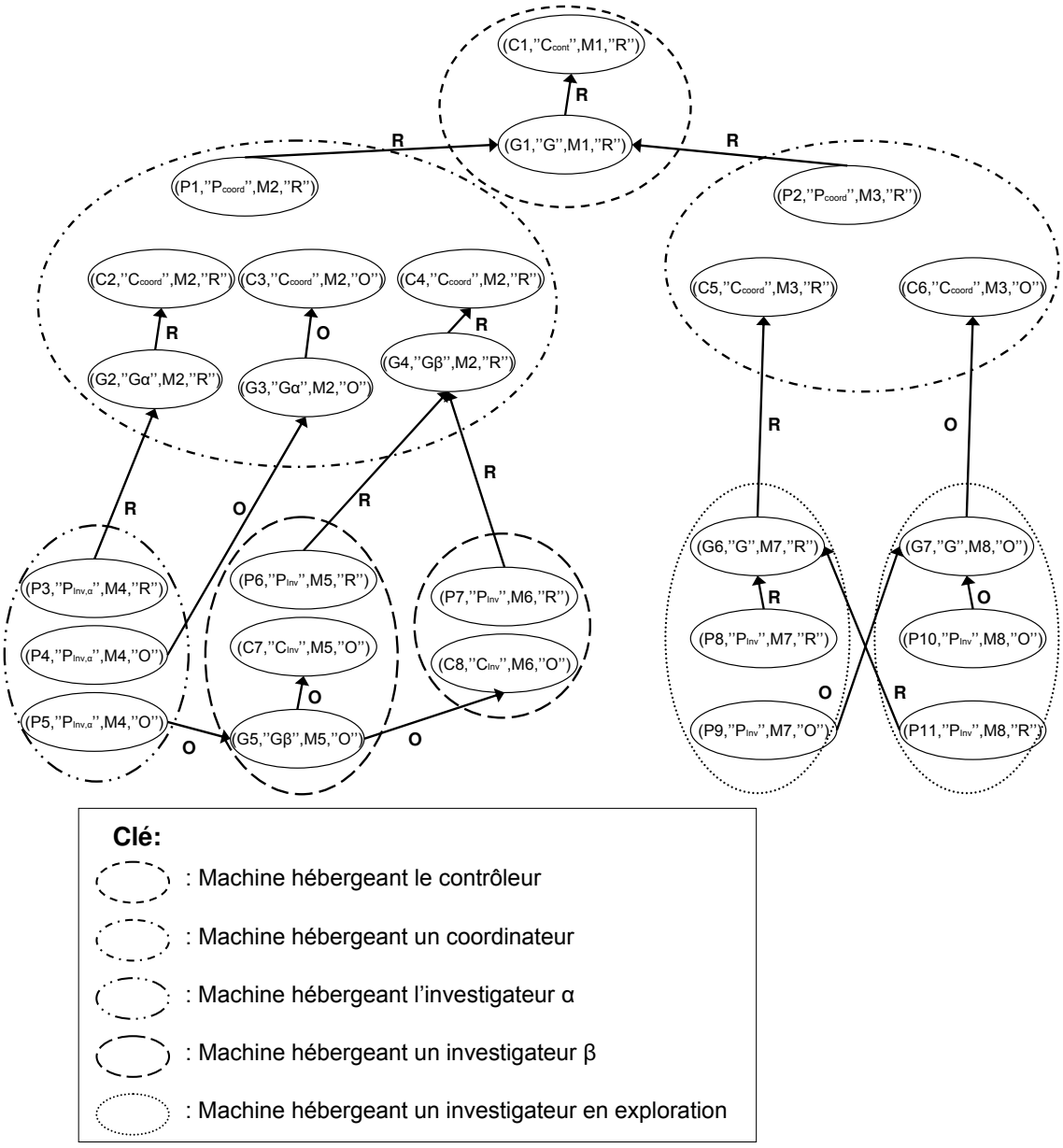


FIG. 3.10 – Exemple de l'architecture dans la phase d'action.

La figure 3.10 présente un exemple de description pour le niveau middleware en considérant l'application dans une phase d'action. L'application décrite contient, en plus du contrôleur, deux coordinateurs.

Le premier coordinateur, situé sur la machine M2, est en phase d'action. Ce coordinateur est en charge de coordonner trois investigateurs : l'investigateur situé sur la machine M4 qui joue le rôle d'investigateur α alors que les deux autres investigateurs (situés sur les machines M5 et M6) sont des investigateurs β. Les gestionnaires α de ce coordinateur sont bien évidemment déployés sur la machine M2. Le gestionnaire

β relatif aux données envoyées par les investigateurs β vers le coordinateur est déployé sur la machine du coordinateur alors que l'autre gestionnaire β correspondant aux données envoyées par l'investigateur α vers les autres investigateurs est déployé sur la machine M5.

Le deuxième coordinateur, situé sur la machine M3, est en phase d'exploration. Il coordonne les deux investigateurs hébergés sur les machines M7 et M8. Les gestionnaires de canaux permettant la communication entre ce coordinateur et les investigateurs de sa section sont déployés chacun sur une machine d'investigateur.

Le style d'architecture de l'application au niveau middleware et en phase d'action est généré par la grammaire $GG_{M,a}$= (AX,NT,T,P) avec :

T={N(X_C,"C_{cont}",M_C,"R"), N(X_P,"P_{coord}",M_P,"R"), N(X_C,"C_{coord}",M_C,"R"),
N(X_C,"C_{coord}",M_C,"O"), N(X_P,"P_{Inv}",M_P,"R"), N(X_P,"P_{Inv}",M_P,"O"),
N(X_P,"$P_{Inv,\alpha}$",M_P,"R"), N(X_P,"$P_{Inv,\alpha}$",M_P,"O"), N(X_C,"C_{Inv}",M_C,"O"),
N(X_G,"G",M_G,"R"), N(X_G,"G",M_G,"O"), N(X_G,"G_α",M_G,"R"),
N(X_G,"G_α",M_G,"O"), N(X_G,"G_β",M_G,"R"), N(X_G,"G_β",M_G,"O")}, et
NT={N("Temp")}, et
P={p_1,...,p_{15}}, et

p_1= (L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_α",M2,"R"), N8(X_{G3},"G_α",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_β",M2,"O"), N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7, N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9, N13(X_{G5},"G_β",M2,"R"), N14(X_{P5},"P_{Inv}",M4,"R"), N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13 , N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N("Temp")})
p_2= (L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{Coord}",M2,"R"), N7(X_{G2},"G_α",M2,"R"), N8(X_{G3},"G_α",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_β",M4,"O"), N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7, N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9, N13(X_{G5},"G_β",M2,"R"), N14(X_{P5},"P_{Inv}",M4,"R"), N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13, N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N("Temp")})
p_3= (L={AX};K={ }; R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_α",M2,"R"), N8(X_{G3},"G_α",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_β",M2,"O"), N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7, N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9, N13(X_{G5},"G_β",M4,"R"), N14(X_{P5},"P_{Inv}",M4,"R"), N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13, N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N("Temp")})

p_4= (L={AX};K={ };
R={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P1},"P_{coord}",M2,"R"),
N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R") , N5(X_{C3},"C_{coord}",M2,"O"),
N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_{α}",M2,"R"), N8(X_{G3},"G_{α}",M2,"O"),
N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_{β}",M4,"O"), N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"),
N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7,
N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9, N13(X_{G5},"G_{β}",M5,"R"), N14(X_{P5},"P_{Inv}",M4,"R"),
N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13 , N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15,
N16(X_{P6},"P_{Inv}",M5,"R"), N17(X_{C6},"C_{Inv}",M5,"O"), N16$\xrightarrow{R}$N17,
N9$\xrightarrow{O}$N16, N("Temp")})

p_5= (L={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp"), N2$\xrightarrow{R}$ $N1$};
K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp"), N2$\xrightarrow{R}$ $N1$};
R={N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$ $N2$, N4(X_{C2},"C_{coord}",M2,"R"),
N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_{α}",M2,"R"),
N8(X_{G3},"G_{α}",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_{β}",M2,"O"),
N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"),
N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7 , N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9,
N13(X_{G5},"G_{β}",M2,"R"), N14(X_{P5},"P_{Inv}",M4,"R"),
N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13 , N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N("Temp")})

p_6= (L={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp") , N2$\xrightarrow{R}$N1};
K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp") , N2$\xrightarrow{R}$N1};
R={ N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"),
N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_{α}",M2,"R"),
N8(X_{G3},"G_{α}",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_{β}",M4,"O"),
N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"),
N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7 , N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9,
N13(X_{G5},"G_{β}",M2,"R"), N14(X_{P5},"P_{Inv}",M4,"R"),
N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13 , N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N("Temp")})

p_7= (L={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp") , N2$\xrightarrow{R}$N1};
K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp") , N2$\xrightarrow{R}$N1};
R={ N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"),
N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_{α}",M2,"R"),
N8(X_{G3},"$G\alpha$",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_{β}",M2,"O"),
N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"),
N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7 , N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9,
N13(X_{G5},"G_{β}",M4,"R"), N14(X_{P5},"P_{Inv}",M4,"R"),
N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13 , N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N("Temp")})

p_8= (L={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp") , N2$\xrightarrow{R}$N1}; K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N("Temp") , N2$\xrightarrow{R}$N1}; R={ N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{C4},"C_{coord}",M2,"R"), N7(X_{G2},"G_{α}",M2,"R"), N8(X_{G3},"G_{α}",M2,"O"), N7$\xrightarrow{R}$N4 , N8$\xrightarrow{O}$N5, N9(X_{G4},"G_{β}",M4,"O"), N10(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N11(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N12(X_{P4},"$P_{Inv,\alpha}$",M3,"O"), N10$\xrightarrow{R}$N7 , N11$\xrightarrow{O}$N8, N12$\xrightarrow{O}$N9, N13(X_{G5},"G_{β}",M5,"R"), N14(X_{P5},"P_{Inv}",M4,"R"), N15(X_{C5},"C_{Inv}",M4,"O"), N14$\xrightarrow{R}$N13, N13$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N15, N16(X_{P6},"P_{Inv}",M5,"R"), N17(X_{C6},"C_{Inv}",M5,"O"), N16$\xrightarrow{R}$N13, N9$\xrightarrow{O}$N17, N("Temp")})
p_9= (L={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1 ,N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R") , N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M2,"R"), N7(X_{G3},"G",M2,"O") , N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7})
p_{10}= (L={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1 ,N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R") , N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1 ,N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M2,"R"), N7(X_{G3},"G",M3,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7})
p_{11}= (L={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1 ,N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1 , N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M3,"R"), N7(X_{G3},"G",M2,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7})
p_{12}= (L={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; K={N1(X_{C1},"C_{cont}",M1,"R"), N2(X_{G1},"G",M1,"R"), N2$\xrightarrow{R}$N1, N("Temp")}; R={N3(X_{P1},"P_{coord}",M2,"R"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{R}$N2, N4(X_{C2},"C_{coord}",M2,"R"), N5(X_{C3},"C_{coord}",M2,"O"), N6(X_{G2},"G",M3,"R"), N7(X_{G3},"G",M4,"O"), N6$\xrightarrow{R}$N4 , N7$\xrightarrow{O}$N5, N8(X_{P2},"P_{Inv}",M3,"R"), N9(X_{P3},"P_{Inv}",M3,"O"), N10(X_{P4},"P_{Inv}",M4,"R"), N11(X_{P5},"P_{Inv}",M4,"O"), N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10$\xrightarrow{R}$N6 , N11$\xrightarrow{O}$N7})
p_{13}= (L={N1(X_{C1},"C_{coord}",M1,"R") , N2(X_{C2},"C_{coord}",M1,"O"), N("Temp"), N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M3,"O"), N3$\xrightarrow{R}$N1, N2$\xrightarrow{O}$N4}; K={N1(X_{C1},"C_{coord}",M1,"R") , N2(X_{C2},"C_{coord}",M1,"O"), N("Temp"), N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M3,"O"), N3$\xrightarrow{R}$N1 , N2$\xrightarrow{O}$N4 }; R={N5(X_{P1},"P_{Inv}",M4,"R"), N6(X_{P2},"P_{Inv}",M4,"O"), N5$\xrightarrow{R}$N3, N6$\xrightarrow{O}$N4})

p_{14}= (L={N1(X_{C1},"C_{coord}",M1,"R"), N2(X_{G1},"G_β",M2,"R"), N("Temp"),
N3(X_{G2},"G_β",M3,"O"), N2$\xrightarrow{R}$N1, N4(X_{P1},"$P_{inv,\alpha}$",M4,"O"),
N5(X_{G3},"G_α",M1,"O"), N4$\xrightarrow{O}$N5, N6(X_{P2},"$P_{inv,\alpha}$",M4,"O"),N6$\xrightarrow{O}$N3};
K={N1(X_{C1},"C_{coord}",M1,"R"), N2(X_{G1},"G_β",M2,"R"), N("Temp"),
N3(X_{G2},"G_β",M3,"O"), N2$\xrightarrow{R}$N1, N4(X_{P1},"$P_{inv,\alpha}$",M4,"O"),
N5(X_{G3},"G_α",M1,"O"),N4$\xrightarrow{O}$N5, N6(X_{P2},"$P_{inv,\alpha}$",M4,"O"),N6$\xrightarrow{O}$ $N3$};
R={N7(X_{P3},"P_{Inv}",M5,"R"), N8(X_{C2},"C_{Inv}",M5,"O"), N7$\xrightarrow{R}$N2, N3$\xrightarrow{O}$N8 })

p_{15}= (L={N("Temp")};K={ };R={ })

FIG. 3.11 – Les productions de la grammaire pour la description de l'architecture dynamique niveau middleware en phase d'action

La production p_1 permet de générer les composants du contrôleur constitué d'un consommateur d'événements envoyés par les coordinateurs et du gestionnaire de canal dédié à ces événements. Elle génère aussi les composants du coordinateur en phase d'action contenant le producteur d'événements de type R à destination du contrôleur, les deux consommateurs des événements R et O envoyés par l'investigateur α et les gestionnaires de canaux dédiés à ces communications. p_1 produit aussi les composants de l'investigateur α constitué des producteurs d'événements à destination du coordinateur, et du producteur d'événements O pour les autres investigateurs. Nous générons aussi les composants d'un investigateur β constitué d'un consommateur pour les événements O produits par l'investigateur α et le producteur d'événements R envoyés au coordinateur.

La production p_1 exprime le choix de déployer les gestionnaires de canaux, entre l'investigateur α et les autres investigateurs et entre les investigateurs β et le coordinateur, sur la machine du coordinateur. Les productions p_2, p_3 et p_4 permettent de générer exactement les mêmes composants que la production p_1 mais correspondent aux trois autres choix de déploiement pour les gestionnaires G_β.

Les productions p_5, p_6, p_7 et p_8 permettent, à chaque application, d'introduire les composants correspondant à un coordinateur $\varphi 2$, à son investigateur α et à un investigateur β. Chacune de ces productions correspond à un des quatre choix de déploiement possibles pour les gestionnaires G_β.

Les productions p_9, p_{10}, p_{11} et p_{12} correspondent à l'introduction de coordinateurs additionnels en phase d'exploration en prenant en compte les différentes options consistantes pour le déploiement des gestionnaires de canaux.

La production p_{13} permet de générer les composants d'un investigateur supplémentaire pour un coordinateur en phase d'exploration. p_{14} génère un investigateur β supplémentaire pour un coordinateur en phase d'action. La production p_{15} termine la génération de la description du style architectural.

L'instance de l'architecture présentée dans la figure 3.10 peut être obtenue par application de la grammaire $GG_{M,a}$ selon le chemin de génération suivant : 1) p_2 pour obtenir le consommateur *C1*, le gestionnaire *G1*, le producteur *P1*, les consommateurs *C2*, *C3* et *C4* , les gestionnaires α *G2* et *G3*, le gestionnaire *G4*, les producteurs de l'investigateur α *P3*, *P4* et *P5*, le producteur *P6* et le consommateur *C7* ainsi que le gestionnaire *G5* qui sont hébergés par un investigateur β ; 2) p_{12} génère les composants correspondant au coordinateur (en phase d'exploration) de la machine *M3* comprenant le producteur *P2* et les consommateurs *C5* et *C2*. Cette production correspond au déploiement des gestionnaires sur les machines des investigateurs. Elle produit, donc, les producteurs *P8*, *P9*, *P10* et *P11* ainsi que les gestionnaires de canaux R et O déployés sur les machines *M7* et *M8* ; 3) p_{14} permettant de générer les composants de l'investigateur de la machine *M6*. Elle génère, donc, le producteur *P7* et le consommateur *C8* et les relie aux gestionnaires *G4* et *G5* ; 4) p_{15} termine en éliminant le nœud non terminal *N("Temp")*.

Propriétés architecturales

Nous donnons dans la figure 3.12 les caractérisations des propriétés considérées au niveau middleware et en phase d'action.

$pr_{M,1'}$= (L={N1(X_P,"$P_{Inv,\alpha}$",M1,T), N2(X_G,"G_α",M1,T')} ; N={ }), Neg
$pr_{M,2'}$= (L={N1(X_P,"$P_{Inv,\alpha}$",M1,T), N2(X_G,"G_β",M1,T')} ; N={ }), Neg
$pr_{M,3'}$= (L={N1(X_P,"P_{Inv}",M1,"R"), N2(X_G,"G_α",M1,T)} ; N={ }), Neg
$pr_{M,4'}$= (L={N1(X_P,"P_{Inv}",M1,"R"), N2(X_{G1},"G_β",M1,T), N3(X_{G2},"G_β",M1,T')} ; N={ }), Neg
$pr_{M,5'}$= (L={N1(X_P,"P_{Inv}",M1,"R"), N2(X_G,"G_β",M1,"R")} ; N={N1$\xrightarrow{R}$N2}), Neg
$pr_{M,6'}$= (L={N1(X_C,"C_{Inv}",M1,"O"), N2(X_G,"G_β",M1,"O")} ; N={N2$\xrightarrow{O}$N1}), Neg

FIG. 3.12 – Les propriétés architecturales au niveau middleware et en phase d'action.

Les propriété négatives $pr_{M,1'}$ et $pr_{M,2'}$ spécifient qu'un investigateur α ne peut, respectivement, héberger ni un gestionnaire de type α ni un gestionnaire de type β. La propriété négative $pr_{M,3'}$ caractérise le fait qu'un investigateur β ne peut héberger un gestionnaire de type α tandis que l'autre propriété $pr_{M,4'}$ spécifie que ce type d'investigateur n'est pas autorisé à héberger deux investigateurs β en même temps. Finalement, les deux propriétés négatives $pr_{M,5'}$ et $pr_{M,6'}$ assurent, qu'un investigateur β ne peut (respectivement pour les types de données "R" et "O") héberger des gestionnaires de type β qui ne font pas partie de sa propre section.

3.4 Raffinement et Abstraction entre les niveaux application et middleware

Dans cette partie, nous introduisons les systèmes de raffinement et d'abstraction (ou de transformation verticale) permettant le passage d'une description à un niveau d'abstraction donné (appelé niveau initial) vers une description correspondante à un autre niveau d'abstraction (appelé niveau cible). La terminologie raffinement correspond au passage d'un haut niveau d'abstraction vers un niveau plus bas alors que l'abstraction correspond au sens inverse.

Pour la définition de ces systèmes, nous utilisons, dans ce qui suit, des grammaires de graphes edNCE étendues. Nous présentons les grammaires de graphes étendues permettant le raffinement des descriptions du niveau application vers le niveau middleware et ceci dans les phases d'exploration et d'action. Nous donnerons aussi les systèmes d'abstraction pour transformer la description du niveau middleware vers le niveau application dans les deux phases.

3.4.1 Raffinement du niveau application vers le niveau middleware (phase d'exploration)

Le raffinement du niveau application vers le niveau middleware en phase d'exploration prend en compte les spécifications introduites dans les sections de description relatives à ces deux niveaux d'abstraction. Il intègre les différentes options retenues pour générer, pour une description donnée, toutes les architectures correspondantes au niveau cible.

Nous obtenons la grammaire de graphe $GG_{(A,e)\rightarrow(M,e)}$. Étant donné que l'objectif est de passer d'une description de niveau application vers une description de niveau middleware, les nœuds non terminaux de cette grammaire correspondent aux nœuds considérés dans la description du niveau application alors que les nœuds terminaux appartiennent à la description du niveau middleware. La sémantique des labels correspond, ainsi, aux spécifications introduites dans la présentation de chaque niveau à la phase d'exploration.

$GG_{(A,e)\rightarrow(M,e)} = (G, NT, T, P)$ avec G représentant le graphe à raffiner au niveau application, et
T={N(X_P,"P_{coord}",M_P,"R"), N(X_P,N(X_C,"C_{coord}",M_C,"R") , N(X_C,"C_{coord}",M_C,"O"),
N(X_P,"P_{Inv}",M_P,"R"), N(X_P,"P_{Inv}",M_P,"O"), N(X_C,"C_{cont}",M_C,"R"), N(X_G,"G",M_G,"R"),
N(X_G,"G",M_G,"O")}}
NT={N(X_I,"Inv",M_I,"O+R"), N(X_{Coord},"Coord",φ_1,M_{Coord},"R"), N(X_{Cont},"Cont",M_{Cont})}
P={p_1,...,p_6}

Les productions de cette grammaire de graphe sont données dans la figure 3.13.

p_1= (L={N1(X_{cont},"Cont",M1)}; K={ };
R={N2(X_{C1},"C_{cont}",M1,"R"), N3(X_{G1},"G",M1,"R"), N3$\xrightarrow{R}$N2}; N={ };
C={c1}), avec
c1 = (N3,(X_{coord},"Coord",φ1,M2,"R"),"R"/"R",in,in)

p_2= (L={N1(X_{coord},"Coord",φ1,M1,"R"), N2(X_{G1},"G",M2,"R"), N3(X_{C1},"C_{cont}",M2,"R"),
N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"), N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1};
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M1,"R"), N9(X_{G3},"G",M1,"O"),
N8$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"),N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9}; N={ }; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in)
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_3= (L={N1(X_{coord},"Coord",φ1,M1,"R"),N2(X_{G1},"G",M2,"R"), N3(X_{C1},"C_{cont}",M2,"R"),
N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"), N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1};
K={N2(X_{G1},"G",M2,"R"), N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M1,"R") ,N9(X_{G3},"G",M3,"O"),
N8$\xrightarrow{R}$N6, N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"), N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9}; N={ }; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in)
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_4= (L={N1(X_{coord},"Coord",φ1,M1,"R"),N2(X_{G1},"G",M2,"R"), N3(X_{C1},"C_{cont}",M2,"R"),
N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"), N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1};
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M3,"R"),N9(X_{G3},"G",M1,"O"),
N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"),N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9}; N={ }; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in)
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_5= (L={N1(X_{coord},"Coord",φ1,M1,"R"),N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"), N1$\xrightarrow{R}$N2,
N4$\xrightarrow{O+R}$N1, N5(X_I,"Inv",M4,"O+R"), N5$\xrightarrow{O+R}$N1};
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M3,"R"),N9(X_{G3},"G",M4,"O"),
N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"), N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9, N12(X_{P4},"P_{Inv}",M4,"R"), N13(X_{P5},"P_{Inv}",M4,"O"),
N12$\xrightarrow{R}$N8 , N13$\xrightarrow{O}$N9}; N={ }; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in)
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_6= (L={N1(X_{G1},"G",M1,"R"),N2(X_{G2},"G",M2,"O"), N3(X_I,"Inv",M3,"O+R"), N3$\xrightarrow{R}$N1, N3$\xrightarrow{O}$N2};
K={N1(X_{G1},"G",M1,"R"),N2(X_{G2},"G",M2,"O")};
R={N4(X_{P1},"P_{Inv}",M3,"R"),N5(X_{P2},"P_{Inv}",M3,"O"), N4$\xrightarrow{R}$N1 , N5$\xrightarrow{O}$N2};
N={ }; C={ })

FIG. 3.13 – Les productions de la grammaire pour le raffinement du niveau application vers le niveau middleware en phase d'exploration

La production p_1 permet de raffiner le composant contrôleur de niveau application (nœud N1). Ce composant est raffiné par le consommateur d'événements R (nœud N2) envoyés par les coordinateurs, et par le gestionnaire du canal qui correspond à ces événements (nœud N3). Ces deux composants correspondent à des composants déployés sur la machine donnée par le composant contrôleur au niveau application (i.e. machine M1). L'instruction de connexion *c1* permet de traduire les liens reliant le contrôleur aux coordinateurs en canaux de communication reliant ces coordinateurs au gestionnaire du canal appartenant au contrôleur.

La production p_2 permet de raffiner un coordinateur (nœud N1) et un de ses investigateurs[29]. Ce coordinateur est raffiné par le producteur d'événements R destinés au contrôleur (nœud N5) et par deux consommateurs d'événements R et O (respectivement N6 et N7) provenant des investigateurs. Cette production exige, pour son application, l'existence d'un nœud de type "C_{cont}" et du gestionnaire du canal correspondant[30]. Le choix de ce raffinement implique le déploiement des gestionnaires de canaux, pour les événements entre le coordinateur et ses investigateurs, sur la machine du coordinateur (i.e. machine M1). L'investigateur, considéré ici, est raffiné par les deux composants correspondant aux producteurs d'événements R et O. Les deux instructions de connexion *c1* et *c2* permettent de raffiner les liens entre le coordinateur et ses investigateurs en les traduisant vers les canaux de communication O et R et en reliant les investigateurs aux gestionnaires des canaux (i.e. X_{G2} et X_{G3}).

Les productions p_3, p_4 et p_5 correspondent aussi au raffinement d'un coordinateur mais répondent à des choix de déploiement différents au niveau des gestionnaires de canaux. p_3 implique le déploiement du gestionnaire R sur la machine du coordinateur et le gestionnaire O sur la machine d'un de ses investigateurs. À la production p_4 correspond le choix inverse. La production p_5 implique un raffinement avec les deux gestionnaires de canaux qui sont déployés sur deux investigateurs.

La production p_6 correspond au raffinement des investigateurs. Un investigateur est raffiné par deux producteurs d'événements O et R. La production exige la présence des gestionnaires correspondant [31] à ces types d'événements. Le fait que ces gestionnaires correspondent au coordinateur de l'investigateur à raffiner est garanti par l'existence d'un lien entre le composant investigateur de niveau application et les gestionnaires de niveau middleware (i.e. les arcs N3$\xrightarrow{R}$N1 et N3$\xrightarrow{O}$N2) générés par les instructions de connexion appartenant aux productions p_2,...,p_5.

Le raffinement de l'exemple de niveau application donné dans la figure 3.1 vers le niveau middleware peut[32] produire l'instance de l'architecture donnée dans la figure 3.7. Ce Raffinement correspond, en partant du graphe décrivant l'architecture au niveau application, au chemin de génération suivant : 1) p_1 permettant de raffiner le contrôleur *Co* en produisant les nœuds correspondant au consommateur *C1* et au gestionnaire *G1* ; 2) p_2 permettant de raffiner le coordinateur de la machine *M2* et l'investigateur *I1* en produisant le producteur *P1*, les consommateurs *C2* et *C3*, les gestionnaires *G2* et *G3* et les producteurs *P3* et *P4* de l'investigateur *I1* ; 3) p_6 permettant de raffiner l'investigateur *I2* en générant les producteurs *P5* et *P6* ; 4) p_6 pour raffiner l'investigateur *I3* et générer les producteurs *P7* et *P8* ; et finalement, 5) p_5 permettant de raffiner le coordinateur de la machine *M3* et les investigateurs *I4* et *I5* en produisant le

[29] Cet investigateur existe puisqu'un coordinateur coordonne au moins un investigateur.

[30] Ceci force l'ordre d'application des productions. p_1 sera forcément appliquée avant les productions p_2, ..., p_5.

[31] Ce qui détermine un ordre d'application des productions imposant le raffinement des coordinateurs avant le raffinement des investigateurs qu'ils coordonnent.

[32] l'instance donnée dans la figure 3.1 n'est pas le seul raffinement possible de celle donnée dans la figure 3.7. Ceci est due aux multiples choix de déploiement possibles pour les gestionnaires de canaux.

producteur *P2* et les consommateurs *C4* et *C5*, le gestionnaire *G4* et les producteurs *P9* et *P10* déployés sur la machine *M7*, et les producteurs *P11* et *P12* déployés sur la machine *M8*.

3.4.2 Abstraction du niveau middleware vers le niveau application (phase d'exploration)

L'abstraction des descriptions appartenant au niveau middleware en phase d'exploration vers les descriptions correspondantes dans le niveau application correspond au système de transformation inverse à celui défini dans la section précédente.

Les terminaux et les non terminaux de la grammaire $GG_{(A,e)\rightarrow(M,e)}$ constituent respectivement les non terminaux et les terminaux de la grammaire $GG_{(M,e)\rightarrow(A,e)}$.

$GG_{(M,e)\rightarrow(A,e)}$=(G,NT,T,P) avec G représentant le graphe qui décrit l'application au niveau middleware :
T={N(X_I,"Inv",M_I,"O+R"), N(X_{coord},"Coord",φ_1, M_{coord},"R"), N(X_{Cont},"Cont",M_{Cont})}
NT={N(X_P,"P_{coord}",M_P,"R"), N(X_C,"C_{coord}",M_C,"R"), N(X_C,"C_{coord}",M_C,"O"),
N(X_P,"P_{Inv}",M_P,"R"), N(X_P,"P_{Inv}",M_P,"O"), N(X_C,"C_{cont}",M_C,"R"), N(X_G,"G",M_G,"R"),
N(X_G,"G",M_G,"O")}
P={p_1,p_2,p_3}

p_1= (L={N1(X_{C1},"C_{cont}",M1,"R"),N2(X_{G1},"G",M1,"R"),N2$\xrightarrow{R}$N1} ; K={ } ; R={N3(X_{cont},"Cont",M1)} ; N={ } ; C={c1}), avec c1 = (N2,(X_P,"P_{coord}",M2,"R"),"R"/"R",in,in)
p_2= (L={N1(X_{P1},"P_{coord}",M1,"R"), N2(X_{cont},"Cont",M2), N1$\xrightarrow{R}$N2, N3(X_{c1},"C_{coord}",M1,"R"), N4(X_{c2},"C_{coord}",M1,"O"), N5(X_{G1},"G",M3,"R"), N6(X_{G2},"G",M4,"O"), N5$\xrightarrow{R}$N3 , N6$\xrightarrow{O}$N4} ; K={N2(X_{cont},"Cont",M2)} ; R={N7(X_{coord},"Coord",φ1,M1,"R"), N7$\xrightarrow{R}$N2} ; N={ } ; C={c1,c2}), avec c1 = (N7,(X_{P2},"P_{Inv}",M5,"R"),"R"/"R",in,in) c2 = (N7,(X_{P3},"P_{Inv}",M5,"O"),"O"/"O",in,in)
p_3= (L={N1(X_{coord},"Coord",φ1,M1,"R"), N2(X_{P1},"P_{Inv}",M2,"R"), N3(X_{P2},"P_{Inv}",M2,"O"), N2$\xrightarrow{R}$N1,N3$\xrightarrow{O}$N1} ; K={N1(X_{coord},"Coord",φ1,M1,"R")} ; R={N4(X_I,"Inv",M2,"O+R"), N4$\xrightarrow{O+R}$N1} ; N={ } ; C={ })

FIG. 3.14 – Les productions pour l'abstraction du niveau middleware vers le niveau application en phase d'exploration

La production p_1 permet d'abstraire les composants de niveau middleware constituant un contrôleur. L'appartenance d'un consommateur d'événements R (nœud N1) et d'un gestionnaire de ces événements (nœud N2) au même contrôleur est garantie par le fait qu'ils sont déployés sur la même machine (machine M1). Ces deux composants sont transformés en un composant de plus haut niveau de type contrôleur (nœud N3). Les canaux reliant les producteurs d'événements de type R vers le consommateur du contrôleur sont redirigés par l'instruction de connexion *c1* vers le composant contrôleur. Ceci permet de le relier correctement aux coordinateurs.

La production p_2 permet le traitement des composants relatifs aux coordinateurs. Elle transforme le producteur d'événements R et les consommateurs d'événements R et O (l'appartenance de ces composants au même coordinateur est justifiée par le fait qu'ils soient déployés sur la même machine, i.e. M1) et les gestionnaires de canaux pour les données R et O. Ces composants appartenant au niveau middleware sont transformés pour obtenir le composant coordinateur. Cette production couvre tous les cas de figure correspondant aux différentes possibilités de déploiement des gestionnaires. En effet, les machines

correspondantes aux gestionnaires sont représentées par des variables (M3 et M4) et peuvent donc être unifiées avec la même valeur attribuée à la variable correspondant à la machine du coordinateur (machine M1). Les instructions de connexion *c1* et *c2* permettent de rediriger les liens reliant les producteurs des investigateurs aux gestionnaires de canaux vers le coordinateur correspondant à ces gestionnaires de canaux.

La production p_3 traite le cas des investigateurs. Les producteurs de données O et R (déployés sur une même machine) sont transformés en un composant investigateur appartenant au niveau application.

3.4.3 Raffinement du niveau application vers le niveau middleware (phase d'action)

Nous traitons ici le raffinement des descriptions au niveau application vers le niveau middleware et ceci en considérant la phase d'action. Les nœuds produits par la grammaire $GG_{(A,a)}$ constituent des non terminaux et les nœuds produits par la grammaire $GG_{(M,a)}$ des terminaux. La grammaire de graphes $GG_{(A,a)\to(M,a)}$ est définie telle que $GG_{(A,a)\to(M,a)}$=(G,NT,T,P) avec G représentant le graphe qui décrit une instance de l'architecture au niveau application :

T={N(X_P,"P_{coord}",M_P,"R"), N(X_C,"C_{coord}",M_C,"R"), N(X_C,"C_{coord}",M_C,"O"),
N(X_P,"$P_{Inv,\alpha}$",M_P,"R"), N(X_P,"$P_{Inv,\alpha}$",M_P,"O"), N(X_C,"C_{Inv}",M_C,"O"),
N(X_P,"P_{Inv}",M_P,"R"), N(X_P,"P_{Inv}",M_P,"O"), N(X_C,"C_{cont}",M_C,"R"),
N(X_G,"G",M_G,"R"), N(X_G,"G",M_G,"O"), N(X_G,"G_α",M_G,"R"),
N(X_G,"G_α",M_G,"O"), N(X_G,"G_β",M_G,"R"), N(X_G,"G_β",M_G,"O")}
NT={N(X_I,"Inv_α",M_I,"O+R"), N(X_I,"Inv",M_I,"R"), N(X_I,"Inv",M_I,"O+R"),
N(X_{Coord},"Coord",φ1,M_{Coord},"R"), N(X_{Coord},"Coord",φ2,M_{Coord},"R"),
N(X_{Cont},"Cont",M_{Cont})}
$P = \{p_1, \ldots, p_{11}\}$

Les productions de cette grammaire de graphe sont données dans la figure 3.15.

p_1= (L={N1(X_{cont},"Cont",M1)};K={ };
R={N2(X_{C1},"C_{cont}",M1,"R"),N3(X_{G1},"G",M1,"R"),N3$\xrightarrow{R}$N2}; N={ };
C={c1,c2}), avec
c1 = (N3,(X_{coord},"Coord",φ1,M2,"R"),"R"/"R",in,in), et
c2 = (N4,(X_{coord},"Coord",φ2,M3,"R"),"R"/"R",in,in)

p_2= (L={N1(X_{coord},"Coord",φ2,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_I,"Inv_α",M3,"O+R"), N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1 };
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"*Coord*",M1,"O"), N8(X_{c3},"Coord",M1,"R"), N9(X_{G2},"G_α",M1,"R"),
N10(X_{G3},"G_α",M1,"O"), N11(X_{G4},"G_β",M1,"R"),N12(X_{G5},"G_β",M1,"O"),
N9$\xrightarrow{R}$N6 , N10$\xrightarrow{O}$N7, N11$\xrightarrow{O}$N8, N13(X_{P2},"$P_{Inv,\alpha}$",M3,"R"),
N14(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N15(X_{P2},"$P_{Inv,\alpha}$",M3,"O"),
N13$\xrightarrow{R}$N9 , N14$\xrightarrow{O}$N10 , N15$\xrightarrow{O}$N12};
N={ }; C={c1,c2}), avec
c1 = (N11,(X_{I2},"Inv",M4,"R"),"R"/"R",in,in)
c2 = (N12,(X_{I2},"Inv",M4,"O"),"O"/"O",out,out)

p_3= (L={N1(X_{coord},"Coord",φ2,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_{I1},"Inv_{α}",M3,"O+R"), N1$\xrightarrow{R}$N2,
N4$\xrightarrow{O+R}$N1, N5(X_{I2},"Inv",M4,"R"), N5$\xrightarrow{R}$N1, N4$\xrightarrow{O}$N5};
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N6(X_{P1},"P_{coord}",M1,"R"), N6$\xrightarrow{R}$N2, N7(X_{c1},"Coord",M1,"R"),
N8(X_{c2},"Coord",M1,"O"), N9(X_{c3},"Coord",M1,"R"),
N10(X_{G2},"G_{α}",M1,"R"), N11(X_{G3},"G_{α}",M1,"O"), N12(X_{G4},"G_{β}",M1,"R"),
N13(X_{G5},"G_{β}",M4,"O"), N10$\xrightarrow{R}$N7 , N11$\xrightarrow{O}$N8, N19$\xrightarrow{O}$N9,
N14(X_{P2},"$P_{Inv,\alpha}$",M3,"R"), N15(X_{P3},"$P_{Inv,\alpha}$",M3,"O"),
N16(X_{P2},"$P_{Inv,\alpha}$",M3,"O"), N14$\xrightarrow{R}$N10 , N15$\xrightarrow{O}$N11, N16$\xrightarrow{O}$N13,
N17(X_{P2},"P_{Inv}",M4,"R"),N18(X_{c4},"C_{α}",M4,"O"), N17$\xrightarrow{R}$N12 , N13$\xrightarrow{O}$N18};
N={ }; C={c1,c2}), avec
c1 = (N12,(X_{I3},"Inv",M4,"R"),"R"/"R",in,in)
c2 = (N13,(X_{I3},"Inv",M4,"O"),"O"/"O",out,out)

p_4= (L={N1(X_{coord},"Coord",φ2,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_{I1},"Inv_{α}",M3,"O+R"), N1$\xrightarrow{R}$N2,
N4$\xrightarrow{O+R}$N1, N5(X_{I2},"Inv",M4,"R"), N5$\xrightarrow{R}$N1, N4$\xrightarrow{O}$N5};
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N6(X_{P1},"P_{coord}",M1,"R"), N6$\xrightarrow{R}$N2, N7(X_{c1},"Coord",M1,"R"),
N8(X_{c2},"Coord",M1,"O"), N9(X_{c3},"Coord",M1,"R"),
N10(X_{G2},"G_{α}",M1,"R"),N11(X_{G3},"G_{α}",M1,"O"),
N12(X_{G4},"G_{β}",M4,"R"),N13(X_{G5},"G_{β}",M1,"O"), N10$\xrightarrow{R}$N7 ,
N11$\xrightarrow{O}$N8, N19$\xrightarrow{O}$N9, N14(X_{P2},"$P_{Inv,\alpha}$",M3,"R"),
N15(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N16(X_{P2},"$P_{Inv,\alpha}$",M3,"O"), N14$\xrightarrow{R}$N10,
N15$\xrightarrow{O}$N11, N16$\xrightarrow{O}$N13, N17(X_{P2},"P_{Inv}",M4,"R"),N18(X_{c4},"C_{Inv}",M4,"O"),
N17$\xrightarrow{R}$N12 , N13$\xrightarrow{O}$N18};
N={ }; C={c1,c2}) avec
c1 = (N12,(X_{I3},"Inv",M4,"R"),"R"/"R",in,in)
c2 = (N13,(X_{I3},"Inv",M4,"O"),"O"/"O",out,out)

p_5= (L={N1(X_{coord},"Coord",φ2,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_{I1},"Inv_{α}",M3,"O+R"),
N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1, N5(X_{I2},"Inv",M4,"R"), N5$\xrightarrow{R}$N1,
N4$\xrightarrow{O}$N5, N6(X_{I2},"Inv",M5,"R"), N6$\xrightarrow{R}$N1, N4$\xrightarrow{O}$N6};
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3};
R={N6(X_{P1},"P_{coord}",M1,"R"), N6$\xrightarrow{R}$N2, N7(X_{c1},"Coord",M1,"R"),
N8(X_{c2},"Coord",M1,"O"), N9(X_{c3},"Coord",M1,"R"),
N10(X_{G2},"G_{α}",M1,"R"),N11(X_{G3},"G_{α}",M1,"O"),
N12(X_{G4},"G_{β}",M4,"R"),N13(X_{G5},"G_{β}",M5,"O"),
N10$\xrightarrow{R}$N7 , N11$\xrightarrow{O}$N8, N19$\xrightarrow{O}$N9, N14(X_{P2},"$P_{Inv,\alpha}$",M3,"R"),
N15(X_{P3},"$P_{Inv,\alpha}$",M3,"O"), N16(X_{P2},"$P_{Inv,\alpha}$",M3,"O"), N14$\xrightarrow{R}$N10 ,
N15$\xrightarrow{O}$N11, N16$\xrightarrow{O}$N13, N17(X_{P2},"P_{Inv}",M4,"R"),N18(X_{c4},"C_{Inv}",M4,"O"),
N17$\xrightarrow{R}$N12, N13$\xrightarrow{O}$N18, N19(X_{P2},"P_{Inv}",M4,"R"),N20(X_{c4},"C_{Inv}",M4,"O"),
N19$\xrightarrow{R}$N12 , N13$\xrightarrow{O}$N20}
N={ }; C={c1,c2}) avec
c1 = (N12,(X_{I3},"Inv",M4,"R"),"R"/"R",in,in)
c2 = (N13,(X_{I3},"Inv",M4,"O"),"O"/"O",out,out)

p_6= (L={N1(X_{coord},"Coord",φ1,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"), N1$\xrightarrow{R}$N2,
N4$\xrightarrow{O+R}$N1 } ;
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3} ;
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M1,"R"), N9(X_{G3},"G",M1,"O"),
N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"), N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8, N11$\xrightarrow{O}$N9}
N={ } ; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in) et
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_7= (L={N1(X_{coord},"Coord",φ1,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"),
N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1 } ;
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3} ;
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M1,"R"),N9(X_{G3},"G",M3,"O"),
N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"),N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9} ;
N={ } ; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in) et
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_8= (L={N1(X_{coord},"Coord",φ1,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"),
N1$\xrightarrow{R}$N2, N4$\xrightarrow{O+R}$N1 } ;
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3} ;
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M3,"R"),N9(X_{G3},"G",M1,"O"),
N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"),N11(X_{P3},"P_{Inv}",M3,"O"),
N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9} ;
N={ } ; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in) et
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_9= (L={N1(X_{coord},"Coord",φ1,M1,"R"), N2(X_{G1},"G",M2,"R"),
N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3, N4(X_I,"Inv",M3,"O+R"), N1$\xrightarrow{R}$N2,
N4$\xrightarrow{O+R}$N1, N5(X_I,"Inv",M4,"O+R"), N5$\xrightarrow{O+R}$N1} ;
K={N2(X_{G1},"G",M2,"R"),N3(X_{C1},"C_{cont}",M2,"R"), N2$\xrightarrow{R}$N3} ;
R={N5(X_{P1},"P_{coord}",M1,"R"), N5$\xrightarrow{R}$N2, N6(X_{c1},"Coord",M1,"R"),
N7(X_{c2},"Coord",M1,"O"), N8(X_{G2},"G",M3,"R"),N9(X_{G3},"G",M4,"O"),
N8$\xrightarrow{R}$N6 , N9$\xrightarrow{O}$N7, N10(X_{P2},"P_{Inv}",M3,"R"),
N11(X_{P3},"P_{Inv}",M3,"O"), N10$\xrightarrow{R}$N8 , N11$\xrightarrow{O}$N9,
N12(X_{P4},"P_{Inv}",M4,"R"), N13(X_{P5},"P_{Inv}",M4,"O"), N12$\xrightarrow{R}$N8,
N13$\xrightarrow{O}$N9} ;
N={ } ; C={c1,c2}), avec
c1 = (N8,(X_{I2},"Inv",M4,"O+R"),"O+R"/"R",in,in)
c2 = (N9,(X_{I2},"Inv",M4,"O+R"),"O+R"/"O",in,in)

p_{10}= (L={N1(X_{G1},"G_β",M1,"R"),N2(X_{G2},"G_β",M2,"O"),
N3(X_I,"Inv",M3,"R"), N3$\xrightarrow{R}$N1, N2$\xrightarrow{O}$N3};
K={N1(X_{G1},"G",M1,"R"),N2(X_{G2},"G",M2,"O")};
R={N4(X_{P1},"P_{Inv}",M3,"R"), N5(X_{C1},"C_{Inv}",M3,"O"), N4$\xrightarrow{R}$N1, N2$\xrightarrow{O}$N5};
N={ }; C={ })

p_{11}= (L={N1(X_{G1},"G",M1,"R"),N2(X_{G2},"G",M2,"O"),
N3(X_I,"Inv",M3,"O+R"), N3$\xrightarrow{R}$N1, N3$\xrightarrow{O}$N2};
K={N1(X_{G1},"G",M1,"R"),N2(X_{G2},"G",M2,"O")};
R={N4(X_{P1},"P_{Inv}",M3,"R"),N5(X_{P2},"P_{Inv}",M3,"O"), N4$\xrightarrow{R}$N1, N5$\xrightarrow{O}$N2};
N={ }; C={ })

FIG. 3.15 – Productions de grammaire pour le raffinement du niveau application vers le niveau middleware en phase d'action

Étant donné que le contrôleur ne change pas de structure interne entre la phase d'investigation et la phase d'action, la production p_1 de la grammaire $GG_{(A,a)\rightarrow(M,a)}$ reste la même que celle de la grammaire $GG_{(A,e)\rightarrow(M,e)}$. La seule différence réside dans la prise en compte, par les instructions de connexion, des coordinateurs en phase d'action (instruction de connexion *c2*).

La production p_2 concerne le raffinement des coordinateurs en phase d'action. Pour ces coordinateurs, nous générons le producteur d'événements R à destination du contrôleur (nœud N5) les deux consommateurs d'événements O et R (respectivement nœuds N6 et N7) et les gestionnaires de canaux G_α pour les événements O et R échangés avec l'investigateur α et les gestionnaires G_β pour les événements O échangés entre l'investigateur α et les autres investigateurs et pour les événements R envoyés par ces investigateurs vers le coordinateur. Cette production permet en même temps de raffiner le composant représentant l'investigateur α par les composants producteurs d'événements R et O pour le coordinateur et le producteur d'événements O pour les autres investigateurs. Les instructions de connexion permettent de rediriger les canaux de communication reliant les investigateurs β vers les gestionnaires β correspondant au type d'événements traités (i.e. le nœud N11 pour les événements R et le nœud N12 pour les événements O).

La production p_2 prend en compte un déploiement des gestionnaires β sur le coordinateur. Les productions p_3, p_4 et p_5 correspondent aux autres choix de déploiement possibles. p_3 correspond au déploiement du gestionnaire β pour les événements O sur le coordinateur et au déploiement du gestionnaire pour les données O sur un investigateur β. p_4 correspond au choix de déploiement inverse alors que la production p_5 correspond au déploiement des deux gestionnaires β sur deux investigateurs différents.

Les productions p_6,...,p_9 correspondent respectivement aux productions p_2,...,p_5 de la grammaire $GG_{(A,a)\rightarrow(M,a)}$ mais traitent le raffinement des branches correspondant aux coordinateurs en phase d'exploration.

La production p_{10} correspond au raffinement des investigateurs β. Leur condition β est vérifiée par rapport au fait qu'ils sont reliés à des gestionnaires de type β (nœuds N1 et N2). Ce raffinement correspond à la génération du consommateur d'événements O envoyés par l'investigateur α (nœud N5) et à la génération du producteur d'événements R à destination du coordinateur (nœud N4) et au raccordement de ces composants aux gestionnaires de canaux correspondants (i.e. respectivement les nœuds N2 et N1).

La production p_{11} traite le cas des investigateurs appartenant aux branches des coordinateurs en phase d'exploration. Cette production correspond à la production p_6 de la grammaire $GG_{(A,e)\rightarrow(M,e)}$.

3.4.4 Abstraction du niveau middleware vers le niveau application (phase d'action)

Le système de transformation verticale permettant l'abstraction des descriptions appartenant au niveau middleware en phase d'action vers le niveau application est donné par la grammaire de graphe $GG_{(M,a)\rightarrow(A,a)}$, avec $GG_{(M,a)\rightarrow(A,a)} = (G, NT, T, P)$. G correspond à un graphe décrivant une instance

de l'architecture au niveau middleware.
T={N(X_I,"Inv_α",M_I,"O+R"), N(X_I,"Inv",M_I,"R"), N(X_I,"Inv",M_I,"O+R"),
N(X_{Coord},"Coord",φ1,M_{Coord},"R"), N(X_{Coord},"Coord",φ2,M_{Coord},"R"), N(X_{Cont},"Cont",M_{Cont})}
NT={N(X_P,"P_{coord}",M_P,"R"), N(X_C,"C_{coord}",M_C,"R"), N(X_C,"C_{coord}",M_C,"O"),
N(X_P,"$P_{Inv,\alpha}$",M_P,"R"), N(X_P,"$P_{Inv,\alpha}$",M_P,"O"), N(X_C,"C_{Inv}",M_C,"O"),
N(X_P,"P_{Inv}",M_P,"R"), N(X_P,"P_{Inv}",M_P,"O"), N(X_C,"C_{cont}",M_C,"R"),
N(X_G,"G",M_G,"R"), N(X_G,"G",M_G,"O"), N(X_G,"G_α",M_G,"R"),
N(X_G,"G_α",M_G,"O"), N(X_G,"G_β",M_G,"R"), N(X_G,"G_β",M_G,"O")}
P={p_1,...,p_5}

Les productions de cette grammaire sont données dans la figure 3.16.

p_1= (L={N1(X_{C1},"C_{cont}",M1,"R"),N2(X_{G1},"G",M1,"R"),N2$\xrightarrow{R}$N1}; K={ }; R={N3(X_{cont},"Cont",M1)}; N={ }; C={c1}), avec c1 = (N2,(X_P,"P_{Coord}",M2,"R"),"R"/"R",in,in)
p_2= (L={N1(X_{P1},"P_{coord}",M1,"R"), N2(X_{cont},"Cont",M2), N1$\xrightarrow{R}$N2, N3(X_{c1},"C_{coord}",M1,"R"),N4(X_{c2},"C_{coord}",M1,"O"), N5(X_{c3},"C_{coord}",M1,"R"), N6(X_{G1},"G_α",M1,"R"),N7(X_{G2},"G_α",M1,"O"), N6$\xrightarrow{R}$N3, N7$\xrightarrow{O}$N4, N8(X_{G3},"G_β",M3,"R"), N8$\xrightarrow{R}$N5, N9(X_{P2},"$P_{Inv,\alpha}$",M4,"R"), N10(X_{P3},"$P_{Inv,\alpha}$",M4,"O"), N11(X_{P4},"$P_{Inv,\alpha}$",M4,"O"), N9$\xrightarrow{R}$N6, N10$\xrightarrow{O}$N7, N12(X_{G4},"G_β",M6,"O"), N11$\xrightarrow{O}$N12, K={N2(X_{Cont},"Cont",M2)}; R={N13(X_{Coord},"Coord","φ2",M1,"R"), N14(X_I,"Inv_α",M4,"O+R"), N13$\xrightarrow{R}$N2, N14$\xrightarrow{O+R}$N13}; N={ }; C={c1,c2}), avec c1 = (N13,(X_{P5},"P_{Inv}",M7,"R"),"R"/"R",in,in) c2 = (N14,(X_{c4},"C_{Inv}",M7,"O"),"O"/"O",out,out)
p_3= (L={N1(X_{P1},"P_{coord}",M1,"R"), N2(X_{Cont},"Cont",M2), N1$\xrightarrow{R}$N2, N3(X_{c1},"C_{coord}",M1,"R"), N4(X_{c2},"C_{coord}",M1,"O"), N5(X_{G1},"G",M3,"R"),N6(X_{G2},"G",M4,"O"), N5$\xrightarrow{R}$N3 , N6$\xrightarrow{O}$N4}; K={N2(X_{Cont},"Cont",M2)}; R={N7(X_{Coord},"Coord","φ1",M1,"R"), N7$\xrightarrow{R}$N2} N={ }; C={c1,c2}), avec c1 = (N7,(X_{P2},"P_{Inv}",M5,"R"),"R"/"R",in,in) c2 = (N7,(X_{P3},"P_{Inv}",M5,"O"),"O"/"O",in,in)
p_4= (L={N1(X_{coord},"Coord","φ2",M1,"R"), N2(X_I,"Inv_α",M2,"O+R"), N2$\xrightarrow{O+R}$N1 N3(X_{P1},"P_{Inv}",M3,"R"), N4(X_{C1},"C_{Inv}",M3,"O"), N3$\xrightarrow{R}$N1, N2$\xrightarrow{O}$N4}; K={N1(X_{coord},"Coord","φ2",M1,"R"), N2(X_I,"Inv_α",M2,"O+R"), N2$\xrightarrow{O+R}$N1}; R={N5(X_I,"Inv",M3,"R"), N5$\xrightarrow{R}$N1 , N2$\xrightarrow{O}$N5}; N={ }; C={ })
p_5= (L={N1(X_{coord},"Coord","φ1",M1,"R"), N2(X_{P1},"P_{Inv}",M2,"R"), N3(X_{P2},"P_{Inv}",M2,"O"), N2$\xrightarrow{R}$N1, N3$\xrightarrow{O}$N1}; K={N1(X_{coord},"Coord","φ1",M1,"R")}; R={N4(X_I,"Inv",M2,"O+R"), N4$\xrightarrow{O+R}$N1}; N={ }; C={ })

FIG. 3.16 – Les productions de la grammaire $GG_{(M,a)\to(A,a)}$

La production p_1 permet d'abstraire les composants de niveau middleware constituant un contrôleur.

Elle correspond à la production p_1 de la grammaire $GG_{(M,a)\rightarrow(A,a)}$ du fait que le contrôleur ne change pas de structure interne entre les deux phases d'exploration et d'action.

La production p_2 correspond à l'abstraction des composants constituant les coordinateurs en phase d'action. Cette production consomme les nœuds correspondant aux producteurs d'événements R pour le contrôleur, les nœuds correspondant aux consommateurs d'événements R et O (respectivement nœuds N3 et N4) provenant de l'investigateur α et des événements R (nœud N5) provenant des investigateurs β, les nœuds correspondant à l'investigateur α (nœuds N10, N11 et N12) et les nœuds relatifs aux quatre gestionnaires de canaux (nœuds N6, N7, N8 et N13). Cette production produit les deux composants de niveau application correspondant au coordinateur et à l'investigateur α. Les instructions de connexion *c1* et *c2* permettent respectivement de traiter les canaux reliant les producteurs d'événements R des investigateurs β et le coordinateur et les canaux reliant l'investigateur α et les consommateurs appartenant aux autres investigateurs.

La production p_3 traite le cas des branches correspondant à des coordinateurs en phase d'exploration. Cette production est équivalente à la production p_2 de la grammaire $GG_{(M,e)\rightarrow(A,e)}$.

La production p_4 implémente l'abstraction des investigateurs β dans une branche coordonnée par un coordinateur en phase d'action. Le producteur d'événements R et le consommateur d'événements O déployés sur une même machine sont remplacés par un composant de niveau application correspondant à un investigateur β. Ce composant est relié avec un lien R au coordinateur et avec un lien O à l'investigateur α.

La production p_5 traite le cas des investigateurs gérés par un coordinateur en phase d'exploration. Cette production est équivalente à la production p_3 de la grammaire $GG_{(M,e)\rightarrow(A,e)}$.

3.5 Description de l'architecture dynamique au niveau réseau

Le niveau réseau constitue le niveau d'abstraction le plus bas pour notre cas d'étude. Il correspond aux communications de processus à processus, c'est-à-dire les connexions de niveau transport qui raffinent les communications de composants à composants du niveau middleware. Nous considérons à ce niveau, les types de données échangées en prenant en compte des messages audio, vidéo et texte ainsi que la qualité de service relatives aux échanges de ces messages.

Les données de type O sont des données descriptives et sont transmises sous forme audio/vidéo (av) tandis que les données de type R sont des données produites et expriment l'analyse de la situation par un investigateur ou un coordinateur. Elles sont transmises sous forme textuelle (t).

Au niveau de la qualité de service nous considérons trois niveaux : *high*, *medium* et *low*. Différentes ressources et protocoles peuvent correspondre au niveau de QdS défini pour chaque connexion au niveau réseau. Nous nous limitons, dans ce cas d'étude, à définir une exigence du niveau de la QdS sans rentrer dans les détails de l'implémentation permettant de la garantir (e.g. protocole, bande passante ...).

Les gestionnaires de canaux au niveau middleware sont raffinés par un serveur au niveau réseau tandis que les producteurs et les consommateurs sont raffinés par des clients. Les clients correspondant aux producteurs de données envoient des requêtes aux serveurs pour y déposer des messages (push) tandis que les clients correspondant à des consommateurs récupèrent les messages en retour des requêtes envoyées aux serveurs (pull).

3.5.1 Description au niveau réseau en phase d'exploration

Les messages de rapport (de type textuel) envoyés par les coordinateurs vers le contrôleur permettent à ce dernier d'avoir une vision globale de la situation sur le terrain. Ces messages se verront, donc, attribué une exigence de QdS maximale. Cela correspond à des connexions avec une exigence de QdS *(t,high)* où "t" exprime que le message est de type texte et "*high*" correspond au niveau de QdS maximal.

Pour les mêmes raisons (i.e. permettre au contrôleur d'avoir une vision globale du contexte), les connexions relatives à l'envoi des messages d'observation par les investigateurs vers les coordinateurs (de type audio et vidéo) correspondent à une exigence de QdS maximale. Nous obtenons des connexions avec une QdS correspondant à *(av,high)*. Étant donné que les coordinateurs disposent de systèmes de décision et d'analyse propres, les connexions correspondantes aux messages de rapport envoyés par les

investigateurs auront une QdS inférieure à celle des messages d'observation. Ces connexions disposent de la QdS correspondant à *(t,med)*.

Nous considérons, donc, les nœuds suivants pour la description des composants appartenant au niveau réseau et à la phase d'exploration

- Les clients du contrôleur permettant de consommer les messages envoyés par les coordinateurs sont décrits par le nœud *N("C","Cl_{cont}","M1","M2","t","high")* où le label *C* correspond à l'identifiant du client, Cl_{cont} indique que le composant est un client qui fait partie du contrôleur, *M1* correspond à la machine de déploiement du composant (i.e. la machine du contrôleur), *M2* correspond à la machine du coordinateur qui produit les messages pour ce client[33], *t* indique que ce composant traite des messages de type textuel et *high* indique que la QdS relative aux communications de ce client correspond au niveau le plus haut.
- Les serveurs qui traitent les communications entre les coordinateurs et le contrôleur sont décrits par le nœud *N("S1","S_{cont},"M1","t","high")* où *S1* est l'identifiant du serveur, S_{cont} indique qu'il s'agit du serveur du contrôleur, *M1* correspond à la machine de déploiement, *t* et *high* indique la QdS relative à ce serveur.
- Les clients permettant l'envoie des messages de rapport des coordinateurs vers le contrôleur sont décrits par le nœud *N("C1","Cl_{coord}","M1","t","high")* où *C1* est l'identifiant du client, Cl_{coord} est son type, *M1* est la machine de déploiement, et *t* et *high* correspondent à la QdS requise pour ce client.
- Les clients appartenant aux coordinateurs et qui consomment les messages produits par les investigateurs sont décrits par des nœuds du type *N("C1","Cl_{coord}","M1","M2",x,y)* où *C1* est l'identifiant du client, Cl_{coord} son type, *M1* est la machine sur laquelle il est déployé, *M2* la machine d'origine des messages qu'il consomme, alors que les labels *x* et *y* correspondent à la QdS (*x* et *y* correspondent respectivement à *av* et *high* quand il s'agit des messages d'observation et correspondent à *t* et *med* quand il s'agit des messages de rapport).
- Les serveurs en charge de la gestion des communications entre les investigateurs et les coordinateurs sont décrits par les nœuds *N("S1","S_{coord}","M1",x,y)* où *S1* est l'identifiant du serveur, S_{coord} son type, *M1* la machine sur laquelle il est déployé. *x* et *y* correspondent à la QdS (*av* et *high* pour les messages d'observation, et *t* et *med* pour les messages de rapport).
- Finalement, les clients permettant l'envoi des messages des investigateurs sont décrits par les nœuds *N("C1","Cl_{Inv}","M1",x,y)* où *C1* est l'identifiant, Cl_{Inv} correspond au type, *M1* est la machine de déploiement tandis que *x* et *y* désignent la QdS exigée pour ces clients (*av* et *high* pour les messages d'observation, et *t* et *med* pour les messages de rapport).

Le déploiement des serveurs considérés au niveau réseau répond aux mêmes considérations que celles des gestionnaires de canaux pour le niveau middleware : le serveur en charge des communications entre les coordinateurs et le contrôleur est toujours déployé sur la machine qui héberge le contrôleur. Les serveurs en charge des communications entre les investigateurs et les coordinateurs peuvent être déployés ou bien sur la machine du coordinateur ou bien sur celles des investigateurs[34].

Une instance de l'architecture est donnée dans la figure 3.17. Cette instance présente, pour le contrôleur, les deux clients *C1* et *C2* qui consomment les messages envoyés respectivement par les clients *C3* et *C10* correspondant aux coordinateurs déployés sur les machines *M2* et *M3*. Le serveur chargé de cette communication correspond au serveur *S1*.

Le coordinateur de la machine *M2* est constitué : 1) du client *C3* qui produit des rapports pour le contrôleur, 2) des clients *C4*, *C5* et *C6* qui consomment les rapports (i.e. type *t,med*) des investigateurs hébergés par les machines *M4*, *M5* et *M6*, 3) des clients *C7*, *C8* et *C9* qui consomment les messages d'observation (i.e. type *av,high*) des investigateurs. Les serveurs qui assurent la connexion entre ce coordinateur et ses investigateurs (i.e. *S2* et *S3*) sont déployés sur la machine du coordinateur. Les trois investigateurs gérés par le coordinateur de la machine M2 sont constitués, chacun, par deux clients (i.e. *C15*/*C16*, *C17*/*C18* et *C19*/*C20*) produisant les deux types de messages avec deux niveaux de QdS (*t,med*) et (*av,high*).

Le coordinateur de la machine *M3* est constitué : 1) du client *C10* produisant les rapports pour le

[33] Pour chaque coordinateur qu'il gère, le contrôleur initie un client pour consommer ses messages.

[34] avec toujours la limitation interdisant le déploiement de plus d'un serveur sur une même machine d'un investigateur.

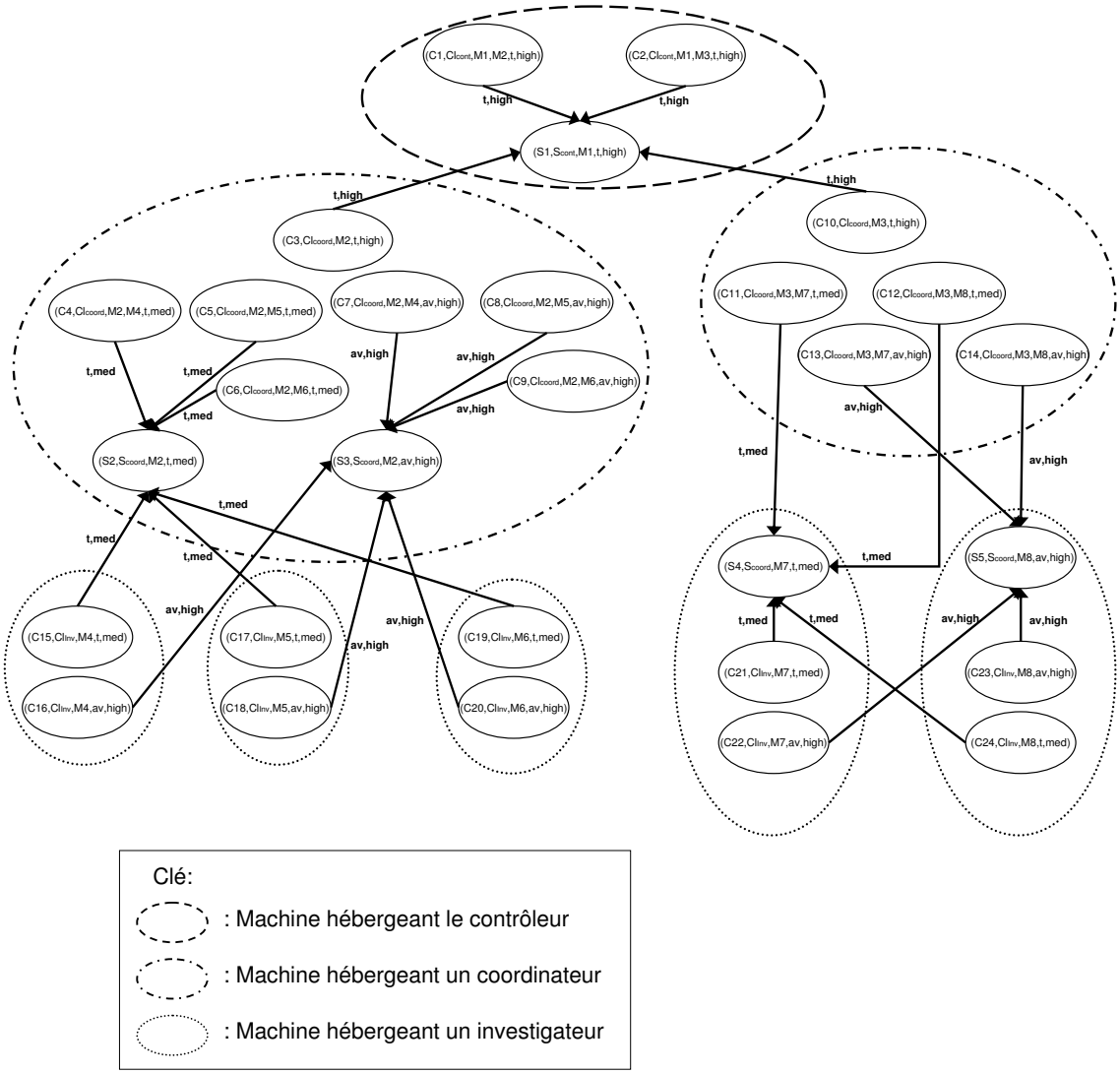


FIG. 3.17 – Une instance de l'architecture au niveau réseau dans la phase d'exploration

contrôleur, 2) des clients *C11* et *C12* qui consomment les rapports des investigateurs des machines *M7* et *M8*, 3) des clients *C13* et *C14* qui consomment les messages d'observation. Les serveurs reliant ce coordinateur à ses investigateurs sont déployés sur les machines *M7* et *M8* correspondant à des investigateurs. Les deux investigateurs gérés par le coordinateur de la machine *M3* contiennent aussi les deux clients qui produisent les messages d'observation et de rapport pour le coordinateur.

Les instances consistantes de l'architecture de l'application au niveau réseau et en phase d'exploration sont décrites par la grammaire $GG_{R,e}$=(AX,NT,T,P) avec :
T={N(X_C,"Cl_{cont}",M_{cont},M_{cont},"t","high"), N(X_S,"S_{cont}",M_{cont},"t","high"),
N(X_C,"Cl_{coord}",M_{coord},"t","high"), N(X_C,"Cl_{coord}",M_{coord},M_{Inv},"t","med"),
N(X_C,"Cl_{coord}",M_{coord},M_{Inv},"av","high"),N(X_S,"S_{coord}",M,"t","med"),
N(X_S,"S_{coord}",M,"av","high"), N(X_C,"Cl_{Inv}",M_{Inv},"t","med"),
N(X_C,"Cl_{Inv}",M_{Inv},"av","high")}
NT={N("Temp")}
P={p_1,...,p_{10}}

p_1= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord},M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord},M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord},M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{Inv}",M3,"t","med"),
N8$\xrightarrow{t,med}$N5, N9(C6,"Cl_{Inv}",M3,"av","high"),N9$\xrightarrow{av,high}$N7,N("Temp")})

p_2= (L={AX}; K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord},M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord},M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord},M2,M3,"av","high"),
N7(S3,"S_{coord}",M3,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{Inv}",M3,"t","med"),
N8$\xrightarrow{t,med}$N5, N9(C6,"Cl_{Inv}",M3,"av","high"), N9$\xrightarrow{av,high}$N7, N("Temp")})

p_3= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord},M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord},M2,M3,"t","med"),
N5(S2,"S_{coord}",M3,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{Inv}",M3,"t","med"),
N8$\xrightarrow{t,med}$N5, N9(C6,"Cl_{Inv}",M3,"av","high"),N9$\xrightarrow{av,high}$N7, N("Temp")})

p_4= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord},M2,M3,"t","med"),
N5(S2,"S_{coord}",M3,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M4,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{Inv}",M3,"t","med"),
N8$\xrightarrow{t,med}$N5, N9(C6,"Cl_{Inv}",M3,"av","high"), N9$\xrightarrow{av,high}$N7,
N10(C7,"Cl_{coord},M2,M4,"t","med"), N10$\xrightarrow{t,med}$N5,
N11(C8,"Cl_{coord},M2,M4,"av","high"), N11$\xrightarrow{av,high}$N7,
N12(C9,"Cl_{Inv}",M4,"t","med"), N12$\xrightarrow{t,med}$N5, N13(C10,"Cl_{Inv}",M4,"av","high"),
N13$\xrightarrow{av,high}$N7, N("Temp")})

p_5= (L={N1(S1,"S_{cont}",M1,"t","high"), N("Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord}",M2,"t","high"), N2$\xrightarrow{t,high}$N1, N3(C2,"Cl_{coord}",M2,M3,"t","med"),
N4(S2,"S_{coord}",M2,"t","med"), N3$\xrightarrow{t,med}$N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M2,"av","high"), N5$\xrightarrow{av,high}$N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N4, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N6})

p_6= (L={N1(S1,"S_{cont}",M1,"t","high"), N(Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord},M2,"t","high"), N2$\xrightarrow{t,high}$N1, N3(C2,"Cl_{coord},M2,M3,"t","med"),
N4(S2,"S_{coord}",M2,"t","med"), N3$\xrightarrow{t,med}$N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M3,"av","high"), N5$\xrightarrow{av,high}$N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N4, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N6})

p_7= (L={N1(S1,"S_{cont}",M1,"t","high"), N("Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord},M2,"t","high"), N2$\xrightarrow{t,high}$N1, N3(C2,"Cl_{coord},M2,M3,"t","med"),
N4(S2,"S_{coord}",M3,"t","med"), N3$\xrightarrow{t,med}$N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M2,"av","high"), N5$\xrightarrow{av,high}$N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N4, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N6})

p_8= (L={N1(S1,"S_{cont}",M1,"t","high"), N("Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord},M2,"t","high"), N2$\xrightarrow{t,high}$N1, N3(C2,"Cl_{coord}",M2,M3,"t","med"),
N4(S2,"S_{coord}",M3,"t","med"), N3$\xrightarrow{t,med}$N4,N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M4,"av","high"), N5$\xrightarrow{av,high}$N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N4, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N6,
N9(C6,"Cl_{coord}",M2,M4,"t","med"), N9$\xrightarrow{t,med}$N4,
N10(C7,"Cl_{coord},M2,M4,"av","high"), N10$\xrightarrow{av,high}$N6,
N11(C4,"Cl_{Inv}",M3,"t","med"), N11$\xrightarrow{t,med}$N4, N12(C5,"Cl_{Inv}",M3,"av","high"),
N12$\xrightarrow{av,high}$N6})

p_9= (L={N1(C1,"Cl_{coord},M1,M2,"t","med"), N2(S1,"S_{coord}",M3,"t","med"),
N3(C3,"Cl_{coord}",M1,M2,"av","high"), N4(S2,"S_{coord}",M4,"av","high"), N1$\xrightarrow{t,med}$N2,
N3$\xrightarrow{av,high}$N4,N("Temp")};
K={N1(C1,"Cl_{coord},M1,M2,"t","med"), N2(S1,"S_{coord}",M3,"t","med"),
N3(C3,"Cl_{coord},M1,M2,"av","high"), N4(S2,"S_{coord}",M4,"av","high"), N1$\xrightarrow{t,med}$N2,
N3$\xrightarrow{av,high}$N4, N("Temp)};
R={N5(C4,"Cl_{coord},M1,M5,"av","high"),N5$\xrightarrow{av,high}$N4,
N6(C5,"Cl_{coord},M1,M5,"t","med"), N5$\xrightarrow{av,high}$N2, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N2, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N4})

p_{10}= (L={N("Temp")};K={ };R={ })

FIG. 3.18 – Les productions de la grammaire $GG_{R,e}$

La production p_1 permet de générer une configuration minimale[35] de l'architecture comprenant : 1) un client du contrôleur (nœud N1) pour consommer les rapports envoyés par le premier coordinateur, 2) le serveur (de QdS *t,high*) gérant les connexions entre le contrôleur et ses coordinateurs (nœud N2), 3) le client permettant au coordinateur de produire ses rapports vers le contrôleur (nœud N3), 4) les deux clients du coordinateur qui consomment les messages du premier investigateur (nœuds N4 et N6), 5) les serveurs en charge des connexions entre l'investigateur et le coordinateur (nœuds N5 et N7), et 6) les clients de l'investigateurs qui produisent les messages relatifs aux observations et aux rapports (respectivement nœuds N9 et N8).

La production p_1 considère le déploiement des serveurs, relatifs aux communications entre les investigateurs et le coordinateur, sur la machine du coordinateur (i.e. M2). Les productions p_2, p_3 et p_4 exposent les trois autres choix de déploiement.

Les productions p_5, p_6, p_7 et p_8 permettent de générer un coordinateur supplémentaire ainsi que son premier investigateur. A chacune de ces productions correspond une des quatre possibilités de déploiement consistantes.

La production p_9 permet de générer, pour un coordinateur donné, un investigateur supplémentaire en produisant les clients producteurs de messages de cet investigateur (nœuds N7 et N8) et les clients

[35] L'application comprend, au minimum, un contrôleur, un coordinateur et un investigateur

du côté coordinateur qui consomment ces messages (nœuds N5 et N6) et en raccordant ces clients aux serveurs correspondants (nœuds N2 et N4).

La production p_{10} termine le processus de génération en consommant le non terminal *N("Temp")*.

L'instance de l'architecture décrite dans la figure 3.17 peut être générée par la grammaire $GG_{R,e}$ selon le chemin d'application des productions suivant : 1) p_1 permettant de générer le client C1, le serveur S1, le client C3, les clients C4 et C7, les serveurs S2 et S3, et les clients C15 et C16 ; 2) p_9 qui génère, pour l'investigateur de la machine M5, les clients C17 et C18 et, pour le coordinateur, les clients C5 et C8 ; 3) p_9 permettant de générer, pour l'investigateur de la machine M6, les clients C19 et C20 et, pour le coordinateur, les clients C6 et C9 ; 4) p_4 permettant de générer les clients et les serveurs correspondant au coordinateur de la machine M3 et aux investigateurs des machines M7 et M8. L'application de cette production génère : les clients C2 et C10 respectivement consommateur et producteur des messages produits par le coordinateur de la machine M3 et consommés par le contrôleur, les clients C11, C12, C13 et C14 consommateurs des messages envoyés par les investigateurs des machines M7 et M8, les serveurs S4 et S5 en charge des communications entre les investigateurs et le coordinateur, et les clients C21/C22 et C23/C24 correspondant respectivement aux investigateurs des machines M7 et M8 ; 5) p_{10} pour terminer la génération de l'instance.

3.5.2 Description au niveau réseau et en phase d'action

Dans cette partie nous considérons le style architectural correspondant au niveau réseau et à la phase d'action. Cette phase correspond toujours au passage d'une section à une configuration α suite à la découverte par un investigateur d'une situation critique.

La réorganisation des communications entre le coordinateur et ses investigateurs implique une prise en compte de l'aspect critique des communications entre l'investigateur α et le coordinateur. Ainsi, les échanges correspondant aux messages descriptifs audio et vidéo exigent le niveau de QdS le plus haut. Les échanges correspondant aux messages textuels rapportés par l'investigateur α au coordinateur sont importants mais restent d'importance moindre que les messages descriptifs[36]. À ces messages rapportés nous attribuons une qualité de service moyenne.

Au vu du caractère critique des messages envoyés par l'investigateur α vers le coordinateur, Les serveurs en charge des communications relatives à ces messages seront obligatoirement déployés sur la machine du coordinateur

En ce qui concerne les données descriptives audio et vidéo envoyées par l'investigateur α aux autres investigateurs, elles sont considérées de moindre importance et nous leur attribuons l'exigence de QdS la plus faible. La même exigence de QdS est attribuée aux liens entre les investigateurs β et le coordinateur. Les deux serveurs relatifs à ces deux types de communications peuvent être déployés sur des machines hébergeant des investigateurs β.

Concernant les branches correspondant aux sections en phase d'exploration, la structure de connexion ainsi que les exigence en QdS restent les mêmes que celles décrites précédemment pour le style architectural de l'application en phase d'exploration.

Les clients appartenant au contrôleur ainsi que le serveur des messages qui leurs sont destinés sont décrits par le même type de nœud que dans la phase d'exploration. La structure interne des coordinateurs qui sont en phase d'exploration ainsi que leurs investigateurs ne change pas non plus par rapport à la phase où toute l'application est en phase d'exploration.

En ce qui concerne les coordinateurs en phase d'action, nous considérons les notations suivantes :

- Le client permettant d'envoyer les messages de rapport vers le coordinateur gardera la même description que dans la phase d'exploration.
- Deux nouveaux clients sont introduits. Ils correspondent aux clients qui consomment les observations et les rapports envoyés par les clients de l'investigateur α. Ces deux clients sont décrits par le nœud *N("C1","Cl_{coord}","M1","M2","av","high")* correspondant au consommateur des observations et dont le niveau de QdS est maximal, et par le nœud N("C1","Cl_{coord}","M1","M2","t","med") correspondant au consommateur des rapports qui possèdent une QdS moyenne.

[36] étant donné que le coordinateur dispose aussi de ses propres moyens d'analyse.

– Les clients permettant de consommer les rapports des investigateurs β sont décrits par les nœuds *N("C1","Cl_{coord}","M1","M2","t","low")* et possèdent donc la QdS la plus faible.

L'investigateur α comprend les composants suivants :

– Les clients permettant de produire les messages d'observation et de rapport à destination du coordinateur. Ces clients sont respectivement décrits par les nœuds *N("C1","Cl_{Inv}","M1","av","high")* et *N("C1","Cl_{Inv}","M1","t","med")*.

– Et le client produisant les messages d'observation pour les investigateurs β décrit par un nœud *N("C1","Cl_{Inv}","M1","av","low")*.

À chaque coordinateur en phase d'action correspond :

– Deux serveurs assurant les communications entre le coordinateur et l'investigateur α. Ces deux serveurs correspondent aux types de messages *t,med* et *av,high* et sont décrits, respectivement, par des nœuds de type *N("S","S_{coord}","M1","t","med")* et *N("S","S_{coord}","M1","av","high")*.

– Un serveur pour les communications entre les investigateurs β et le coordinateur décrit par un nœud *N("S","S_{coord}","M1","t","low")*.

– Un serveur pour les communications entre l'investigateur α et les investigateurs β décrit par un nœud *N("S","S_{Inv}","M1","av","low")*.

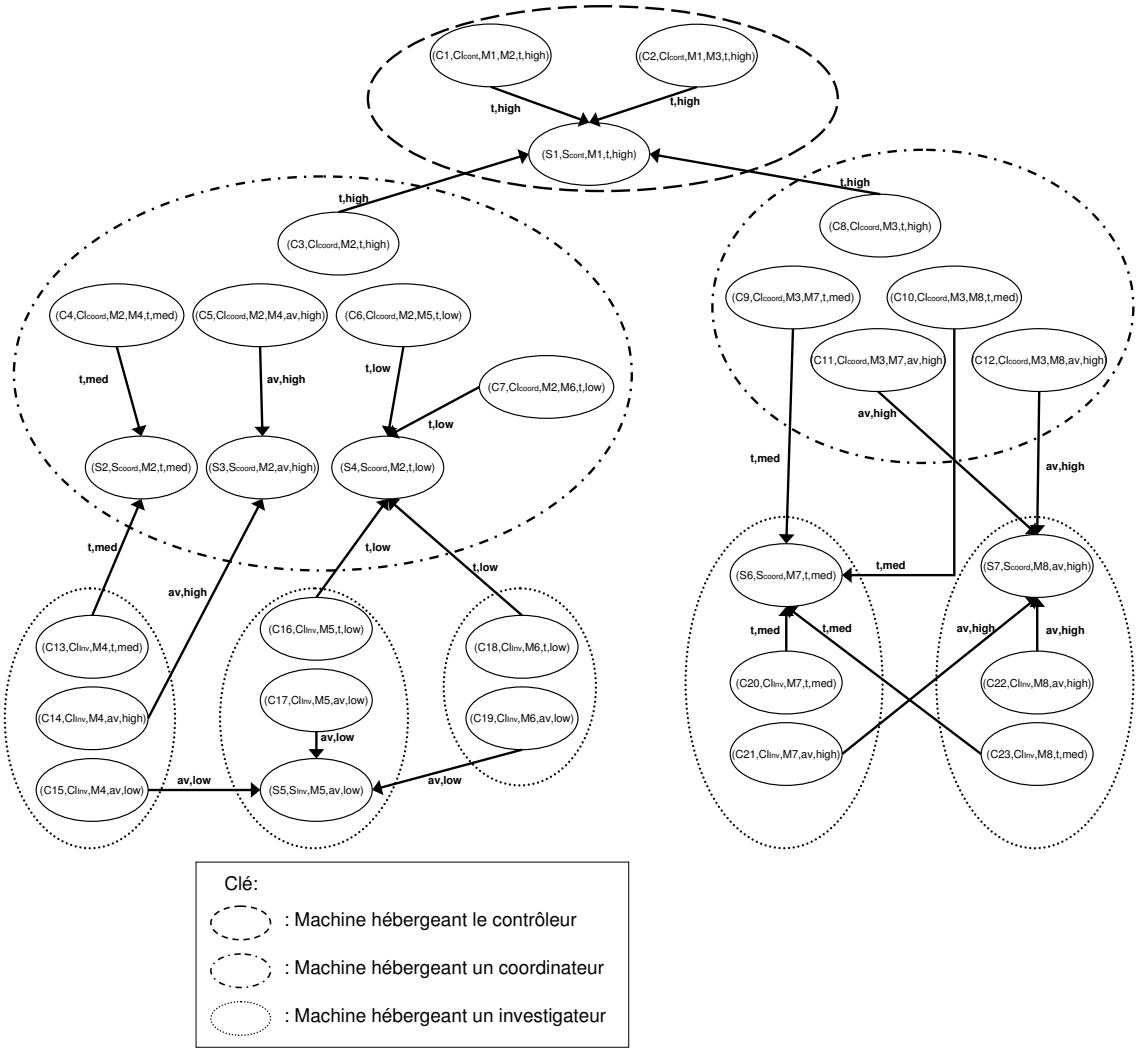


FIG. 3.19 – Exemple de l'architecture dans la phase d'action

La figure 3.19 donne un exemple d'une instance de l'architecture appartenant au niveau réseau en phase d'action. Le coordinateur de la machine M2 est en phase d'action alors que celui de la machine M3 est en phase d'exploration. La machine M4 héberge l'investigateur α et donc les clients permettant de produire les messages pour le coordinateur et les investigateurs β. Les machines $M5$ et $M6$ correspondent à deux investigateurs β. Les serveurs qui traitent les communications entre le coordinateur et l'investigateur α (serveurs S2 et S3) ainsi que le serveur qui gère les communications entre les investigateurs β et le coordinateur (serveur S4) sont déployés sur la machine du coordinateur. Le serveur relatif aux communications entre l'investigateur α et les investigateurs β est déployé sur la machine $M5$ correspondant à un investigateur β.

Le style architectural de l'application au niveau réseau et en phase d'action est décrit par la grammaire $GG_{R,a}$=(AX,NT,T,P) avec :
T={N(X_C,"Cl_{cont}",M_{cont},M_{cont},"t","high"), N(X_S,"S_{cont}",M_{cont},"t","high"),
N(X_C,"Cl_{coord}",M_{coord},"t","high"), N(X_C,"Cl_{coord}",M_{coord},M_{Inv},"t","med"),
N(X_C,"Cl_{coord}",M_{coord},M_{Inv},"av","high"), N(X_C,"Cl_{coord}",M_{coord},M_{Inv},"t","low"),
N(X_S,"S_{coord}",M,"t","med"), N(X_S,"S_{coord}",M,"av","high"),
N(X_S,"S_{coord}",M,"t","low"), N(X_S,"S_{coord}",M,"av","low"),
N(X_C,"Cl_{Inv}",M_{Inv},"av","low"), N(X_C,"Cl_{Inv}",M_{Inv},"t","low"),
N(X_C,"Cl_{Inv}",M_{Inv},"t","med"), N(X_C,"Cl_{Inv}",M_{Inv},"av","high")}
NT={N("Temp")}, et
P={p_1,...,p_{15}}

Les productions de cette grammaire sont données dans la figure 3.20.

p_1= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M2,"t","low"), N8$\xrightarrow{t,low}$N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10$\xrightarrow{t,med}$N5, N11(C7,"Cl_{Inv}",M3,"av","high"), N11$\xrightarrow{av,high}$N7,
N12(C8,"Cl_{Inv}",M3,"av","low"), N13(S5,"S_{Inv}",M2,"av","low"), N12$\xrightarrow{t,low}$N13,
N14(C9,"Cl_{Inv}",M4,"t","low"),N14$\xrightarrow{t,low}$N9, N15(C10,"Cl_{Inv}",M4,"av","low"),
N15$\xrightarrow{av,low}$N13, N("Temp")})

p_2= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M2,"t","low"), N8$\xrightarrow{t,low}$N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10$\xrightarrow{t,med}$N5, N11(C7,"Cl_{Inv}",M3,"av","high"), N11$\xrightarrow{av,high}$N7,
N12(C8,"Cl_{Inv}",M3,"av","low"), N13(S5,"S_{Inv}",M4,"av","low"), N12$\xrightarrow{t,low}$N13,
N14(C9,"Cl_{Inv}",M4,"t","low"),N14$\xrightarrow{t,low}$N9,
N15(C10,"Cl_{Inv}",M4,"av","low"),N15$\xrightarrow{av,low}$N13, N("Temp")})

p_3= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M4,"t","low"), N8$\xrightarrow{t,low}$N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10$\xrightarrow{t,med}$N5, N11(C7,"Cl_{Inv}",M3,"av","high"),N11$\xrightarrow{av,high}$N7,
N12(C8,"Cl_{Inv}",M3,"av","low"), N13(S5,"S_{Inv}",M2,"av","low"), N12$\xrightarrow{t,low}$N13,
N14(C9,"Cl_{Inv}",M4,"t","low"),N14$\xrightarrow{t,low}$N9,
N15(C10,"Cl_{Inv}",M4,"av","low"),N15$\xrightarrow{av,low}$N13, N("Temp")})

p_4= (L={AX};K={ };
R={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"), N1$\xrightarrow{t,high}$N2,
N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M4,"t","low"), N8$\xrightarrow{t,low}$N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10$\xrightarrow{t,med}$N5, N11(C7,"Cl_{Inv}",M3,"av","high"), N11$\xrightarrow{av,high}$N7,
N12(C8,"Cl_{Inv}",M3,"av","low"), N13(S5,"S_{Inv}",M5,"av","low"), N12$\xrightarrow{t,low}$N13,
N14(C9,"Cl_{Inv}",M4,"t","low"), N14$\xrightarrow{t,low}$N9, N15(C10,"Cl_{Inv}",M4,"av","low"),
N15$\xrightarrow{av,low}$N13, N16(C9,"Cl_{Inv}",M4,"t","low"), N16$\xrightarrow{t,low}$N9,
N17(C10,"Cl_{Inv}",M4,"av","low"), N17$\xrightarrow{av,low}$N13, N("Temp")})

p_5= (L={N1(C1,"Cl_{cont}",M1,M2,"t","high"),N2(S1,"S_{cont}",M1,"t","high"),N1$\xrightarrow{t,high}$N2,N("Temp")};
K={N1(C1,"Cl_{cont}",M1,M2,"t","high"),N2(S1,"S_{cont}",M1,"t","high"),N1$\xrightarrow{t,high}$N2,N("Temp")};
R={N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7,
N8(C5,"Cl_{coord}",M2,M4,"t","low"), N9(S4,"S_{coord}",M2,"t","low"), N8$\xrightarrow{t,low}$N9,
N10(C6,"Cl_{Inv}",M3,"t","med"), N10$\xrightarrow{t,med}$N5, N11(C7,"Cl_{Inv},M3,"av","high"),
N11$\xrightarrow{av,high}$N7, N12(C8,"Cl_{Inv}",M3,"av","low"),N13(S5,"S_{Inv}",M2,"av","low"),
N12$\xrightarrow{t,low}$N13, N14(C9,"Cl_{Inv},M4,"t","low"), N14$\xrightarrow{t,low}$N9,
N15(C10,"Cl_{Inv}",M4,"av","low"), N15$\xrightarrow{av,low}$N13,

p_6= (L={N1(C1,"Cl_{cont}",M1,M2,"t","high"),N2(S1,"S_{cont}",M1,"t","high"),N1$\xrightarrow{t,high}$N2,N("Temp")};
K={N1(C1,"Cl_{cont}",M1,M2,"t","high"),N2(S1,"S_{cont}",M1,"t","high"),N1$\xrightarrow{t,high}$N2,N("Temp")};
R={N3(C2,"Cl_{coord}",M2,"t","high"), N3$\xrightarrow{t,high}$N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4$\xrightarrow{t,med}$N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6$\xrightarrow{av,high}$N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M2,"t","low"), N8$\xrightarrow{t,low}$N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10$\xrightarrow{t,med}$N5, N11(C7,"Cl_{Inv}",M3,"av","high"), N11$\xrightarrow{av,high}$N7,
N12(C8,"Cl_{Inv}",M3,"av","low"),N13(S5,"S_{Inv}",M4,"av","low"), N12$\xrightarrow{t,low}$N13,
N14(C9,"Cl_{Inv}",M4,"t","low"),N14$\xrightarrow{t,low}$N9,
N15(C10,"Cl_{Inv}",M4,"av","low"),N15$\xrightarrow{av,low}$N13, N("Temp")})

p_7= (L={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"),
N1 $\xrightarrow{t,high}$ N2, N("Temp")};
K={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"),
N1 $\xrightarrow{t,high}$ N2, N("Temp")};
R={N3(C2,"Cl_{coord}",M2,"t","high"), N3 $\xrightarrow{t,high}$ N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4 $\xrightarrow{t,med}$ N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6 $\xrightarrow{av,high}$ N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M4,"t","low"), N8 $\xrightarrow{t,low}$ N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10 $\xrightarrow{t,med}$ N5, N11(C7,"Cl_{Inv}",M3,"av","high"), N11 $\xrightarrow{av,high}$ N7,
N12(C8,"Cl_{Inv}",M3,"av","low"), N13(S5,"S_{Inv}",M2,"av","low"), N12 $\xrightarrow{t,low}$ N13,
N14(C9,"Cl_{Inv}",M4,"t","low"), N14 $\xrightarrow{t,low}$ N9,
N15(C10,"Cl_{Inv}",M4,"av","low"), N15 $\xrightarrow{av,low}$ N13, N("Temp")})

p_8= (L={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"),
N1 $\xrightarrow{t,high}$ N2, N("Temp")};
K={N1(C1,"Cl_{cont}",M1,M2,"t","high"), N2(S1,"S_{cont}",M1,"t","high"),
N1 $\xrightarrow{t,high}$ N2, N("Temp")};
R={N3(C2,"Cl_{coord}",M2,"t","high"), N3 $\xrightarrow{t,high}$ N2, N4(C3,"Cl_{coord}",M2,M3,"t","med"),
N5(S2,"S_{coord}",M2,"t","med"), N4 $\xrightarrow{t,med}$ N5, N6(C4,"Cl_{coord}",M2,M3,"av","high"),
N7(S3,"S_{coord}",M2,"av","high"), N6 $\xrightarrow{av,high}$ N7, N8(C5,"Cl_{coord}",M2,M4,"t","low"),
N9(S4,"S_{coord}",M4,"t","low"), N8 $\xrightarrow{t,low}$ N9, N10(C6,"Cl_{Inv}",M3,"t","med"),
N10 $\xrightarrow{t,med}$ N5, N11(C7,"Cl_{Inv}",M3,"av","high"), N11 $\xrightarrow{av,high}$ N7,
N12(C8,"Cl_{Inv}",M3,"av","low"), N13(S5,"S_{Inv}",M5,"av","low"), N12 $\xrightarrow{t,low}$ N13,
N14(C9,"Cl_{Inv}",M4,"t","low"), N14 $\xrightarrow{t,low}$ N9, N15(C10,"Cl_{Inv}",M4,"av","low"),
N15 $\xrightarrow{av,low}$ N13, N16(C9,"Cl_{Inv}",M4,"t","low"),N16 $\xrightarrow{t,low}$ N9,
N17(C10,"Cl_{Inv}",M4,"av","low"),N17 $\xrightarrow{av,low}$ N13, N("Temp")})

p_9= (L={N1(S1,"S_{cont}",M1,"t","high"),N("Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord}",M2,"t","high"), N2 $\xrightarrow{t,high}$ N1, N3(C2,"Cl_{coord}",M2,M3,"t","med"),
N4(S2,"S_{coord}",M2,"t","med"), N3 $\xrightarrow{t,med}$ N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M2,"av","high"), N5 $\xrightarrow{av,high}$ N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7 $\xrightarrow{t,med}$ N4, N8(C5,"Cl_{Inv}",M3,"av","high"),N8 $\xrightarrow{av,high}$ N6})

p_{10}= (L={N1(S1,"S_{cont}",M1,"t","high"),N(Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord}",M2,"t","high"), N2 $\xrightarrow{t,high}$ N1, N3(C2,"Cl_{coord}",M2,M3,"t","med"),
N4(S2,"S_{coord}",M2,"t","med"), N3 $\xrightarrow{t,med}$ N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M3,"av","high"), N5 $\xrightarrow{av,high}$ N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7 $\xrightarrow{t,med}$ N4, N8(C5,"Cl_{Inv}",M3,"av","high"),N8 $\xrightarrow{av,high}$ N6})

p_{11}= (L={N1(S1,"S_{cont}",M1,"t","high"),N("Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord}",M2,"t","high"), N2 $\xrightarrow{t,high}$ N1, N3(C2,"Cl_{coord}",M2,M3,"t","med"),
N4(S2,"S_{coord}",M3,"t","med"), N3 $\xrightarrow{t,med}$ N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M2,"av","high"), N5 $\xrightarrow{av,high}$ N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7 $\xrightarrow{t,med}$ N4, N8(C5,"Cl_{Inv}",M3,"av","high"), N8 $\xrightarrow{av,high}$ N6})

p_{12}= (L={N1(S1,"S_{cont}",M1,"t","high"),N("Temp")};
K={N2(S1,"S_{cont}",M1,"t","high"), N("Temp")};
R={N2(C1,"Cl_{coord}",M2,"t","high"), N2$\xrightarrow{t,high}$N1, N3(C2,"Cl_{coord}",M2,M3,"t","med"),
N4(S2,"S_{coord}",M3,"t","med"), N3$\xrightarrow{t,med}$N4, N5(C3,"Cl_{coord}",M2,M3,"av","high"),
N6(S3,"S_{coord}",M4,"av","high"), N5$\xrightarrow{av,high}$N6, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N4, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N6,
N9(C6,"Cl_{coord}",M2,M4,"t","med"), N9$\xrightarrow{t,med}$N4,
N10(C7,"Cl_{coord}",M2,M4,"av","high"), N10$\xrightarrow{av,high}$N6,
N11(C4,"Cl_{Inv}",M3,"t","med"), N11$\xrightarrow{t,med}$N4, N12(C5,"Cl_{Inv}",M3,"av","high"),
N12$\xrightarrow{av,high}$N6}),

p_{13}= (L={N1(C1,"Cl_{coord}",M1,M2,"av","high"), N2(S1,"S_{coord}",M1,"av","high"),
N1$\xrightarrow{av,high}$N2, N3(C2,"Cl_{coord}",M1,M3,"t","low"), N4(S2,"S_{coord}",M4,"t","low"),
N3$\xrightarrow{t,low}$N4, N5(C3,"C_{Inv}",M5,"av","high"),N5$\xrightarrow{av,high}$N2,
N6(C4,"C_{Inv}",M6,"av","low"), N7(S3,"S_{Inv}",M6,"av","low"),
N6$\xrightarrow{av,low}$N7, N("Temp")};
K={N1(C1,"Cl_{coord}",M1,M2,"av","high"), N2(S1,"S_{coord}",M1,"av","high"),
N1$\xrightarrow{av,high}$N2, N3(C2,"Cl_{coord}",M1,M3,"t","low"), N4(S2,"S_{coord}",M4,"t","low"),
N3$\xrightarrow{t,low}$N4, N5(C3,"C_{Inv}",M5,"av","high"),N5$\xrightarrow{av,high}$N2,
N6(C4,"C_{Inv}",M6,"av","low"), N7(S3,"S_{Inv}",M6,"av","low"), N6$\xrightarrow{av,low}$N7,
N("Temp")};
R={N8(C6,"C_{Inv}",M7,"t","low"), N8$\xrightarrow{t,low}$N4, N9(C7,"C_{Inv}",M7,"av","low"),
N9$\xrightarrow{av,low}$N7, N10(C8,"Cl_{coord}",M1,M7,"t","low"), N10$\xrightarrow{t,low}$N4,})

p_{14}= (L={N1(C1,"Cl_{coord}",M1,M2,"t","med"), N2(S1,"S_{coord}",M3,"t","med"),
N3(C3,"Cl_{coord}",M1,M2,"av","high"), N4(S2,"S_{coord}",M4,"av","high"),
N1$\xrightarrow{t,med}$N2, N3$\xrightarrow{av,high}$N4, N("Temp")};
K={N1(C1,"Cl_{coord},M1,M2,"t","med"), N2(S1,"S_{coord}",M3,"t","med"),
N3(C3,"Cl_{coord}",M1,M2,"av","high"), N4(S2,"S_{coord}",M4,"av","high"),
N1$\xrightarrow{t,med}$N2, N3$\xrightarrow{av,high}$N4, N("Temp)};
R={N5(C4,"Cl_{coord}",M1,M5,"av","high"), N5$\xrightarrow{av,high}$N4,
N6(C5,"Cl_{coord}",M1,M5,"t","med"), N6$\xrightarrow{av,high}$N2, N7(C4,"Cl_{Inv}",M3,"t","med"),
N7$\xrightarrow{t,med}$N2, N8(C5,"Cl_{Inv}",M3,"av","high"), N8$\xrightarrow{av,high}$N4})

p_{15}= (L={N("Temp")};K={ };R={ })

FIG. 3.20 – Productions de la grammaire $GG_{R,a}$

La production p_1 permet de générer une configuration minimale[37] de l'architecture comprenant : 1) le client du contrôleur (nœud N1) pour consommer les rapports envoyés par le premier coordinateur, 2) le serveur (de QdS *t,high*) gérant les connexions entre le contrôleur et ses coordinateurs (nœud N2), 3) le client permettant au coordinateur de produire ses rapports vers le contrôleur (nœud N3), 4) les deux clients du coordinateur qui consomment les messages de l'investigateur α (nœuds N4 et N6), 5) les serveurs en charge des connexions entre l'investigateur α et le coordinateur (nœuds N5 et N7), 6) le client du coordinateur qui consomme les messages produits par le premier investigateur β (nœud N8), 7) le serveur reliant les investigateurs β au coordinateur (nœud N9), 8) les clients de l'investigateur α

[37] L'application en phase d'action comprend, au minimum, un contrôleur, un coordinateur, un investigateur α et un investigateur β.

produisant des messages pour le coordinateur (nœuds N10 et N11) et pour les investigateurs β (nœud N12), 9) le serveur en charge des communications entre l'investigateur α et les investigateurs β (nœud N13), 10) le client du premier investigateur β qui consomme les messages de l'investigateur α (nœud N15), et 11) le client du premier investigateur β qui produit les messages pour le coordinateur (nœud N14).

La production p_1 correspond au choix de déployer les serveurs, entre l'investigateur α et les autres investigateurs, et entre les investigateurs β et le coordinateur, sur la machine du coordinateur. Les productions p_2, p_3 et p_4 permettent de considérer les trois autres choix de déploiement pour ces serveurs.

Les productions p_5, p_6, p_7 et p_8 permettent d'introduire les clients et serveurs correspondant à un nouveau coordinateur en phase d'action, à son investigateur α et à un premier investigateur β. Chacune de ces productions correspond à un choix de déploiement des serveurs entre l'investigateur α et les investigateurs β et entre les investigateurs β et le coordinateur.

Les productions p_9, p_{10}, p_{11} et p_{12} correspondent à l'introduction de coordinateurs en phase d'exploration en prenant en compte les différentes options consistantes pour le déploiement des serveurs entre les investigateurs et le coordinateur.

La production p_{13} permet de générer les clients producteurs de messages appartenant à un investigateur supplémentaire pour un coordinateur en phase d'exploration. p_{14} génère le client consommateur des messages de l'investigateur α et le client producteur de messages pour le coordinateur appartenant à un investigateur β supplémentaire pour un coordinateur en phase d'action.

La production p_{15} termine le processus de génération.

L'instance décrite dans la figure 3.19 peut être générée par la grammaire $GG_{R,a}$ selon le chemin d'application des productions suivant : 1) p_2 permettant de générer le client *C1*, le serveur *S1*, le client *C3*, les clients *C4*, *C5* et *C6*, les serveurs *S2*, *S3* et *S4*, les clients *C13*, *C14* et *C15* de l'investigateur α déployés sur la machine *M4*, le serveur *S5*, et les clients *C16* et *C17* ; 2) la production p_{13} pour générer le client *C7* et les clients *C18* et *C19* ; 3) p_8 permettant de générer le client *C8*, les clients *C9*, *C10*, *C11* et *C12*, les serveurs *S6* et *S7*, et les clients *C20*, *C21*, *C22* et *C23* ; et finalement 3) p_{15} pour terminer.

3.6 Gestion de l'architecture dynamique

Les actions de reconfiguration à un niveau d'abstraction donné ont forcément une incidence sur l'architecture considérée aux niveaux d'abstraction inférieurs. Cependant, chaque niveau d'abstraction peut posséder des actions de reconfiguration propres et des événements de reconfiguration qui n'influent pas sur l'architecture et le protocole de reconfiguration des niveaux d'abstraction de plus haut niveau.

Nous nous situons, donc, dans un contexte où les événements de reconfiguration considérés pour un protocole de reconfiguration à un niveau d'abstraction donné seront considérés par les protocoles de reconfiguration relatifs aux niveaux les plus bas. Les protocoles de reconfiguration de plus haut niveau peuvent, par contre, ignorer les événements de reconfiguration des niveaux inférieurs.

Dans cette section, nous définissons les différents protocoles de reconfiguration de l'architecture. Nous identifions, pour chaque niveau d'abstraction, les événements qui nécessitent une transformation de l'architecture ainsi que les règles de reconfiguration correspondant à chaque événement.

Pour notre cas de figure, le niveau réseau constitue le niveau le plus bas relatif aux composants concrets de l'application. Les deux autres niveaux concernent des vues abstraites de l'architecture. Le protocole de reconfiguration de plus bas niveau sera donc celui qui va agir réellement sur la structure exécutable de l'application tandis que les deux autres protocoles permettent de définir des actions de reconfiguration de plus haut niveau et de garder une vision consistante de l'architecture courante (i.e. les instances décrites sur les trois niveaux doivent être cohérentes).

À un niveau d'abstraction donné, l'application peut passer d'une phase d'exploration vers une phase d'action et vice versa. Ce passage d'une phase vers une autre est perçu comme une action d'adaptation au contexte d'exécution. Nous définissons, dans ce qui suit, un protocole de reconfiguration pour les deux niveaux supérieurs.

3.6.1 Le niveau application

Nous supposons, pour le niveau application, que le nombre des participants à une mission est fixé et nous excluons, par conséquent, l'intégration de nouveaux participants pendant le déroulement de la mission. Par contre, vu que la mission s'effectue sur un terrain accidenté et hostile, les investigateurs peuvent être détruits, subir des pannes ou épuiser leur réserves d'énergie. Les coordinateurs et le contrôleur mieux protégés et disposant d'accès permanent à l'énergie, sont considérés comme robustes et leur panne n'est pas prise en compte par le protocole de reconfiguration. Nous considérons des pannes permanentes et supposons qu'un élément en panne restera dans cet état pendant toute la durée de la mission.

Outre la panne des investigateurs, nous considérons aussi, les actions de reconfiguration qui doivent être réalisées suite à la découverte d'une situation critique ou suite à la fin de la situation critique.

Dans le premier cas et lorsqu'un investigateur découvre une situation critique, l'architecture est reconfigurée de telle manière que cet investigateur devient l'investigateur α alors que les autres investigateurs de sa section possèdent le niveau β. Le coordinateur de la section passe aussi du mode exploration vers le mode action.

Dans le deuxième cas impliquant la fin de la situation critique suite à son traitement par l'application ou à un événement extérieur, la section concernée repasse en phase d'exploration. Le coordinateur ainsi que tous les investigateurs (l'investigateur α compris) reprennent une exécution correspondant à cette phase.

Nous considérons, à ce niveau d'abstraction, les événements et les actions de reconfiguration suivants :

- L'événement correspondant à la perte ou à la panne d'un investigateur alors que sa section est en phase d'exploration est spécifié par "$Exclude_{\varphi 1}(m)$" où le paramètre m correspond à la machine qui héberge l'investigateur en panne. Cet événement implique l'exclusion de l'architecture de ce composant. Le coordinateur ne le prend plus en compte dans la répartition des tâches et les tâches qu'il devait réaliser sont réaffectées aux autres investigateurs de sa section.
- L'événement correspondant à la perte d'un investigateur β (la section est forcément en phase d'action) est spécifié par "$Exclude_{\beta}(m)$" où m correspond à la machine qui héberge cet investigateur. Les actions de reconfiguration pour le traitement de cet événement restent les mêmes que dans le cas de l'exclusion d'un investigateur en phase d'exploration (non prise en compte de ce composant dans les affectations futures et réaffectation de ses tâches aux autres investigatreurs β).
- L'événement correspondant à la perte de l'investigateur α d'une section en phase d'action est plus critique que les deux précédents puisque le rôle d'investigateur α est déterminant dans le traitement de la situation critique. Pour cela, il faut désigner un autre investigateur qui va le remplacer pour jouer son rôle. L'événement est donc décrit par "$Exclude_{\alpha}$*(m1,m2)*" où *m1* et *m2* correspondent respectivement aux machines qui hébergent l'investigateur α en panne et l'investigateur β qui va le remplacer et jouer le rôle d'investigateur α.
- L'événement correspondant à la découverte d'une situation critique est spécifié par "*CriticalSituation(m)*" où m correspond à la machine sur laquelle est déployé l'investigateur qui découvre cette situation. Cet investigateur prend donc le rôle d'investigateur α et toute sa section passe à la phase d'action impliquant une reconfiguration de la structure de communication telle que cela a été décrit précédemment.
- L'événement correspondant à la fin de la situation critique est spécifié par "*EndCritical(m)*" où m correspond à la machine qui héberge le coordinateur qui dirige la section et qui décrète la fin de la situation critique. Les actions de reconfiguration impliquent le passage de toute la section (i.e. le coordinateur, l'investigateur α et les investigateurs β) en phase d'exploration.

Le protocole de reconfiguration relatif au niveau application est donc défini par le système PR_A. Les règles de reconfiguration sont définies dans la figure 3.21.

PR_A= {(T=$Exclude_{\varphi 1}$(m),
C= $R1_A$(X_{I1},M1,X_{coord},M2,X_{I2},M3),
I={(M1,m)}),
(T=$Exclude_{\beta}$(m),
C= $R2_A$(X_{I1},M1,X_{coord},M2),
I={(M1,m)}),
(T=$Exclude_{\alpha}$(m1,m2),

C= $R3_A(X_{I1}$,M1,X_{I2},M2,X_{coord},M3),
I={(M1,m1),(M2,m2)}),
(T=*CriticalSituation*(m),
C= [$R4a_A(X_{I1}$,M1,X_{coord},M2) ; $R4b_A(X_{I1}$,M1,X_{coord},M2,X_{I2},M3)*],
I={(M1,m)}),
(T=*EndCritical*(m),
C= $R5a_A(X_{I1}$,M1,X_{coord},M2) ; $R5b_A(X_{I1}$,M1,X_{coord},M2,X_{I2},M3)*],
I={(M2,m)})}

$R1_A$= (L= {N1(X_{I1},"Inv",M1,"O+R"), N2(X_{coord},"Coord","φ1",M2),
N3(X_{I2},"Inv",M3,"O+R"), N1$\xrightarrow{O+R}$N2, N3$\xrightarrow{O+R}$N2} ;
K= {N2(X_{coord},"Coord","φ1",M2), N3(X_{I2},"Inv",M3,"O+R"), N3$\xrightarrow{O+R}$N2} ;
R= { } ; N={ } ; C={ })

$R2_A$= (L= {N1(X_{I1},"Inv",M1,"R"), N2(X_{coord},"Coord","φ2",M2),
N3(X_{I2},"Inv",M3,"R"), N1$\xrightarrow{R}$N2, N3$\xrightarrow{R}$N2} ;
K= {N2(X_{coord},"Coord","φ2",M2), N3(X_{I2},"Inv",M3,"R"), N3$\xrightarrow{R}$N2} ;
R= { } ; N={ } ; C={ })

$R3_A$= (L= {N1(X_{I1},"Inv_α",M1,"O+R"), N2(X_{I2},"Inv",M2,"R"), N3(X_{I3},"Inv",M3,"R"),
N4(X_{coord},"Coord","φ2",M4), N1$\xrightarrow{O+R}$N4, N2$\xrightarrow{O}$N4, N3$\xrightarrow{O}$N4,} ;
K= {N4(X_{coord},"Coord","φ2",M4), N3(X_{I3},"Inv",M3), N3$\xrightarrow{O}$N4} ;
R= {N5(X_{I2},"Inv_α",M2), N5$\xrightarrow{O+R}$N4} ;
N={ } ; C={ic}), avec
ic=(N5,(X_I,"Inv",M5,"R"),"O"/"O",out,out)

$R4a_A$= (L= {N1(X_{I1},"Inv",M1,"O+R"), N2(X_{I2},"Inv",M2,"O+R"), N3(X_{coord},"Coord","φ1",M3),
N1$\xrightarrow{O+R}$N3, N2$\xrightarrow{O+R}$N3} ;
K= {N2(X_{I2},"Inv",M2,"O+R")} ;
R= {N4(X_{I1},"Inv_α",M1,"O+R"), N5(X_{coord},"Coord","φ2",M3), N4$\xrightarrow{O+R}$N5, N2$\xrightarrow{O+R}$N5} ;
N={ } ; C={ic}), avec
ic=(N5,(X_I,"Inv",M,"O+R"),"O+R"/"O+R",in,in)

$R4b_A$= (L= {N1(X_{I1},"Inv_α",M1,"O+R"), N2(X_{coord},"Coord","φ2",M2),
N3(X_{I2},"Inv",M3,"O+R"), N3$\xrightarrow{O+R}$N2} ;
K= {N1(X_{I1},"Inv_α",M1), N2(X_{coord},"Coord","φ2",M2)} ;
R= {N4(X_{I2},"Inv",M3,"R"), N1$\xrightarrow{O}$N4, N4$\xrightarrow{R}$N2} ;
N={ } ; C={ })

$R5a_A$= (L= {N1(X_{I1},"Inv_α",M1,"O+R"), N2(X_{coord},"Coord","φ2",M2), N1$\xrightarrow{O+R}$N2} ;
K= { } ;
R= {N3(X_{I1},"Inv",M1,"O+R"), N4(X_{coord},"Coord","φ1",M2), N3$\xrightarrow{O+R}$N4} ;
N={ } ; C={ic}), avec
ic=(N4,(X_I,"Inv",M,"R"),"R"/"R",in,in)

$R5b_A$= (L={N1(X_{I1},"Inv",M1,"O+R"), N2(X_{coord},"Coord","φ1",M2), N1$\xrightarrow{O+R}$N2,
N3(X_{I2},"Inv",M3,"R"), N3$\xrightarrow{R}$N2} ;
K= {N1(X_{I1},"Inv",M1,"O+R"), N2(X_{coord},"Coord","φ1",M2), N1$\xrightarrow{O+R}$N2} ;
R= {N4(X_{I2},"Inv",M3,"O+R"), N4$\xrightarrow{O+R}$N2} ;
N={ } ; C={ })

FIG. 3.21 – Règles de transformation pour le protocole de reconfiguration au niveau application

La règle de transformation $R1_A$ correspondant à la reconfiguration de l'architecture suite à la panne d'un investigateur, a pour conséquence la suppression du nœud correspondant à ce composant (nœud N1). Pour que cette règle soit applicable, nous exigeons que le coordinateur (nœud N2) qui coordonne cet investigateur possède au moins un autre investigateur (nœud N3). Ceci permet de ne pas produire des descriptions comprenant des coordinateurs isolés ce qui est interdit par la spécification.

La règle $R2_A$ correspondant à la panne d'un investigateur β permet de supprimer le nœud correspondant (nœud N1) de telle manière qu'il ne soit plus pris en compte par l'application. L'application de cette règle exige l'existence d'au moins un autre investigateur β (nœud N3) puisqu'une section en action en contient au moins un.

La règle $R3_A$ permet de traiter le cas de la panne d'un investigateur α (nœud N1). L'application de cette règle exige l'existence d'au moins deux autres investigateurs (i.e. le nœud N2 et N3 représentant deux investigateurs β) appartenant à la même section (la spécification de l'application exige que la section résultante contienne au moins un investigateur α et un investigateur β). Le fait que ces trois investigateurs appartiennent à la même section est garanti par l'existence d'un lien de communication entre ces composants et entre un même coordinateur (i.e. arcs N1$\xrightarrow{O+R}$N4, N2$\xrightarrow{O}$N4 et N3$\xrightarrow{O}$N4). L'application de cette règle supprime le nœud représentant l'investigateur fautif et remplace le nœud représentant l'investigateur β par un nœud représentant ce même composant (i.e. même identifiant et même machine) mais dans le rôle α. L'instruction de connexion ic permet de relier tous les autres investigateurs β vers le nouvel investigateur α par un lien O.

La combinaison $R4a_A; R4b_A*$ traite la reconfiguration de l'architecture suite au passage d'une section de la phase d'exploration vers la phase d'action. L'application de la règle $R4a_A$ permet de remplacer le nœud représentant le coordinateur de la section (nœud N3) par un nœud représentant ce même coordinateur (i.e. même identifiant et même machine) en phase d'action (nœud N5). Le nœud (i.e. N1) représentant l'investigateur qui découvre la situation critique (i.e. dont la machine correspond au paramètre transporté par l'événement de reconfiguration) est remplacé par un nœud (i.e. N4) décrivant le même investigateur mais dans le rôle α. L'instruction de connexion ic permet de transférer les liens des investigateurs vers le nouveau composant du coordinateur en phase d'action.

La règle de reconfiguration $R4b_A$ permet, à chaque application, de remplacer le composant d'un investigateur en phase d'exploration par le composant correspondant au rôle β d'un investigateur en phase d'action et d'introduire correctement les liens avec le coordinateur et l'investigateur α de la section.

La combinaison $R5a_A; R5b_M$ correspond au passage d'une section de la phase d'action vers la phase d'exploration. La règle $R5a_M$ implique, donc, de remplacer le nœud décrivant le coordinateur (i.e. N2) dans la phase d'action par un nœud représentant ce coordinateur en phase d'exploration (i.e. N4). Le nœud représentant l'investigateur α est replacé par un nœud correspondant à ce même investigateur en phase d'exploration. L'instruction ic permet de relier les autres investigateurs au coordinateur. La règle $R5b_M$ transforme les nœuds des investigateurs β en les remplaçant par des nœuds correspondant à la phase d'exploration.

3.6.2 Le niveau middleware

En plus des événements considérés dans le niveau application, le niveau réseau considère d'autres problématiques correspondant à d'autres besoins d'adaptabilité et de provisionnement de la QdS.

On suppose que les gestionnaires déployés sur le contrôleur ou sur les coordinateurs béneficient d'un contexte d'exécution optimum. Par contre, les gestionnaires déployés sur les machines mobiles des investigateurs peuvent se retrouver dans une situation où il ne peuvent délivrer la QdS éxigée pour que l'application s'exécute normalement. Dans ce cas il est nécessaire de redéployer ces gestionnaires sur d'autres machines. Nous considérons les deux cas de figure où le redéploiement est effectué sur la machine du coordinateur de la section ou sur la machine d'un autre investigateur.

D'un autre côté, les machines des coordinateurs sont des machines fixes alors que les investigateurs sont déployés sur des machines mobiles. Il peut arriver, donc, que les investigateurs se trouvent physiquement très éloignés du coordinateur. Dans ce cas, il est probablement plus judicieux, pour les communications entre les investigateurs, de redéployer les gestionnaires des canaux sur la machine d'un investigateur afin d'optimiser les temps de communication.

Nous définissons donc les quatre événements de reconfiguration suivants correspondant aux cas du redéploiement d'un gestionnaire de canal d'une machine vers une autre machine et aux deux phases d'exécution de l'application comprenant l'exploration et l'action :

- *Redeploy*$\varphi 1_{I\rightarrow I}$(m1,m2) décrit un événement relatif au redéploiement d'un gestionnaire de canal déployé sur la machine *m1* d'un investigateur, vers une machine *m2* qui héberge un autre investigateur appartenant à la même section. Cet événement correspond à une section en phase d'exploration.
- *Redeploy*$\varphi 2_{I\rightarrow I}$(m1,m2) traite le redéploiement d'un gestionnaire de canal mais pour une section en phase d'action. Les paramètres de cet événement ont la même sémantique que pour l'événement précédent.
- *Redeploy*$\varphi 1_{I\rightarrow C}$(m1,m2) correspond à un événement de redéploiement d'un gestionnaire de canal dont la section est en phase d'exploration. Ce gestionnaire de canal sera redéployé de la machine $m1$ hébergeant un investigateur vers la machine m2 qui héberge le coordinateur de la section.
- *Redeploy*$\varphi 2_{I\rightarrow C}$(m1,m2) est l'événement équivalent à l'événement $Redeploy_{\varphi 1,C}$(m1,m2) en considérant une section dans la phase d'action.
- *Redeploy*$\varphi 1_{C\rightarrow I}$(m1,m2) permet de redéployer, en phase d'exploration, un gestionnaire de canal de la machine du coordinateur vers la machine d'un investigateur appartenant à sa section.
- *Redeploy*$\varphi 2_{C\rightarrow I}$(m1,m2) permet de redéployer, en phase d'action, un gestionnaire de canal β de la machine du coordinateur vers la machine d'un investigateur β appartenant à sa section.

Nous obtenons, donc, le protocole de reconfiguration PR_M qui prend en compte les événements définis ci-dessus ainsi que les événements considérés au niveau application.

PR_M= {(T=$Exclude_{\varphi 1}$(m),
C= ([$R1a_M$(X_G,M1,T) ; $R1b_M$(X_{P1},X_{P2},M1,X_{G1},M2,X_{G2},M3,X_{P3},X_{P4},M4)] ||
[$R1c_M$(X_{P1},X_{P1},X_{G1},M1,X_{G2},M2,X_{P3},X_{P4},M3,X_{C1},X_{C2},M4)] ||
[$R1d_M$(X_{P1},X_{P2},X_{G2},M1,X_{G1},M2,X_{P3},X_{P4},M3,X_{C1},X_{C2},M4)]),
I={(M1,m)}) ;
(T=$Exclude_{\beta}$(m),
C= ([$R2a_M$(X_G,M1,T) ; $R2b_M$(X_{P1},X_{C1},M1,X_{G1},M2,X_{G2},M3,X_{P2},X_{C2},M4)] ||
[$R2c_M$(X_{P1},X_{C1},X_{G1},M1,X_{G2},M2,X_{P2},X_{C2},M3,X_{C3},M4)] ||
[$R2d_M$(X_{P1},X_{C1},X_{G2},M1,X_{G1},M2,X_{P2},X_{C2},M3,X_{C3},M4,X_{P3},M5)]),
I={(M1,m)}) ;
(T=$Exclude_{\alpha}$(m1,m2),
C= ([$R3a_M$(X_G,M2,T) ;
$R3b_M$(X_{P1},X_{P2},X_{P3},M1,X_{P4},X_{C1},M2,X_{P5},X_{C2},M3,X_{G1},X_{G2},M4,X_{G3},M5)] ||
[$R3c_M$(X_{P1},X_{P2},X_{P3},M1,X_{P4},X_{C1},X_{G3},M2,X_{P5},X_{C2},M3,X_{G1},X_{G2},M4] ||
[$R3d_M$(X_{P1},X_{P2},X_{P3},M1,X_{P4},X_{C1},X_{G4},M2,X_{P5},X_{C2},M3,X_{G1},X_{G2},M4,X_{G3},
M5)]),
I={(M1,m1),(M2,m2)}) ;
(T=*CriticalSituation*(m),
C= ([$R4a_M$(X_{P1},X_{P2},M1,X_{P3},X_{P4},M2,X_{G1},M3,X_{G2},M4,X_{C1},X_{C2},M5)] ;
[$R4b_M$(X_{G1},X_{G2},M2,X_{P1},X_{P2},M3) *]),
I={(M1,m)}) ;
(T=*EndCritical*(m),
C= ([$R5a_M$(X_{G1},X_{G2},X_{C1},X_{C2},X_{C3},M1,X_{G3},M2,X_{G4},M3,X_{P1},X_{P2},X_{P3},M4,X_{P4},
X_{C4},M5)] ;
[$R5b_M$(X_{G1},X_{G2},M1,X_{P1},X_C,M2) *]),
I={(M1,m)}) ;
(T=*Redeploy*$\varphi 1_{I\rightarrow I}$(m1,m2),
C= ([$R6a_M$(X_{P1},X_{G1},M1,X_{P2},X_{G2},M2,X_C,M3)]||
[$R6b_M$(X_{P1},X_{G1},M1,X_{P2},X_{G2},M2,X_C,M3)]),
I={(M1,m1),(M2,m2)}) ;
(T=*Redeploy*$\varphi 2_{I\rightarrow I}$(m1,m2),
C= ([$R7a_M$(X_{P1},X_{G1},M1,X_{P2},X_{G2},M2,X_C,M3)]||
[$R7b_M$(X_{C1},X_{G1},M1,X_{C2},X_{G2},M2,X_P,M3)]),
I={(M1,m1),(M2,m2)}) ;

(T=$Redeploy\varphi 1_{I\to C}$(m1,m2),
C= ([$R8a_M$(X_{P1},X_{G1},M1,X_C,M2)]||
[$R8b_M$(X_{P1},X_{G1},M1,X_C,M2)]),
I={(M1,m1),(M2,m2)});
(T=$Redeploy\varphi 2_{I\to C}$(m1,m2),
C= ([$R9a_M$(X_{P1},X_{G1},M1,X_C,M2)]||
[$R9b_M$(X_{P1},X_{C1},X_{G1},M1,X_{P2},M3,X_{G2},M4,X_{C2},M2)]),
I={(M1,m1),(M2,m2)});
(T=$Redeploy\varphi 1_{C\to I}$(m1,m2),
C= ([$R10a_M$(X_{C1},X_{G1},M1,X_{P1},X_{G2},M2)]||
[$R10b_M$(X_{C1},X_{G1},M1,X_{P1},X_{G2},M2)]),
I={(M1,m1),(M2,m2)});
C= ([$R11a_M$(X_{C1},X_{G1},M1,X_{P1},X_{G2},M2)]||
[$R11b_M$(X_{G1},M1,X_{P1},M2,X_{C1},X_{G2},M3)]),
I={(M1,m1),(M3,m2)});

Nous donnons dans les figures qui suivent, pour chaque événement de reconfiguration du protocole PR_M, les règles de reconfiguration considérées dans les combinaisons correspondantes.

$R1a_M$ = (L= { }; K= { }; R= { }; N={N1(X_G,"G",M1,T)}; C={ })
$R1b_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{P2},"P_{Inv}",M1,"O"), N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M3,"O"), N1$\xrightarrow{R}$N3, N2$\xrightarrow{O}$N4, N5(X_{P3},"P_{Inv}",M4,"R"), N6(X_{P4},"P_{Inv}",M4,"O"), N5$\xrightarrow{R}$N3,N6$\xrightarrow{O}$N4}; K= {N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M3,"O"), N5(X_{P3},"P_{Inv}",M4,"R"), N6(X_{P4},"P_{Inv}",M4,"O"), N5$\xrightarrow{R}$N3, N6$\xrightarrow{O}$N4}; R={ }; N={ }; C={ })
$R1c_M$= (L={N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{P2},"P_{Inv}",M1,"O"), N3(X_{G1},"G",M1,"R"), N4(X_{G2},"G",M2,"O"), N1$\xrightarrow{R}$N3, N2$\xrightarrow{O}$N4, N5(X_{P3},"P_{Inv}",M3,"R"), N6(X_{P4},"P_{Inv}",M3,"O"), N5$\xrightarrow{R}$N3, N6$\xrightarrow{O}$N4, N7(X_{C1},"C_{coord}",M4,"R"), N8(X_{C2},"C_{coord}",M4,"O"), N3$\xrightarrow{R}$N7, N4$\xrightarrow{O}$N8}; K= {N4(X_{G2},"G",M2,"O"), N5(X_{P3},"P_{Inv}",M3,"R"), N6(X_{P4},"P_{Inv}",M3,"O"), N6$\xrightarrow{O}$N4, N7(X_{C1},"C_{coord}",M4,"R"), N8(X_{C2},"C_{coord}",M4,"O"), N3$\xrightarrow{R}$N7, N4$\xrightarrow{O}$N8 }; R={N9(X_{G1},"G",M4,"R"), N5$\xrightarrow{R}$N7, N9$\xrightarrow{R}$N7}; N={ }; C={ic}), avec ic=(N9,(X_P,"P_{Inv}",M1,"R"),"R"/"R",in,in)
$R1d_M$= (L={N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{P2},"P_{Inv}",M1,"O"), N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M1,"O"), N1$\xrightarrow{R}$N3, N2$\xrightarrow{O}$N4, N5(X_{P3},"P_{Inv}",M3,"R"), N6(X_{P4},"P_{Inv}",M3,"O"), N5$\xrightarrow{R}$N3, N6$\xrightarrow{O}$N4, N7(X_{C1},"C_{coord}",M4,"R"), N8(X_{C2},"C_{coord}",M4,"O"), N3$\xrightarrow{R}$N7, N4$\xrightarrow{O}$N8}; K= {N3(X_{G1},"G",M2,"R"), N5(X_{P3},"P_{Inv}",M3,"R"), N6(X_{P4},"P_{Inv}",M3,"O"), N5$\xrightarrow{O}$N3, N7(X_{C1},"C_{coord}",M4,"R"), N8(X_{C2},"C_{coord}",M4,"O"), N3$\xrightarrow{R}$N7, N4$\xrightarrow{O}$N8 }; R={N9(X_{G2},"G",M4,"O"), N6$\xrightarrow{O}$N9, N9$\xrightarrow{O}$N8}; N={ }; C={ic}), avec ic=(N9,(X_P,"P_{Inv}",M1,"O"),"O"/"O",in,in)

FIG. 3.22 – Règles de transformation pour le traitement de la panne d'un investigateur en phase d'exploration

En phase d'exploration, l'exclusion d'un investigateur en panne est traitée par la combinaison $C = (R1a_M; R1b_M)||(R1c_M)||(R2c_M)$ (figure 3.22). Cette combinaison tient compte des trois possibilités qui concernent le déploiement des gestionnaires de canaux correspondant à : 1) la machine de l'investigateur fautif n'héberge pas de gestionnaires de canaux, 2) la machine de l'investigateur fautif héberge le gestionnaire du canal R, 3) la machine de l'investigateur fautif héberge le gestionnaire du canal O. Ces trois cas de figure sont exclusifs puisque la machine mobile d'un investigateur ne peut héberger les deux gestionnaires de canaux en même temps.

La sous combinaison $R1a_M; R1b_M$ correspond au cas de figure où l'investigateur à éliminer n'héberge aucun gestionnaire. Ceci est assuré par la première règle de transformation (le nœud N1 considéré dans la condition d'application négative) dont l'applicabilité conditionne l'applicabilité de la sous combinaison. Dans le cas où $R1a_M$ est applicable[38], l'application de la règle de transformation $R1b_M$ permet de supprimer les nœuds décrivant l'investigateur de la machine M1 (nœuds N1 et N2) en s'assurant qu'il existe au moins un autre investigateur dans sa section (nœuds N5 et N6).

Les règles de transformation $R1c_M$ et $R1d_M$ traitent respectivement les cas où un gestionnaire de type R ou un gestionnaire de type O sont hébergés par la machine M1. Ces règles de transformation impliquent le redéploiement de ce gestionnaire sur la machine du coordinateur de la section (dans les deux règles la machine M4). Ces deux règles exigent aussi la présence d'un autre investigateur (nœuds N5 et N6 dans les deux règles) pour être appliquées.

$R2a_M$ = (L= { }; K= { }; R= { }; N={N1(X_G,"G_β",M1,T)}; C={ })
$R2b_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{C1},"C_{Inv}",M1,"O"), N3(X_{G1},"G_β",M2,"R"), N4(X_{G2},"G_β",M3,"O"), N1$\xrightarrow{R}$N3, N4$\xrightarrow{O}$N2, N5(X_{P2},"P_{Inv}",M4,"R"), N6(X_{C2},"C_{Inv}",M4,"O"), N5$\xrightarrow{R}$N3, N4$\xrightarrow{O}$N6}; K= {N3(X_{G1},"G_β",M2,"R"), N4(X_{G2},"G_β",M3,"O"), N5(X_{P2},"P_{Inv}",M4,"R"), N6(X_{C2},"C_{Inv}",M4,"O"), N5$\xrightarrow{R}$N3, N4$\xrightarrow{O}$N6}; R={ }; N={ }; C={ })
$R2c_M$= (L={N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{C1},"C_{Inv}",M1,"O"), N3(X_{G1},"G_β",M1,"R"), N4(X_{G2},"G_β",M2,"O"), N1$\xrightarrow{R}$N3, N4$\xrightarrow{O}$N2, N5(X_{P2},"P_{Inv}",M3,"R"), N6(X_{C2},"C_{Inv}",M3,"O"), N5$\xrightarrow{R}$N3, N4$\xrightarrow{O}$N6, N7(X_{C3},"C_{coord}",M4,"R"), N3$\xrightarrow{R}$N7}; K= {N4(X_{G2},"G",M2,"O"), N5(X_{P2},"P_{Inv}",M3,"R"), N6(X_{C2},"C_{Inv}",M3,"O"), N6$\xrightarrow{O}$N4, N7(X_{C3},"C_{coord}",M4,"R")}; R={N8(X_{G1},"G_β",M4,"R"), (N5$\xrightarrow{R}$N8), N8$\xrightarrow{R}$N7}; N={ }; C={ic}), avec ic=(N8,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)
$R2d_M$= (L={N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{C1},"C_{Inv}",M1,"O"), N3(X_{G1},"G",M2,"R"), N4(X_{G2},"G",M1,"O"), N1$\xrightarrow{R}$N3, N4$\xrightarrow{O}$N2, N5(X_{P2},"P_{Inv}",M3,"R"), N6(X_{C2},"C_{Inv}",M3,"O"), N5$\xrightarrow{R}$N3, N6$\xrightarrow{O}$N4, N7(X_{C3},"C_{coord}",M4,"R"), N3$\xrightarrow{R}$N7, N8(X_{P3},"$P_{Inv,\alpha}$",M5,"O"), N8$\xrightarrow{O}$N4}; K= {N3(X_{G1},"G",M2,"R"), N5(X_{P2},"P_{Inv}",M3,"R"), N6(X_{C2},"C_{Inv}",M3,"O"), N5$\xrightarrow{O}$N3, N7(X_{C3},"C_{coord}",M4,"R"), N3$\xrightarrow{R}$N7, N8(X_{P3},"$P_{Inv,\alpha}$",M5,"O")}; R={N10(X_{G2},"G",M4,"O"), N8$\xrightarrow{O}$N10, N10$\xrightarrow{O}$N7}; N={ }; C={ic}), avec ic=(N10,(X_P,"P_{Inv}",M,"O"),"O"/"O",in,in)

FIG. 3.23 – Règles de transformation pour le traitement de la panne d'un investigateur β

[38] L'application de la règle $R1a_M$ ne transforme pas le graphe de l'instance de l'architecture puisque L, K et R sont vides

Pour la panne d'un investigateur β, il existe trois cas de figure correspondant à : 1) la machine de cet investigateur n'héberge pas de gestionnaires de canaux, 2) cette machine héberge le gestionnaire du canal reliant les investigateurs β au coordinateur et 3) cette machine héberge le gestionnaire du canal reliant l'investigateur α aux investigateurs β. Les trois possibilités sont exclusives.

La sous combinaison $R2a_M; R2b_M$ (figure 3.23) traite le premier cas en éliminant le producteur de rapports pour le coordinateur (nœud N1) et le consommateur des observations de l'investigateur α (nœud N2). De la même manière que pour la combinaison $R1a_M; R1b_M$, la règle de transformation $R2b_M$ permet de s'assurer que l'architecture courante correspond bien au cas numéro 1.

La règle $R2c_M$ traite le cas où le gestionnaire du canal reliant les investigateurs β au coordinateur est hébergé par la machine M1 de l'investigateur fautif. Dans ce cas, ce gestionnaire est redéployé sur la machine du coordinateur (machine M4) et la machine M1 est exclue de l'application (suppression des nœuds N1 et N2). L'application de cette règle exige bien évidemment la présence d'un autre investigateur β (i.e. nœuds N5 et N6). L'instruction de connexion ic permet de relier les autres investigateurs β au nouveau gestionnaire déployé sur le coordinateur.

La règle $R2d_M$ traite le cas où le gestionnaire du canal permettant de relier l'investigateur α aux investigateurs β est déployé sur la machine de l'investigateur fautif. L'application de cette règle implique le redéploiement de ce gestionnaire sur la machine du coordinateur. Il implique, outre la suppression des composants de l'investigateur fautif, de relier l'investigateur α ainsi que les autres investigateurs β au gestionnaire qui les connecte.

La panne d'un investigateur α hébergé dans une machine M1 implique son remplacement par un investigateur β hébergé dans une machine M2 et appartenant à sa section. La spécification de l'architecture de l'application en phase d'action interdit de déployer les gestionnaires de canaux sur la machine de l'investigateur α afin de préserver ses ressources. Pour cela, nous devons considérer les cas où la machine M2 héberge des gestionnaires.

La sous combinaison $R3a_M; R3b_M$ (figure 3.24) correspond au cas où la machine M2 n'héberge pas de gestionnaires. Dans ce cas, la règle de transformation $R3b_M$ permet d'exclure la machine M1 et de redéployer les composants de l'investigateur α sur la machine M2 et de les relier correctement aux gestionnaires de canaux permettant le lien avec le coordinateur et les autres investigateurs.

$R3a_M$ = (L= { }; K= { }; R= { }; N={N1(X_G,"G_β",M2,T)}; C={ })
$R3b_M$= (L= {N1(X_{P1},"$P_{Inv,\alpha}$",M1,"R"), N2(X_{P2},"$P_{Inv,\alpha}$",M1,"O"), N3(X_{P3},"$P_{Inv,\alpha}$",M1,"O"), N4(X_{P4},"P_{Inv}",M2,"R"), N5(X_{C1},"C_{Inv}",M2,"O"), N6(X_{P5},"P_{Inv}",M3,"R"), N7(X_{C2},"C_{Inv}",M3,"O"), N8(X_{G1},"G_α",M4,"R"), N9(X_{G2},"G_α",M4,"O"), N10(X_{G3},"G_β",M5,"O"), N1$\xrightarrow{R}$N8, N2$\xrightarrow{O}$N9, N3$\xrightarrow{O}$N10, N10$\xrightarrow{R}$N5, N10$\xrightarrow{R}$N7}; K= {N6(X_{P5},"P_{Inv}",M3,"R"), N7(X_{C2},"C_{Inv}",M3,"O"), N8(X_{G1},"G_α",M4,"R"), N9(X_{G2},"G_α",M4,"O"), N10(X_{G3},"G_β",M5,"O"), N10$\xrightarrow{R}$N5, N10$\xrightarrow{R}$N7}; R={N11(X_{P1},"$P_{Inv,\alpha}$",M2,"R"), N12(X_{P2},"$P_{Inv,\alpha}$",M2,"O"), N13(X_{P3},"$P_{Inv,\alpha}$",M2,"O"), N11$\xrightarrow{R}$N8, N12$\xrightarrow{O}$N9, N13$\xrightarrow{O}$N10}; N={ }; C={ })
$R3c_M$= (L= {N1(X_{P1},"$P_{Inv,\alpha}$",M1,"R"), N2(X_{P2},"$P_{Inv,\alpha}$",M1,"O"), N3(X_{P3},"$P_{Inv,\alpha}$",M1,"O"), N4(X_{P4},"P_{Inv}",M2,"R"), N5(X_{C1},"C_{Inv}",M2,"O"), N6(X_{P5},"P_{Inv}",M3,"R"), N7(X_{C2},"C_{Inv}",M3,"O"), N8(X_{G1},"G_α",M4,"R"), N9(X_{G2},"G_α",M4,"O"), N10(X_{G3},"G_β",M2,"O"), N1$\xrightarrow{R}$N8, N2$\xrightarrow{O}$N9, N3$\xrightarrow{O}$N10, N10$\xrightarrow{R}$N5, N10$\xrightarrow{R}$N7}; K= {N6(X_{P5},"P_{Inv}",M3,"R"), N7(X_{C2},"C_{Inv}",M3,"O"), N8(X_{G1},"G_α",M4,"R"), N9(X_{G2},"G_α",M4,"O")}; R={N11(X_{P1},"$P_{Inv,\alpha}$",M2,"R"), N12(X_{P2},"$P_{Inv,\alpha}$",M2,"O"), N13(X_{P3},"$P_{Inv,\alpha}$",M2,"O"), N14(X_{G3},"G_β",M4,"O"), N14$\xrightarrow{R}$N5, N14$\xrightarrow{R}$N7, N11$\xrightarrow{R}$N8, N12$\xrightarrow{O}$N9, N13$\xrightarrow{O}$N14}; N={ }; C={ic}), avec ic=(N14,(X_C,"C_{Inv}",M,"O"),"O"/"O",out,out)

$R3d_M$= (L= {N1(X_{P1},"$P_{Inv,\alpha}$",M1,"R"), N2(X_{P2},"$P_{Inv,\alpha}$",M1,"O"), N3(X_{P3},"$P_{Inv,\alpha}$",M1,"O"),
N4(X_{P4},"P_{Inv}",M2,"R"), N5(X_{C1},"C_{Inv}",M2,"O"), N6(X_{P5},"P_{Inv}",M3,"R"),
N7(X_{C2},"C_{Inv}",M3,"O"), N8(X_{G1},"G_α",M4,"R"), N9(X_{G2},"G_α",M4,"O"),
N10(X_{G3},"G_β",M5,"O"), N11(X_{G4},"G_β",M2,"R"), N1$\xrightarrow{R}$N8, N2$\xrightarrow{O}$N9,
N3$\xrightarrow{O}$N10, N10$\xrightarrow{R}$N5, N10$\xrightarrow{R}$N7, N4$\xrightarrow{R}$N11, N6$\xrightarrow{R}$N11};
K= {N6(X_{P5},"P_{Inv}",M3,"R"), N7(X_{C2},"C_{Inv}",M3,"O"), N8(X_{G1},"G_α",M4,"R"),
N9(X_{G2},"G_α",M4,"O"), N10(X_{G3},"G_β",M5,"O"), N10$\xrightarrow{R}$N5, N10$\xrightarrow{R}$N7};
R={N12(X_{P1},"$P_{Inv,\alpha}$",M2,"R"), N13(X_{P2},"$P_{Inv,\alpha}$",M2,"O"), N14(X_{P3},"$P_{Inv,\alpha}$",M2,"O"),
N15(X_{G4},"G_β",M4,"R"), N12$\xrightarrow{R}$N8, N13$\xrightarrow{O}$N9, N14$\xrightarrow{O}$N10, N6$\xrightarrow{R}$N15};
N={ }; C={ic})
ic=(N14,(X_C,"C_{Inv}",M,"O"),"O"/"O",in,in)

FIG. 3.24 – Règles de transformation pour la panne de l'investigateur α

La règle $R3c_M$ correspond au cas où le gestionnaire (nœud N10) reliant l'investigateur α aux investigateurs β est déployé sur la machine M2. Dans ce cas, ce gestionnaire est redéployé sur la machine du coordinateur (machine M4). De la même manière que pour la règle $R3b_M$, les composants de l'investigateur α sont redéployés sur la machine M2. L'instruction de connexion *ic* permet de connecter correctement les autres investigateurs β au gestionnaire les reliant au nouvel investigateur α.

La règle $R3d_M$ traite le cas où le gestionnaire (nœud N11) reliant les investigateurs β au coordinateur est déployé sur la machine M2. Ce gestionnaire est redéployé sur la machine du coordinateur. Les composants de l'investigateur α sont redéployés sur la machine M2. L'instruction de connexion *ic* permet de connecter les investigateurs β au coordinateur.

$R4a_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{P2},"P_{Inv}",M1,"O"), N3(X_{P3},"P_{Inv}",M2,"R"),
N4(X_{P4},"P_{Inv}",M2,"O"), N5(X_{G1},"G",M3,"R"), N6(X_{G2},"G",M4,"O"),
N7(X_{C1},"C_{coord}",M5,"R"), N8(X_{C2},"C_{coord}",M5,"O"), N1$\xrightarrow{R}$N5, N2$\xrightarrow{O}$N6, N3$\xrightarrow{R}$N5,
N4$\xrightarrow{O}$N6, N5$\xrightarrow{R}$N7, N6$\xrightarrow{O}$N8};
K= { };
R={N9(X_{P1},"$P_{Inv,\alpha}$",M1,"R"), N10(X_{P2},"$P_{Inv,\alpha}$",M1,"O"), N11(X_{P5},"$P_{Inv,\alpha}$",M1,"O"),
N12(X_{P3},"P_{Inv}",M2,"R"), N13(X_{C3},C_{Inv},M2,"O"), N14(X_{G1},"G_α",M5,"R"),
N15(X_{G2},"G_α",M5,"O"), N16(X_{G3},"G_β",M5,"R"), N17(X_{G4},"G_β",M5,"O"),
N18(X_{C1},"C_{coord}",M5,"O"), N19(X_{C2},"C_{coord}",M5,"R"), N20(X_{C4},"C_{coord}",M5,"R"),
N9$\xrightarrow{R}$N14, N10$\xrightarrow{O}$N15, N11$\xrightarrow{O}$N17, N12$\xrightarrow{R}$N16, N17$\xrightarrow{O}$N13, N14$\xrightarrow{R}$N18, N15$\xrightarrow{O}$N19,
N16$\xrightarrow{R}$N20};
N= { }; C={ic1,ic2}), avec
ic1=(N16,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)
ic2=(N17,(X_P,"P_{Inv}",M,"O"),"O"/"O",in,in)

$R4b_M$= (L= {N1(X_{G1},"G_β",M2,"R"), N2(X_{G2},"G_β",M2,"O"),
N3(X_{P1},"P_{Inv}",M3,"R"), N4(X_{P2},"P_{Inv}",M3,"O"), N3$\xrightarrow{R}$N1, N4$\xrightarrow{O}$N2};
K= {N1(X_{G1},"G_β",M2,"R"), N2(X_{G2},"G_β",M2,"O")};
R={N5(X_{P1},"P_{Inv}",M3,"R"), N6(X_{C1},"C_{inv}",M3,"O"), N5$\xrightarrow{R}$N1, N2$\xrightarrow{O}$N6};
N= { }; C={ })

FIG. 3.25 – Règles de transformation pour le passage de la phase exploration vers la phase d'action

L'avènement d'une situation critique découverte par un investigateur d'une machine $M1$ implique le passage de l'ensemble de sa section à la phase d'action. La combinaison $R4a_M$; $R4b_M$ (figure 3.25) permet

de transformer l'architecture pour s'adapter à cette nouvelle phase d'exécution.

La règle de transformation $R4a_M$ permet de remplacer les composants de l'investigateur de la machine $M1$ (i.e. nœuds N1 et N2) par des composants correspondant au nouveau rôle d'investigateur α. elle permet aussi de remplacer les gestionnaires de canaux relatifs à la phase d'exploration (i.e. nœuds N5 et N6) par des gestionnaires correspondant à une configuration en phase d'action (i.e. nœuds N14,...,N17). Le déploiement de ces gestionnaires est fait de telle manière qu'ils sont tous hébergés par la machine du coordinateur (i.e. machine M5). Cette règle produit aussi les consommateurs du coordinateur permettant de recevoir les données envoyées par l'investigateur α et les investigateurs β. Les instructions de connexion $ic1$ et $ic2$ permettent de relier le coordinateur et l'investigateur α aux autres investigateurs de la section à travers les gestionnaires correspondants (i.e. nœuds N16 et N17).

La règle de transformation $R4b_M$ permet, à chaque application, de traiter le cas d'un investigateur β de la section en remplaçant les composants correspondant à la phase d'exploration par des composants de la phase d'action. L'utilisation de l'opérateur $*$ permet de transformer tous les investigateurs de la section traitée[39] par la règle précédente pour les adapter au contexte d'exécution correspondant à la phase d'action.

$R5a_M$= (L= {N1(X_{G1},"G_α",M1,"R"), N2(X_{G2},"G_α",M1,"O"), N3(X_{G3},"G_β",M2,"R"),
N4(X_{G4},"G_β",M3,"O"), N5(X_{C1},"C_{coord}",M1,"R"), N6(X_{C2},"C_{coord}",M1,"O"),
N7(X_{C3},"C_{coord}",M1,"O"), N8(X_{P1},"$P_{Inv,\alpha}$",M4,"R"), N9(X_{P2},"$P_{Inv,\alpha}$",M4,"O"),
N10(X_{P3},"$P_{Inv,\alpha}$",M4,"O"), N11(X_{P4},"P_{Inv}",M5,"R"), N12(X_{C4},"C_{Inv}",M5,"O"),
N1$\xrightarrow{R}$N5, N2$\xrightarrow{O}$N6, N1$\xrightarrow{R}$N8, N3$\xrightarrow{R}$N7, N4$\xrightarrow{O}$N12, N8$\xrightarrow{R}$N1,
N9$\xrightarrow{O}$N2, N10$\xrightarrow{O}$N4, N11$\xrightarrow{R}$N3};
K= { };
R= {N13(X_{G1},"G",M1,"R"), N14(X_{G2},"G",M1,"O"), N15(X_{C1},"C_{coord}",M1,"R"),
N16(X_{C2},"C_{coord}",M1,"O"), N17(X_{P1},"P_{Inv}",M4,"R"), N18(X_{P2},"P_{Inv}",M4,"O"),
N19(X_{P4},"P_{Inv}",M5,"R"), N20(X_{P4},"P_{Inv}",M5,"O"), N13$\xrightarrow{R}$N15, N14$\xrightarrow{O}$N16,
N17$\xrightarrow{R}$N13, N18$\xrightarrow{O}$N14, N19$\xrightarrow{R}$N13, N20$\xrightarrow{O}$N14};
N= { }; C={ic1,ic2}), avec
ic1=(N13,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)
ic2=(N14,(X_C,"C_{Inv}",M,"O"),"O"/"O",out,out)

$R5b_M$= (L= {N1(X_{G1},"G",M1,"R"), N2(X_{G2},"G",M1,"O"), N3(X_{P1},"P_{Inv}",M2,"R"),
N4(X_C,"C_{Inv}",M2,"O"), N3$\xrightarrow{R}$N1, N2$\xrightarrow{O}$N4};
K= {N1(X_{G1},"G",M1,"R"), N2(X_{G2},"G",M1,"O")};
R= {N5(X_{P1},"P_{Inv}",M2,"R"), N6(X_{P2},"P_{Inv}",M2,"O"), N5$\xrightarrow{R}$N1, N6$\xrightarrow{O}$N2};
N= { }; C={ })

FIG. 3.26 – Règles de transformation pour la fin de la situation critique

La fin de la situation critique est traitée par la combinaison $R5a_M; R5b_M*$ (figure 3.26). Sur le même modèle que le traitement de la situation critique, la règle $R5a_M$ permet de transformer les composants de l'investigateur α et du coordinateur pour revenir à la phase d'exploration. La règle $R5b_M$ permet de ramener les investigateurs β à une architecture correspondant à la phase d'exploration.

Dans la phase d'exploration, le redéploiement d'un gestionnaire de canal de la machine d'un investigateur vers la machine d'un autre investigateur appartenant à la même section est traité par la combinaison $R6a_M||R6b_M$ (figure 3.27). La première règle correspond au cas où le gestionnaire est de type R alors que la deuxième correspond au cas où le gestionnaire traite des données de type O. Dans les deux cas le fait que les deux investigateurs fassent partie de la même section est garanti par le fait qu'ils soient reliés au même gestionnaire. La condition d'application négative introduite pour les deux règles permet de garantir

[39] C'est la bonne section qui est traitée par la règle $R4a_M$ parce que c'est la seule section qui est dans une phase transitoire de reconfiguration où des producteurs de type (X_P,"P_{Inv}",M,"O") sont connectés à des gestionnaires G_β.

$R6a_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_{P2},"P_{Inv}",M2,"R"),
N4(X_C,"C_{coord}",M3,"R"), N1$\xrightarrow{R}$N2, N3$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N4};
K= {N1(X_{P1},"P_{Inv}",M1,"R"), N3(X_{P2},"P_{Inv}",M2,"R"), N4(X_C,"C_{coord}",M3,"R")};
R= {N5(X_{G1},"G",M2,"R"), N1$\xrightarrow{R}$N5, N3$\xrightarrow{R}$N5, N5$\xrightarrow{R}$N4};
N= {N6(X_{G2},"G",M2,"O")}; C={ic1}), avec
ic1=(N5,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)

$R6b_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"O"), N2(X_{G1},"G",M1,"O"), N3(X_{P2},"P_{Inv}",M2,"O"),
N4(X_C,"C_{coord}",M3,"O"), N1$\xrightarrow{O}$N2, N3$\xrightarrow{O}$N2, N2$\xrightarrow{O}$N4};
K= {N1(X_{P1},"P_{Inv}",M1,"O"), N3(X_{P2},"P_{Inv}",M2,"O"), N4(X_C,"C_{coord}",M3,"O")};
R= {N5(X_{G1},"G",M2,"O"), N1$\xrightarrow{O}$N5, N3$\xrightarrow{O}$N5, N5$\xrightarrow{O}$N4};
N= {N6(X_{G2},"G",M2,"O")}; C={ic2}), avec
ic2=(N5,(X_P,"P_{Inv}",M,"O"),"O"/"O",in,in)

FIG. 3.27 – Règles pour le redéploiement des gestionnaires dans la phase d'exploration sur la machine d'investigateurs

que la machine sur laquelle est redéployé le gestionnaire n'héberge pas l'autre gestionnaire (puisqu'une machine d'investigateur ne peut en héberger deux en même temps). Les instructions de connexion $ic1$ et $ic2$ permettent de connecter les producteurs des investigateurs de la section au nouveau gestionnaire.

$R7a_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{G1},"G_β",M1,"R"), N3(X_{P2},"P_{Inv}",M2,"R"),
N4(X_C,"C_{coord}",M3,"R"), N1$\xrightarrow{R}$N2, N3$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N4};
K= {N1(X_{P1},"P_{Inv}",M1,"R"), N3(X_{P2},"P_{Inv}",M2,"R"), N4(X_C,"C_{coord}",M3,"R")};
R= {N5(X_{G1},"G_β",M2,"R"), N1$\xrightarrow{R}$N5, N3$\xrightarrow{R}$N5, N5$\xrightarrow{R}$N4};
N= {N6(X_{G2},"G_β",M2,"O")}; C={ic}), avec
ic=(N5,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)

$R7b_M$= (L= {N1(X_{C1},"C_{Inv}",M1,"O"), N2(X_{G1},"G_β",M1,"O"), N3(X_{C2},"C_{Inv}",M2,"O"),
N4(X_P,"$P_{Inv,\alpha}$",M3,"O"), N4$\xrightarrow{O}$N2, N2$\xrightarrow{O}$N1, N2$\xrightarrow{O}$N3};
K= {N1(X_{C1},"C_{Inv}",M1,"O"), N3(X_{C2},"C_{Inv}",M2,"O"), N4(X_P,"$P_{Inv,\alpha}$",M3,"O")};
R= {N5(X_{G1},"G_β",M2,"O"), N4$\xrightarrow{O}$N5, N5$\xrightarrow{O}$N1, N5$\xrightarrow{O}$N3};
N= {N6(X_{G2},"G_β",M2,"O")}; C={ic}), avec
ic=(N5,(X_C,"C_{Inv}",M,"O"),"O"/"O",out,out)

FIG. 3.28 – Règles de transformation pour le redéploiement des gestionnaires β sur des machines d'investigateurs

La combinaison $R7a_M||R7b_M$ (figure 3.28) traite le cas du redéploiement des gestionnaires (forcément β) de la machine d'un investigateur vers la machine d'un autre investigateur. La règle $R7a_M$ traite le cas du gestionnaire reliant les investigateurs β au coordinateur alors que la règle $R7b_M$ correspond au redéploiement du gestionnaire reliant l'investigateur α aux investigateurs β de sa section.

$R8a_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{G1},"G",M1,"R"), N3(X_C,"C_{coord}",M2,"R"),
N1$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N3};
K= {N1(X_{P1},"P_{Inv}",M1,"R"), N3(X_C,"C_{coord}",M2,"R")};
R= {N4(X_{G1},"G",M2,"R"), N1$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N3};
N= { }; C={ic}), avec
ic=(N4,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)

$R8b_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"O"), N2(X_{G1},"G",M1,"O"), N3(X_C,"C_{coord}",M2,"O"),
N1$\xrightarrow{O}$N2, N2$\xrightarrow{O}$N3};
K= {N1(X_{P1},"P_{Inv}",M1,"O"), N3(X_C,"C_{coord}",M2,"O")};
R= {N4(X_{G1},"G",M2,"O"), N1$\xrightarrow{O}$N2, N2$\xrightarrow{O}$N3};
N= { }; C={ic}), avec
ic=(N4,(X_P,"P_{Inv}",M,"O"),"O"/"O",in,in)

$R9a_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{G1},"G_β",M1,"R"),
N3(X_C,"C_{coord}",M2,"R"), N1$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N3};
K= {N1(X_{P1},"P_{Inv}",M1,"R"), N3(X_C,"C_{coord}",M2,"R")};
R= {N4(X_{G1},"G_β",M2,"R"), N1$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N3};
N= { }; C={ic}), avec
ic=(N4,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)

$R9b_M$= (L= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{C1},"C_{Inv}",M1,"O"),
N3(X_{G1},"G_β",M1,"O"), N4(X_{P2},"$P_{Inv,\alpha}$",M3,"O"),
N5(X_{G2},"G_β",M4,"R"), N6(X_{C2},"C_{coord}",M2,"R"),
N1$\xrightarrow{R}$N5, N3$\xrightarrow{O}$N2, N4$\xrightarrow{O}$N3, N5$\xrightarrow{R}$N6};
K= {N1(X_{P1},"P_{Inv}",M1,"R"), N2(X_{C1},"C_{Inv}",M1,"O"),
N4(X_{P2},"$P_{Inv,\alpha}$",M3,"O"), N5(X_{G2},"G_β",M4,"R"),
N6(X_{C2},"C_{coord}",M2,"R"), N1$\xrightarrow{R}$N5, N5$\xrightarrow{R}$N6};
R= {N7(X_{G1},"G_β",M2,"O"), N7$\xrightarrow{O}$N2, N4$\xrightarrow{O}$N7};
N= { }; C={ic}), avec
ic=(N7,(X_C,"C_{Inv}",M,"O"),"O"/"O",out,out)

FIG. 3.29 – Règles de transformation pour le redéploiement des gestionnaires sur la machine du coordinateur

Les combinaisons $R8a_M||R8b_M$ et $R9a_M||R9b_M$ (figure 3.29) traitent le redéploiement des gestionnaires de canaux vers la machine du coordinateur respectivement dans les phases d'exploration et d'action. Pour chaque combinaison, la première règle correspond au traitement des gestionnaires relatifs aux rapports alors que la deuxième règle correspond aux gestionnaires acheminant les données O.

$R10a_M$= (L= {N1(X_{C1},"C_{coord}",M1,"R"), N2(X_{G1},"G",M1,"R"),
N3(X_{P1},"P_{Inv}",M2,"R"), N3$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N1};
K= {N1(X_{C1},"C_{coord}",M1,"R"), N3(X_{P1},"P_{Inv}",M2,"R")};
R= {N4(X_{G1},"G",M2,"R"), N3$\xrightarrow{R}$N4, N4$\xrightarrow{R}$N1};
N= {N5(X_{G2},"G",M2,"O")};
C={ic}), avec
ic=(N4,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)

$R10b_M$= (L= {N1(X_{C1},"C_{coord}",M1,"O"), N2(X_{G1},"G",M1,"O"),
N3(X_{P1},"P_{Inv}",M2,"O"), N3$\xrightarrow{O}$N2, N2$\xrightarrow{O}$N1};
K= {N1(X_{C1},"C_{coord}",M1,"O"), N3(X_{P1},"P_{Inv}",M2,"O")};
R= {N4(X_{G1},"G",M2,"O"), N3$\xrightarrow{O}$N4, N4$\xrightarrow{O}$N1};
N= {N5(X_{G2},"G",M2,"R")};
C={ic}), avec
ic=(N4,(X_P,"P_{Inv}",M,"O"),"O"/"O",in,in)

$R11a_M$= (L= {N1(X_{C1},"C_{coord}",M1,"R"), N2(X_{G1},"G_β",M1,"R"),
N3(X_{P1},"P_{Inv}",M2,"R"), N3$\xrightarrow{R}$N2, N2$\xrightarrow{R}$N1};
K= {N1(X_{C1},"C_{coord}",M1,"R"), N3(X_{P1},"P_{Inv}",M2,"R")};
R= {N4(X_{G1},"G_β",M2,"R"), N3$\xrightarrow{R}$N4, N4$\xrightarrow{R}$N1};
N= {N5(X_{G2},"G_β",M2,"O")};
C={ic}), avec
ic=(N4,(X_P,"P_{Inv}",M,"R"),"R"/"R",in,in)

$R11b_M$= (L= {N1(X_{G1},"G_β",M1,"O"), N2(X_{P1},"$P_{Inv,\alpha}$",M2,"O"),
N3(X_{C1},"C_{Inv}",M3,"O"), N1$\xrightarrow{O}$N3, N2$\xrightarrow{O}$N1};
K= {N2(X_{P1},"$P_{Inv,\alpha}$",M2,"O"), N3(X_{C1},"C_{Inv}",M3,"O")};
R= {N4(X_{G1},"G_β",M3,"O"), N2$\xrightarrow{O}$N4, N4$\xrightarrow{O}$N3};
N= {N5(X_{G2},"G_β",M3,"R")};
C={ic}), avec
ic=(N4,(X_C,"C_{Inv}",M,"O"),"O"/"O",out,out)

FIG. 3.30 – Règles de transformation pour le redéploiement des gestionnaires de la machine du coordinateur

La combinaison $R10a_M||R10b_M$ correspond au redéploiement d'un gestionnaire de canal en phase d'exploration de la machine d'un coordinateur vers la machine d'un investigateur de sa section. L'appartenance de l'investigateur à la même section est garanti par le fait que le producteur de l'investigateur et le consommateur du coordinateur sont reliés par le gestionnaire à redéployer (i.e. arcs reliant N3 à N2 et reliant N2 à N1). Chacune des deux règles qui compose cette combinaison correspond à un des deux types de données traités par le gestionnaire (R ou O). Pour ces deux règles nous nous assurons (via la condition d'application négative) que l'investigateur destinataire n'héberge pas déjà l'autre gestionnaire de la section. Ensuite, nous remplaçons le nœud correspondant au gestionnaire par un autre nœud correspondant à la machine de redéploiement (i.e. machine M2). L'instruction de connexion *ic* permet, pour les deux règles, de connecter les producteurs des autres investigateurs au gestionnaire redéployé.

La combinaison $R11a_M||R11b_M$ correspond au redéploiement des gestionnaires de canaux β de la machine du coorrdinateur vers la machine d'un investigateur β. De la même manière que la combinaison précédente, chacune des règles de la combinaison permet de traiter le cas d'un type de données échangées et interdit la reconfiguration dans le cas où l'investigateur héberge l'autre gestionnaire β.

3.7 Conclusion

Dans ce chapitre nous avons présenté un cas d'étude relatif aux activités d'intervention d'urgence. Ce cas d'étude admet des exigences d'adaptabilité de son architecture dues aux changements du contexte d'exécution (passage d'une phase d'exécution à une autre, à la panne des composants (destruction ou épuisement des ressources d'énergie des investigateurs) et aux besoins de provisionnement de la qualité de service.

Pour les deux niveaux d'abstraction supérieurs, nous avons donné les systèmes de raffinement et d'abstraction ainsi que les protocoles de reconfiguration de l'architecture et les propriétés architecturales. Afin de ne pas surcharger ce chapitre, les systèmes transformations horizontales et verticales impliquant le niveau le plus bas sont donnés dans [Gue06].

Les différents modèles présentés précédemment concernant la description des styles architecturaux, de l'abstraction et du raffinement des architectures, et de la gestion de l'évolution dynamique sont basés sur des grammaires de graphes et des règles de transformation. La base théorique, que ce soit pour les productions de grammaire ou pour les règles de transformation, permettant l'implémentation de ce type de modèles concerne la recherche d'homomorphismes de graphes.

Nous présentons, dans le chapitre qui suit, deux algorithmes de recherche d'homomorphismes. Ces algorithmes prennent en compte les extensions introduites pour notre modèle en considérant des nœuds

multi-labélés, des arcs multi-étiquetés et des labels variables. Le premier algorithme est généraliste en s'adressant aux systèmes de réécriture de graphes alors que le second algorithme tire profit du fait que l'ensemble des règles de réécriture (dans les deux cas des grammaires et du protocole de reconfiguration) est stable.

Les deux algorithmes de recherche d'homomorphismes sont étendus pour permettre l'implémentation des systèmes de réécriture de graphes prenant en compte les modèles considérés dans la description des styles et dans les transformations verticales et horizontales.

Chapitre 4

Algorithmes de Recherche d'Homomorphismes et de Transformation de Graphes

4.1 Introduction

Dans les chapitres précédents, nous avons présenté un méta-modèle pour la description et la gestion de l'évolution des architectures logicielles dynamiques. Un des objectifs importants de nos travaux est d'implémenter des outils logiciels pour la vérification et la simulation de ces modèles, et d'offrir des modules de programmation génériques pour les protocoles de reconfiguration. L'avantage que nous désirons en tirer est de passer de la phase de spécification et de conception vers la phase d'implémentation en gardant les mêmes concepts et la même vision de l'application (i.e. 1 nœud= 1 composant, 1 arc = 1 interdépendance).

La réécriture de graphes et les grammaires de graphes constituent la base formelle de notre modèle. L'implémentation de tout outil logiciel relatif à ce modèle exige, au préalable, de définir et d'implémenter les algorithmes performants de recherche d'homomorphismes et de transformation de graphes en prenant en compte le type de graphes et de règles de réécriture considérés. L'analyse de complexité en plus des résultats expérimentaux nous apportent des enseignement importants par rapport à la faisabilité de l'approche et à son applicabilité dans le cas d'architectures de grandes tailles. En effet, le temps d'exécution de ces algorithmes est un facteur déterminant dans la mesure où l'approche est destinée à une utilisation pendant l'exécution.

Nous prenons en compte les multiples extensions pour la définition de nos systèmes de réécriture impliquant, par exemple, des variables dans les labels, des conditions d'application négatives et des instructions de connexion.

Dans ce chapitre, nous définissons, dans le cas le plus général, un algorithme de recherche d'homomorphismes et de réécriture de graphes relevant de notre méta-modèle. Pour cet algorithme, nous fournissons une étude de complexité et des résultats expérimentaux.

Ensuite, nous définissons, pour le contexte spécifique de notre modèle qui implique un ensemble de règles de réécriture stable[40], un algorithme de recherche d'homomorphismes et de transformation de graphes qui exploite cette spécificité. Pour cet algorithme nous réalisons une étude de complexité en comparaison avec le premier algorithme.

[40] Dans le cas des styles architecturaux et des systèmes de transformation verticale : un ensemble fixe de productions de grammaire. Dans le cas du protocole de reconfiguration, un ensemble de règles de reconfiguration pratiquement stable.

4.2 Définitions et notations

Dans le cadre des problématiques liées à la recherche d'homomorphismes, nous utiliserons la terminologie *graphe modèle* et *graphe d'entrée* pour désigner respectivement les graphes constituant le domaine d'entrée et le domaine d'arrivée des homomorphismes[41]. Dans le cadre de la réécriture de graphes nous utilisons le terme générique *règle de réécriture* et le terme *graphe hôte* pour respectivement désigner la règle à appliquer et le graphe sur lequel elle va être appliquée.

Dans le cadre de notre méta-modèle, nous prenons en compte des graphes qui considèrent des labels multiples et variables. Nous reprenons donc les définitions les plus génériques données dans le chapitre 2. Nous donnons ici, pour rappel, la définition du type de graphes considéré pour la recherche d'homomorphismes.

***Définition* 13 (Caractérisation des graphes du modèle)** *Un graphe considérant n types de nœuds et m types d'arcs différents et l'ensemble des variables $S_X = \{X_1, \ldots, X_i\}$ est défini par le système $G = (N, A, Lab, D_{Lab}, Etiq, D_{Etiq})$ avec $D_{Lab} = [(L_1 \cup S1_X) \cup \ldots \cup (L_n \cup Sn_X)]$ et $D_{Etiq} = (E_1 \cup \ldots \cup E_m)$. N et A correspondent à l'ensemble des nœuds et des arcs du graphe. $L_1, \ldots, L_n$ correspondent aux domaines de définition des labels des différents types de nœuds, Lab correspond à la fonction permettant la génération de ces labels, et Si_X est un sous ensemble de S_X. $E_1, \ldots, E_m$ et $Etiq$ correspondent respectivement aux domaines de définition des étiquettes des différents types d'arcs et de la fonction permettant leur génération.*

Nous rappelons, dans la définition suivante, le concept d'unification des labels de nœuds (abordé de manière informelle dans le chapitre 2) en considérant des variables et des constantes.

***Définition* 14 (Unification de labels)** *Deux labels l_1 appartenant à un nœud du graphe modèle et l_2 appartenant à un nœud du graphe d'entrée sont unifiables si et seulement si les deux conditions suivantes sont vérifiées :*
1- l_1 et l_2 sont deux constantes de même type et de même valeur.
2- l_1 est une variable de même type que la constante l_2.

La procédure présentée par la figure 4.1 implémente l'unification des labels de nœuds telle que cela est défini ci-dessus. Si la procédure considère un label variable, elle retourne l'identifiant de cette variable et sa valeur associée. Elle retourne $NULL$ si les deux labels ne sont pas unifiables.

UNIFY_PARAMS(l_1, l_2)
1- if l_1 is *constant* then
a. if $l1.type \neq l2.type$ or $l_1.value \neq l_2.value$ then return(NULL)
b. else return($\emptyset$)
2- if l_1 is *variable* then
a. if $l_1.type = l_2.type$ then return($\{(l_1, l_2)\}$)
b. else return(NULL)

FIG. 4.1 – Pseudo-code de la procédure UNIFY_PARAMS.

L'unification des nœuds qui découle de cette définition est donnée dans la définition 15 en prenant en compte l'ensemble des labels d'un nœud appartenant à un graphe modèle et l'ensemble des labels d'un nœud appartenant à un graphe d'entrée[42].

***Définition* 15 (Unification de nœuds)** *Deux nœuds n_1 et n_2 sont unifiables si et seulement si les trois conditions suivantes sont vérifiées :*

[41] De manière informelle, la recherche d'homomorphismes entre un graphe modèle et un graphe d'entrée établi une application injective entre les nœuds du premier et les nœuds du deuxième graphe en respectant la structure.

[42] Comme on a vu dans le chapitre 2, le premier nœud admet des labels variables alors que le second considère exclusivement des labels constant

1- Les deux nœuds possèdent le même nombre de labels.
2- Ces labels sont, deux à deux, unifiables en tenant compte de l'ordre de leur occurrence [43].
3- Le résultat de toutes les unifications de labels est consistant.

La consistance des unifications de labels (évoquée dans la condition 3) doit être maintenue pour prévenir une variable, présente par exemple dans deux labels, d'être affectée par deux valeurs différentes. Par exemple, un nœud $n_1(x, y, x, 3)$ n'est pas unifiable avec le nœud $n_2(1, 1, 2, 3)$ (même si le label x est unifiable à la fois avec la constante 1 et avec la constante 2) mais est unifiable avec $n'_2(1, 2, 1, 3)$ et produit, comme résultat, l'ensemble d'affectation $\{(x, 1), (y, 2)\}$. Il est, donc, nécessaire de vérifier que les affectations assignées au différents labels d'un nœud du graphe modèle sont consistantes. Cette consistance est présentée formellement dans la définition 16.

***Définition* 16 (Affectations consistantes)** *Deux affectations (ou valuations) Val_1 et Val_2 sont consistantes si et seulement si, pour chaque paire $(x_1, value_1)$ appartenant à Val_1 et chaque paire $(x_2, value_2)$ appartenant à Val_2, une des deux conditions suivantes est vérifiée :*
1- x_1 et x_2 sont deux variables différentes (syntaxiquement), ou
2- x_1 et x_2 correspondent à la même variable et lui associent la même affectation (i.e. $x_1 = x_2$ and $value_1 = value_2$).

MERGE(Val_1, Val_2)
0- Let $Val_{1.2} = \emptyset$
1- For all valuations $(x_1, value_1) \in Val_1$
a. For all valuations $(x_2, value_2) \in Val_2$
i. if $x_1 = x_2$ and $value_1 \neq value_2$ then return(NULL)
2- $Val_{1.2} = Val_1 \cup Val_2$
3- return$(Val_{1.2})$

FIG. 4.2 – Pseudo-code de la procédure MERGE.

L'opération de fusion de deux affectations est implémentée par la procédure MERGE. La description algorithmique de cette procédure est donnée dans la figure 4.2. Le résultat de la fusion de deux affectations Val_1 et Val_2, quand elles sont consistantes, est la valuation $Val_{1.2}$ contenant toutes les affectations de Val_1 et de Val_2. La procédure MERGE retourne la valeur *NULL* si la fusion des deux affectations est non consistante.

UNIFY_VERTICES(n_1, n_2)
0- Let Val = $\emptyset$ and let $Lab(n_1) = (l_{1.1}, \ldots, l_{1.m})$ and let $Lab'(n_2) = (l_{2.1}, \ldots, l_{2.k})$
1- if $(m \neq k)$ then return(NULL)
2- Let i=1
3- While $i \leq m$
a. V=UNIFY_PARAMS$(l_{1.i}, l_{2.i})$
b. if V=NULL then return(NULL)
c. Val=MERGE(Val, V)
d. if Val=NULL then return(NULL)
e. i=i+1
4- return(Val)

FIG. 4.3 – Pseudo-code de la procédure UNIFY_VERTICES.

[43] Ceci correspond à dire que si n_1 est labélé par (x_1, x_2) et n_2 labélé par (1,"s"), alors n_1 est unifiable avec n_2 implique que le labels x_1 est unifiable avec 1, et que x_2 est unifiable avec "s".

L'unification de deux nœuds, appartenant respectivement à un graphe modèle et à un graphe d'entrée, est implémentée par la procédure UNIFY_VERTICES. La description algorithmique de cette procédure est donnée dans la figure 4.3. Cette procédure retourne l'ensemble de toutes les affectations consistantes assignées aux labels variables des nœuds du graphe modèle si les deux nœuds sont unifiables. Autrement, elle retourne *NULL*.

4.3 Algorithmes génériques

4.3.1 Algorithme pour la recherche des homomorphismes de graphes

En considérant les éléments d'extension introduits pour les graphes étendus et les contraintes relatives à la consistance des affectations des labels, nous introduisons les homomorphismes de graphes dans la définition suivante.

***Définition* 17 (homomorphismes de graphes)** *Deux graphes G et G' tels que $G = (N, A, Lab, D_{Lab}, Etiq, D_{Etiq})$ et $G' = (N', A', Lab', D_{Lab'}, Etiq', D_{Etiq'})$ sont homomorphes si et seulement s'il existe une affectation consistante Val_{G_1,G_2} et une application injective $f \subseteq N \times N'$, telles que :*
1) Pour tous nœuds $n_1, n_2 \in N$ et $n'_1, n'_2 \in N'$ avec $(n_1, n'_1) \in f$ et $(n_2, n'_2) \in f$, si $(n_1, n_2) \in A$, alors $(n'_1, n'_2) \in A'$ et $Etiq((n_1, n_2)) = Etiq'((n'_1, n'_2))$.
2) L'unification des paramètres de tous les nœuds associés par f (i.e. $(n, n') \in f$), est possible et produit un résultat global consistant.
L'homomorphisme de graphes résultant est caractérisé par la paire $(f, Val_{G,G'})$.

Pour la définition de l'algorithme de recherche d'homomorphismes de graphes, nous considérons une approche générale similaire à celle introduite par l'algorithme de *Messmer & Bunk* décrit dans le chapitre relatif à l'état de l'art. Le choix de se baser sur une approche similaire à cet algorithme (au lieu d'un algorithme de type *Ullmann*) est motivé par le fait qu'elle se révèle très efficace dans le cas où la recherche d'homomorphismes implique plusieurs graphes similaires. Notre problématique se situe dans ce contexte puisque le modèle des règles de réécriture implique de trouver les homomorphismes pour les graphes $L \cup K$ et $L \cup K \cup N$ qui sont similaires par construction.

À chaque itération de l'algorithme, nous calculons tous les homomorphismes d'un sous graphe *MG'* du graphe modèle *MG* vers le graphe d'entrée *IG*. *MG'* contient initialement un nœud unique et l'algorithme progresse en intégrant, à chaque itération, un nouveau nœud du graphe modèle et les arcs reliant ce nœud et les nœuds appartenant à *MG'*.

TEST_VERTEX (n_M, MG, IG)

0- Let $MG = (N_M, A_M, Lab_M, D_{LabM}, Etiq_M, D_{EtiqM})$ and
$IG = (N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$ and let F=$\emptyset$
1- For all $n_I \in N_I$
  a. Val= UNIFY_VERTEX(n_M, n_I)
  b. if $Val \neq NULL$ then
  c. if $[(n_M, n_M) \notin A_M]$[44]
    or if $[(n_M, n_M) \in A_M$ and $(n_I, n_I) \in A_I$ and $Etiq_I((n_I, n_I)) = Etiq_M((n_M, n_M))]$[45] then
  d. set $f(n_M) = n_I$; $F = F \cup (f, Val)$
2- return (F)

FIG. 4.4 – Pseudo-code de la procédure TEST_VERTEX.

Grâce au principe domino, l'algorithme s'arrête par un échec si, à n'importe quelle itération, il ne peut intégrer un nouveau nœud. Il s'arrête avec succès s'il réussit à intégrer tous les nœuds et les arcs

[44] Il n'y a pas de boucle qui relie n_M à lui même.
[45] Il existe une boucle reliant n_M à lui même et une boucle reliant n_I à lui même et les deux boucles ont le même label.

de *MG*. En intégrant un nouveau nœud, l'algorithme vérifie, en plus de la préservation de la structure, que les affectations des variables sont consistantes et qu'elles peuvent être combinées avec les affectations obtenues pour les variables de *MG'*.

La procédure TEST_VERTEX permet de calculer toutes les unifications possibles d'un nœud n_M appartenant au graphe modèle *MG* avec tous les nœuds n_I du graphe d'entrée *IG*. La description algorithmique de cette procédure est présentée dans la figure 4.4. Elle retourne l'ensemble vide si n_M n'est unifiable avec aucun nœud de *IG*.

L'intégration d'un nouveau nœud est implémentée par la procédure ADD_VERTEX. La description algorithmique de cette procédure est donnée dans la figure 4.5. Cette procédure prend en entrée : 1) *IG* le graphe d'entrée, 2) *MG'* un sous graphe du graphe modèle, 3) n_M le nœud du graphe modèle à intégrer dans *MG'* (avec $n_M \notin MG'$), 4) F'_M l'ensemble de tous les homomorphismes de *MG'* vers *IG*, 5) F l'ensemble de toutes les unifications de n_M avec les nœuds de *IG*, et 6) A_M l'ensemble des arcs du graphe modèle. Cette procédure retourne l'ensemble de tous les homomorphismes de *MG'* augmenté du nœud n_M vers le graphe *IG*. Elle retourne l'ensemble vide si l'intégration du nœud n_M échoue.

ADD_VERTEX$(IG, MG', n_M, F'_M, F, A_M)$

0- Let $MG' = (N'_M, A'_M, Lab'_M, D'_{LabM}, Etiq_M, D'_{EtiqM})$ and
let $IG = (N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$, and let $F_{res}=\emptyset$
1- For every pair f_1, f_2 and Val_1, Val_2 where $(f_1, Val_1) \in F'_M$ and $(f_2, Val_2) \in F$
  a. Test the conditions :
    i. $f_1(N_M) \cap f_2(\{n_M\}) = \emptyset$[46]
    ii. For each edge $e_1 = (n'_M, n_M) \in A_M$ such that $n'_M \in N'_M$,
      there exists an edge $e'_1 = (f_1(n'_M), f_2(n_M)) \in A_I$ such that $Etiq_M(e_1) = Etiq_I(e'_1)$
    iii. For each edge $e_2 = (n_M, n'_M) \in A_M$ such that $n'_M \in N'_M$,
      there exists an edge $e'_2 = (f_2(n_M), f_1(n'_M)) \in A_I$) such that $Etiq_M(e_2) = Etiq_I(e'_2))$
  b. if a.i. and a.ii. and a.iii. hold then
    i. MERGE$(Val_1, Val_2) = Val_{I,M}$
    ii. if $Val_{I,M} \neq NULL$ then build $f \subseteq (N'_M \cup \{n_M\}) \times N_I$ such that,
      $f(n) = f_1(n)$ if $n \in N'_M$ and $f(n_M) = f_2(n_M)$
    iii. $F_{res} = F_{res} \cup \{(f, Val_{I,M})\}$
2- Return(F_{res})

FIG. 4.5 – Pseudo-code de la procédure ADD_VERTEX.

L'approche générale de l'algorithme de recherche d'homomorphismes, telle que cela est illustré dans la figure 4.6, démarre par un sous graphe du graphe modèle contenant un nœud unique choisi aléatoirement (i.e. nœud n1). À chaque itération, un nouveau nœud est intégré (n2 puis n3) en produisant des homomorphismes partiels. Les homomorphismes partiels de l'interaction i sont générés en combinant les homomorphismes partiels de l'itération $i-1$ avec les unifications possibles du nœud à intégrer. Le même procédé est reproduit jusqu'à intégrer l'ensemble des nœuds du graphe modèle, auquel cas nous obtenons tous les homomorphismes possibles.

[46] i.e. n_M n'est pas unifié, par f_2, avec un nœud déjà unifié, avec f_1, avec un nœud appartenant à N_M. Autrement, deux nœuds différents seront unifiés avec le même nœud alors qu'un homomorphisme est, par définition, une application injective.

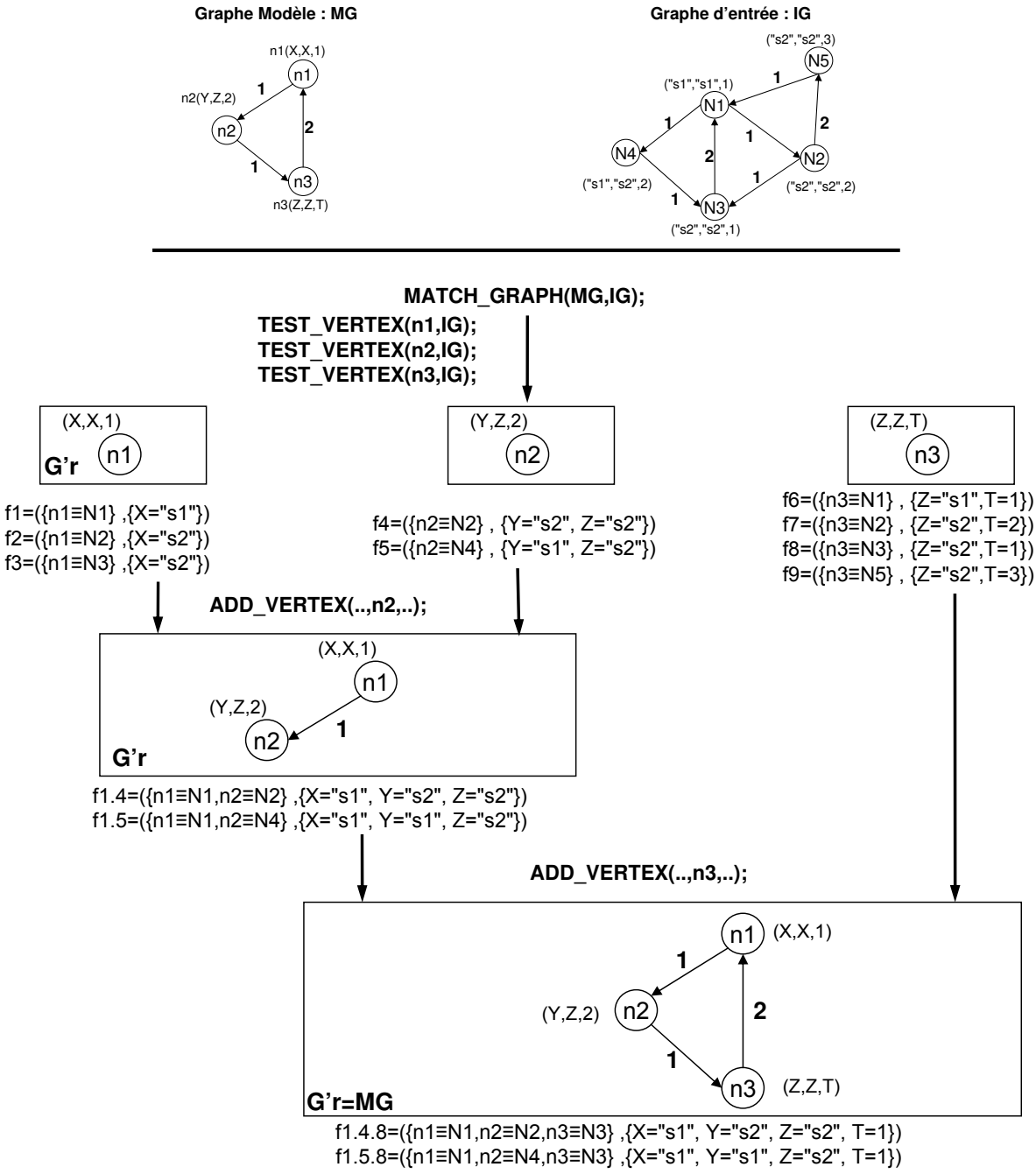


FIG. 4.6 – Exemple illustrant l'approche globale pour la recherche d'homomorphismes

Le pseudo-code de la procédure MATCH_GRAPH (figure 4.7) reprend cette approche en s'appuyant sur les différentes procédures définies précédemment. Cette procédure prend en entrée : 1) un graphe modèle MG et 2) un graphe d'entrée IG. Elle retourne l'ensemble vide s'il n'existe pas d'homomorphismes de MG vers IG.

MATCH_GRAPH (IG, MG)

0- Let $MG = (N_M, A_M, Lab_M, D_{LabM}, Etiq_M, D_{EtiqM})$ and let
IG= $(N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$. Let $n = Card(N_M)$ and $m = Card(N_I)$ and $F_{res} = \emptyset$
1- If $n > m$ then return($\emptyset$)
2- For all $i \in [0..n-1]$
  a. Let $n_{Mi} = N_M[i]$ and let F_i=TEST_VERTEX(n_{Mi}, IG)
  b. if $F_i = \emptyset$ then return($\emptyset$)
3- Let $MG' = \{n_{M0}\}$ and let $F_{res} = F_0$
4- For all $i \in [1..n-1]$
  a. F_{res}=ADD_VERTEX($IG, MG', n_{Mi}, F_{res}, F_i, E_M$)
  b. if $F_{res} = \emptyset$ then return($\emptyset$)
  c. $MG' = MG' \cup \{n_{Mi}\}$
  d. $i = i + 1$
5- Return(F_{res})

FIG. 4.7 – Pseudo-code de l'algorithme de recherche d'homomorphismes

4.3.2 Application à la transformation de graphes

Nous nous basons sur l'algorithme de recherche d'homomorphismes de la section précédente pour définir un algorithme de transformation de graphes. Avant cela, nous reprenons les définitions des règles de réécriture et de leur structure. Nous considérons le cas le plus générique de notre modèle en prenant en compte des conditions d'application négatives et des instructions de connexion.

Dans le cas des grammaires de graphes définies pour la description des styles architecturaux, notre algorithme reste valable puisque les productions de grammaires considérées peuvent être vues comme un cas particulier de ce modèle. Dans ce cas particulier, N et C correspondent respectivement au graphe vide et à l'ensemble vide.

***Définition* 18** *Une règle de réécriture est un quintuplet $RR = (L, K, R, N, C)$ tel que L et R sont deux graphes et K un sous graphe de R et de L. N est un graphe exprimant la condition d'application négative et C un ensemble d'instructions de connexion de type edNCE et de la forme $(n, l, etiq1/etiq2, d, d')$.*
RR est applicable à un graphe hôte G s'il existe un homomorphisme (tel que cela est défini dans la section précédente) $h : L \rightarrow G$ et s'il n'existe pas d'homomorphismes $h' : N \cup L \rightarrow G$ tels que $h'(n) = h(n), \forall n \in L$. Son application implique : 1) la suppression du sous graphe $h(L \setminus K)$, 2) la suppression des arcs suspendus (approche SPO), 3) l'introduction d'une copie homomorphe à $R \setminus K$ qui intègre les affectations obtenues pour h, puis 4) l'exécution de toutes les instructions de connexion appartenant à l'ensemble C.

Afin de vérifier l'applicabilité d'une règle RR à un graphe hôte HG, implémentée par la procédure MATCH_RULE (figure 4.8), nous commençons par rechercher l'ensemble des homomorphismes F de L vers HG (c.f. ligne 1). S'il n'y a pas d'homomorphismes, alors RR n'est pas applicable (ligne 2). Sinon, nous procédons en intégrant à chaque itération un nouveau nœud de $N \setminus L$ jusqu'à : ou bien 1) détecter l'absence de tout homomorphisme entre $L \cup N$ et HG (ligne 4.b) ou, 2) atteindre $L \cup N$ et trouver l'ensemble des homomorphismes possibles $F_{N \cup L}$ de $L \cup N$ vers HG (boucle de la ligne 6). RR est applicable à HG si et seulement si l'une des deux conditions suivantes est valide :
1- Il n'y a pas d'homomorphismes entre $L \cup N$ vers HG (lignes 3 ou 6-b).
2- Il existe un homomorphisme $H \in F$ tel que pour tous les homomorphismes $H' \in F'_{N \cup L}$, H n'est pas une projection de H' sur le domaine d'entrée L.

Le résultat de la procédure d'applicabilité de la règle RR sur le graphe HG est $F \setminus (F_{L \cup N} \setminus_L)$ (ligne 7-a) avec $F_{L \cup N} \setminus_L$ est la fonction $F_{L \cup N}$ projetée sur le domaine de définition L, et $\setminus$ correspondant à l'opérateur de soustraction pour les ensembles. La procédure MATCH_RULE retourne l'ensemble vide si RR n'est pas applicable à HG.

MATCH_RULE (RR, HG)

0- Let $RR = (L, K, R, N, C)$, let $N \setminus L = (N_{N\setminus L}, A_{N\setminus L}, Lab_{N\setminus L}, D_{LabN\setminus L}, Etiq_{N\setminus L}, D_{EtiqN\setminus L})$
Let $L \cup N = (N_{L\cup N}, A_{L\cup N}, Lab_{L\cup N}, D_{LabL\cup N}, Etiq_{L\cup N}, D_{EtiqL\cup N})$,
let $HG = (N_H, A_H, Lab_H, D_{LabH}, Etiq_H, D_{EtiqH})$, let $F_{res} = \emptyset$
Let $n = Card(N_{L\cup N})$, $m = Card(N_H)$, $k = Card(N_{N\setminus L})$
1- $F =$ MATCH_GRAPH(L, HG)
2- if $F = \emptyset$ then return$(\emptyset)$
3- if $n > m$ then return(F)
4- For all $i \in [0..k-1]$
  a. Let $n_{N\setminus L,i} = N_{N\setminus L}[i]$ and let $F_i =$TEST_VERTICES$(n_{N\setminus L,i}, HG)$
  b. if $F_i = \emptyset$ then return(F)
5- Let $RG' = L$, let $F_{N\cup L} = F$
6- For all $i \in [0..k-1]$
  a. $F_{N\cup L} =$ADD_VERTEX$(HG, RG', n_{N\setminus L,i}, F_{N\cup L}, F_i, A_{L\cup N})$
  b. if $F_{N\cup L} = \emptyset$ then return(F)
  c. $RG' = RG' \cup \{n_{N\setminus L,i}\}$
7- For all f_1, f_2 and val_1, val_2 such that, $val_1 \in val_2$, $(f_1, val_1) \in F$ and $(f_2, val_2) \in F_{N\cup L}$
  a. If for all $n \in L$, $f_1(n) = f_2(n)$ then $F = F \diagdown \{(f_1, val_1)\}$
8- return(F)

FIG. 4.8 – Pseudo-code de la procédure MATCH_RULE.

La considération des variables dans les règles de réécriture implique la prise en compte des différentes affectations obtenues par l'étape de recherche d'homomorphismes. En effet, l'utilisation de variables liées dans la partie gauche et dans la partie droite d'une règle (i.e. une même variable apparaissant dans les labels d'un nœud de L et dans les labels d'un nœud de $R \setminus K$) doit avoir comme conséquence la prise en compte de l'affectation de cette variable. La procédure VALUATE_VERTEX permet, pour un nœud n donné, d'affecter les labels variables selon un ensemble d'affectation Val.

VALUATE_VERTEX (n, Val, G)

0- let $G = (N, A, Lab, D_{Lab}, Etiq, D_{Etiq})$, let $Lab(n) = (l_1, \ldots, l_n)$,
  Let $L = Lab(n)$
1- For all $i = 1, \ldots, n$
  a. If l_i is variable and $(l_i, val) \in Val$, then
    i. $L[i] = val$[47]
6- Return(L)

FIG. 4.9 – Pseudo-code de la procédure VALUATE_VERTEX.

[47] Ceci signifie que si $L = (l_1, \ldots, l_i, \ldots, l_n)$ alors L est mis-à-jour à $(l_1, \ldots, val, \ldots, l_n)$.

APPLY_RULE (RR, HG)

0- let $RR = (L, K, R, N, C)$, let $HG = (N_H, A_H, Lab_H, D_{LabH}, Etiq_H, D_{EtiqH})$
Let $HG' = HG$ and let note $HG' = (N'_H, A'_H, Lab'_H, D'_{LabH}, Etiq'_H, D'_{EtiqH})$ and
F =MATCH_RULE(RG, HG)
1- if $F = \emptyset$ then return(FALSE)
2- Let $(f, val) \in F$
3- For all vertices $n_L \in (L \setminus K)$
a. For all edges $e \in A'_H$ such that $e = (f(n_L), n)$ or $e = (n, f(n_L))$ with $n \in HG'$
i. $A'_H = A'_H \setminus \{e\}$
b. $N'_H = N'_H \setminus \{f(n_L)\}$
4- For all vertices $n_R \in R \setminus K$, let $n_{R,H}$ a new vertex
a. let L=VALUATE_VERTEX$(n_R, val, R \setminus K)$
b. $N'_H = N'_H \cup \{n_{R,H}\}$
c. $Lab'_H = Lab'_H \cup (n_{R,H}, L)$
d. $f = f \cup (n_R, n_{R,H})$ [48]
5- For all edges $e = (n_{R1}, n_{R2}) \in A_{R \setminus K}$
a. Let $e' = (f(n_{R1}), f(n_{R2}))$
b. $A'_H = A'_H \cup e'$ and $Etiq'_H = Etiq'_H \cup (e', Etiq_{R \setminus K}((n_{R1}, n_{R2})))$
6- For all $c_i = (n, l_1, etiq_1/etiq_2, d, d')$ with $n \in R$
a. if $d = in$ then
i. For all $n_H \in N_H$ and $n_L \in L \setminus K$ such that $(n_H, f(n_L)) \in A_H$ and $Etiq_H((n_H, f(n_L))) = etiq1$
a. if $d' = in$ then
i. $A'_H = A'_H \cup (n_H, f(n))$
ii. $Etiq'_H = Etiq'_H \cup ((n_H, f(n)), etiq_2)$
a. else
i. $A'_H = A'_H \cup (f(n), n_H)$
ii. $Etiq'_H = Etiq'_H \cup ((f(n), n_H), etiq_2)$
b. if $d = out$ then
i. For all $n_H \in N_H$ and $n_L \in L \setminus K$ such that $(f(n_L), n_H) \in A_H$ and $Etiq_H((f(n_L), n_H)) = etiq1$
a. if $d' = in$ then
i. $A'_H = A'_H \cup (n_H, f(n))$
ii. $Etiq'_H = Etiq'_H \cup ((n_H, f(n)), etiq_2)$
a. else
i. $A'_H = A'_H \cup (f(n), n_H)$
ii. $Etiq'_H = Etiq'_H \cup ((f(n), n_H), etiq_2)$
7- $HG = HG'$
8- return(TRUE)

FIG. 4.10 – Pseudo-code de la procédure d'application des règles de réécriture

L'application d'une règle de réécriture est implémentée par la procédure APPLY_RULE. La description algorithmique de cette procédure est donnée dans la figure 4.10. Cette procédure retourne "TRUE" dans le cas où une transformation du graphe a effectivement eu lieu, et "FALSE" sinon.

De manière informelle, la procédure d'application d'une règle RG sur un graphe HG respecte les cinq étapes suivantes :
1- Choisir un homomorphisme $H = (f, val)$ parmi ceux qui ont été construits par la procédure précédente (ligne 2). Ici ce choix est aléatoire, mais il est possible d'introduire des heuristiques permettant d'effectuer un choix motivé par des objectifs d'optimisation.
2- Pour tous les nœuds n de HG tels qu'il existe un nœud n_L avec n_L appartenant à $L \setminus K$ et $f(n_L) = n$, supprimer n (ligne 3-b.) et tous les arcs qui le lient avec les autres nœuds de HG (ligne 3-a.i.).

[48] Cette affectation est nécessaire pour garder une trace de l'image des nœuds introduit afin de permettre le traitement et l'ajout des arcs appartenant à $R \setminus K$

3- Pour tous les nœuds $n_R \in R \setminus K$, introduire des copies instanciées $n_{R,H}$ (ligne 4-b). Les variables appartenant aux labels de n_R sont remplacées par les affections assignées par val (ligne 4-a).
4- Introduire un arc $e = (n_1, n_2)$ entre chaque paire de nœuds n_1 et n_2 appartenant au graphe HG' résultant des étapes 1 à 3. S'il existe deux nœuds n_{R1} et n_{R2} connectés par un arc $e = (n_{R1}, n_{R2})$ dans la règle et tels que $(n_{R1}, n_1) \in f$ et $(n_{R2}, n_2) \in f$, et $e \in A_{R\setminus K}$. Cet arc est étiqueté par la même étiquette que e (ligne 5-b.).
5- Exécuter toutes les instructions de connections[49] (boucle de la ligne 6).

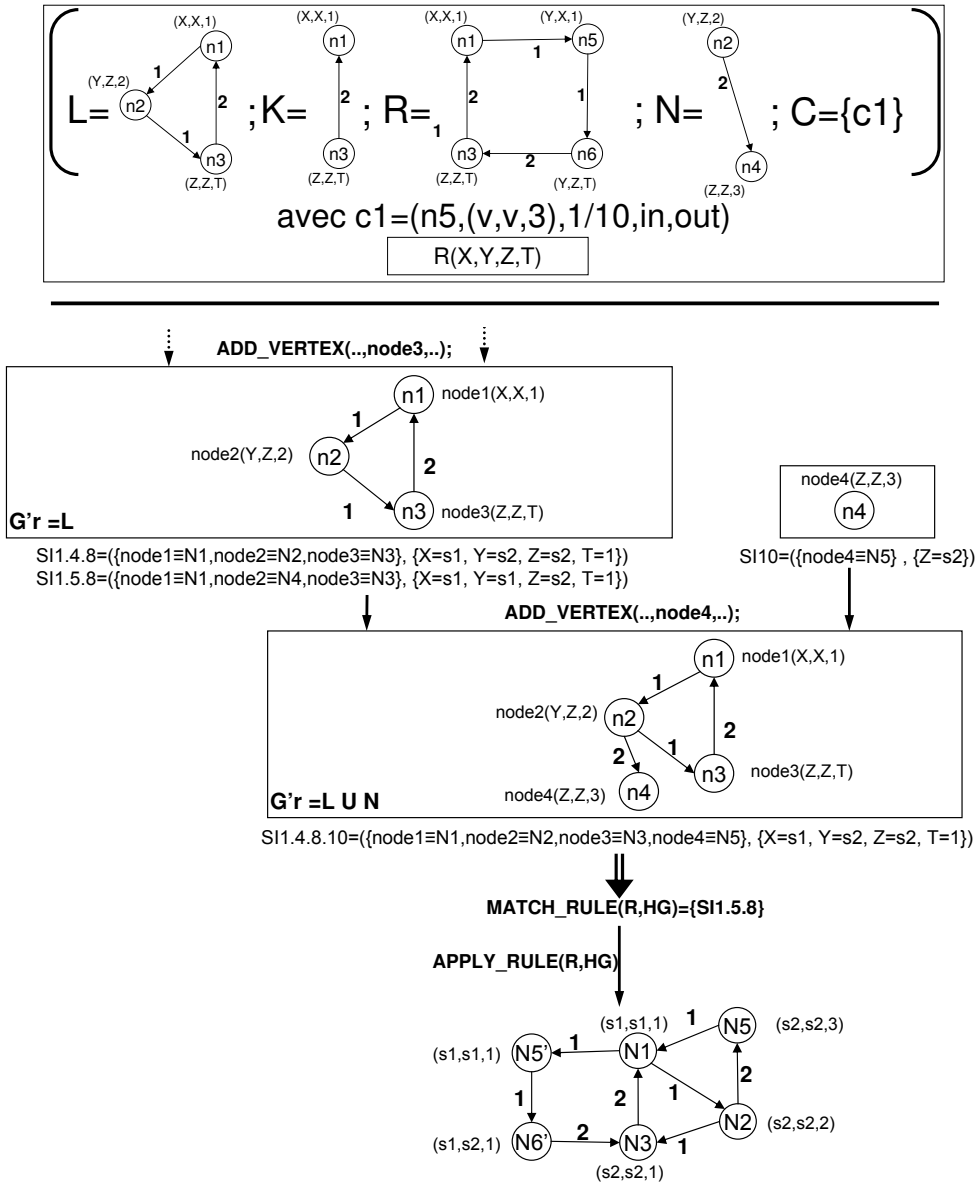


FIG. 4.11 – Exemple de la transformation de graphes avec la procédure APPLY_RULE

La figure 4.11 présente l'exploitation de la recherche d'homomorphismes obtenus dans la figure 4.6 pour l'application d'une règle de transformation appartenant au modèle considéré par nos approches.

[49] L'utilisation du graphe temporaire HG' dans la procédure est importante pour ne pas écraser les arcs appartenant à $L \setminus K$ et permettre de prendre correctement en compte les instructions de connexion.

4.4 Analyse de complexité de l'algorithme générique

Dans cette section, nous présentons une analyse de complexité pour l'algorithme générique. Nous focaliserons cette analyse sur l'algorithme de recherche d'homomorphismes. Nous considérons que la complexité de l'algorithme de transformation de graphes est équivalente à son étape préliminaire de recherche d'homomorphismes. Cette assertion est fondée dans la mesure où les actions de transformation du graphe (i.e. suppression des nœuds dans $L \setminus K$, ajout des nœuds de $R \setminus K$ ... etc) impliquent une très faible complexité.

Nous considérons l'algorithme de recherche d'homomorphisme et nous prenons en compte les deux paramètres suivants :
1- N_{MG} : le nombre de nœuds dans le graphe modèle MG.
2- N_{IG} : le nombre de nœuds dans le graphe d'entrée IG.

Nous étudions la complexité de l'algorithme pour le meilleur et le pire cas. Dans le meilleur cas, les entrées de l'algorithme se situent dans un contexte favorable impliquant un minimum de calcul. Dans le pire cas, l'algorithme réalise un nombre maximum d'opérations. Nous allons considérer, en premier, le cas où les nœuds du graphe modèle sont labélés exclusivement avec des constantes. Ensuite, nous étudierons la complexité en intégrant le nombre de paramètres dans les labels des nœuds. Les instructions que nous considérons sont celles qui correspondent à des boucles de calcul. Nous assumons que les calculs dus aux autres instructions ne sont pas significatifs et sont négligeables dans le cas des graphes de grande taille.

4.4.1 Analyse de complexité avec labels constants

Le meilleur cas

Le meilleur cas correspond à la situation où chaque nœud du graphe modèle est labélé par une liste unique et correspond à un et un seul nœud dans le graphe d'entrée. Dans ce cas, la ligne (2) de la procédure MATCH_GRAPH est appelée N_{MG} fois pour exécuter la procédure TEST_VERTEX. À chaque appel, TEST_VERTEX exécute la ligne (1) N_{IG} fois et renvoie à chaque fois une ensemble d'homomorphismes F_i correspondant à un singleton. Ainsi, la première étape de l'algorithme possède une complexité de $N_{MG} \times N_{IG}$.

Pour la ligne (4) de la procédure MATCH_GRAPH, nous allons toujours obtenir un ensemble F_{res} contenant un unique homomorphisme et F_i qui contient aussi un seul homomorphisme. Alors, la ligne (1) de la procédure ADD_VERTEX va être exécutée une fois à chaque intégration d'un nouveau nœud. Par conséquent, nous comptons $(N_{MG} - 1)$ exécutions jusqu'à l'intégration des $(N_{MG} - 1)$ nœuds qui restent avant d'atteindre MG. La complexité de l'algorithme, dans le meilleur cas, est donc égale à :

$$N_{MG} - 1 + N_{MG} \times N_{IG} = O(N_{MG} \times N_{IG})$$

Le pire cas

Le pire cas correspond à la situation où tous les nœuds du graphe modèle et ceux du graphe d'entrée sont, deux à deux, unifiables et que ces deux graphes sont des graphes complets [50] et que tous leurs arcs sont labélés par le même label.

La complexité introduite par l'exécution de la ligne (2) de la procédure MATCH_GRAPH et la ligne (1) de la procédure TEST_VERTEX reste la même que dans le meilleur cas : $N_{MG} \times N_{IG}$. Cependant, le résultat n'est pas le même puisque chaque nœud de MG peut être unifié avec les N_{IG} nœuds du graphe IG. L'ensemble des homomorphismes possibles F_i contient, par conséquent, N_{IG} homomorphismes.
Concernant la ligne (4) de la procédure MATCH_GRAPH, après l'itération i, nous obtenons F_{res} contenant un ensemble d'homomorphismes de cardinalité égale à $A^i_{N_{IG}}$, le nombre des différents arrangements de i dans N_{IG}. Par conséquent, la ligne (1) de la procédure ADD_VERTEX sera exécutée, à chaque étape i, (avec $1 \leq i \leq N_{MG} - 1$), $(N_{IG} \times A^i_{N_{IG}})$ fois. La complexité de l'algorithme est, donc, de

$$N_{MG} \times N_{IG} + N_{IG} \times \sum_{i=1}^{N_{MG}-1} A^i_{N_{IG}}$$

[50] Un graphe complet est un graphe dont tous les nœuds sont tous reliés deux à deux par un arc.

***Proposition* 1** *La complexité de l'algorithme dans le cas de graphes avec des labels constants et avec* $N_{MG} \ll N_{IG}$[51] *est égale dans le pire cas à*

$$\boldsymbol{O(\sqrt{\frac{N_{IG}}{N_{IG}-N_{MG}}} \times \frac{1}{N_{IG}-N_{MG}} \times N_{IG}^{N_{MG}+1})}$$

***Lemme* 1** *Quand* m *et* n *tendent vers l'infini avec* $n \ll m$*, nous avons*

$$\boldsymbol{\sum_{i=1}^{n} A_m^i \simeq \frac{1}{m-n}\sqrt{\frac{m}{m-n}} m^{n+1}}$$

La preuve du Lemme 1 est donnée dans l'annexe A.

***Preuve* 1 (Proposition 1)** *En partant de la complexité suivante :*

$$C1 = N_{MG} \times N_{IG} + N_{IG} \times \sum_{i=1}^{N_{MG}-1} A_{N_{IG}}^i$$

nous déduisons, par le lemme 1, que :

$$C1 \quad = \quad O(N_{MG} \times N_{IG} + N_{IG} \times \sqrt{\frac{N_{IG}}{N_{IG}-N_{MG}-1}} \times \frac{1}{N_{IG}-N_{MG}-1} \times N_{IG}^{N_{MG}+1-1})$$

Puisque

$$(N_{IG}-N_{MG}-1) \simeq (N_{IG}-N_{MG})$$

et $(N_{MG} \times N_{IG})$ *est négligeable par rapport à*

$$(\sqrt{\frac{N_{IG}}{N_{IG}-N_{MG}}} \times \frac{1}{N_{IG}-N_{MG}} \times N_{IG}^{N_{MG}+1})$$

nous obtenons :

$$C1 = O(\sqrt{\frac{N_{IG}}{N_{IG}-N_{MG}}} \times \frac{1}{N_{IG}-N_{MG}} \times N_{IG}^{N_{MG}+1})$$

□

4.4.2 Complexité dans le pire cas avec labels variables

La considération de labels variables implique des calculs supplémentaires par l'exécution de la procédure MERGE et plus spécifiquement pour la boucle de sa ligne (1). En plus des paramètres N_{MG} et N_{IG}, nous introduisons un nouveau paramètre K correspondant au nombre de labels dans les nœuds du graphe modèle. Le pire cas correspond à la situation où il existe, dans le graphe modèle, autant de variables différentes que de labels pour chaque nœud. Ceci implique que le nombre de variables est égal à $N_{MG} \times K$.

Nous commençons, ici, par calculer la complexité de l'étape d'unification de deux nœuds et ainsi la complexité de la procédure UNIFY_VERTICES en prenant en compte sa ligne (3). Nous obtenons, après chaque itération i de la ligne (3) (avec i correspondant au nombre de labels traités dans chaque nœud), l'ensemble des affectations Val qui contient $i-1$ affectations ($i-1$ variables traitées) et V contenant une affectation unique (puisque tous les labels $l_{1,i}$ sont des variables). Ainsi, la complexité introduite par la

[51] Sans cette supposition nous obtenons, au lieu d'un équivalent, une borne max de la complexité de l'algorithme.

procédure MERGE(Val, V) correspond à $(i-1) \times 1$ exécutions de la ligne (1.a). Et par conséquent, la complexité introduite par la procédure UNIFY_VERTICES est, dans ce cas, égale à :

$$\sum_{i=1}^{K} i - 1 = \sum_{i=0}^{K-1} i = \frac{(K-1) \times K}{2} = O(\frac{K^2}{2})$$

Pour la procédure MATCH_GRAPH, la complexité introduite par la ligne (2), dans l'itération i, est équivalente à la complexité générée par la procédure TEST_VERTEX. Cette complexité est équivalente à N_{IG} fois la complexité de la procédure UNIFY_VERTICES puisqu'à chaque exécution de TEST_VERTEX, il y a un appel de UNIFY_VERTICES pour chaque nœud du graphe IG. Ainsi, la complexité introduite par la ligne (2.a) est égale à $N_{IG} \times O(\frac{K^2}{2})$. Et par la suite, si nous considérons toutes les itérations de $i = 0$ à $N_{MG} - 1$, nous obtenons la complexité suivante générée par la première partie de l'algorithme qui correspond à la ligne (2) de MATCH_GRAPH :

$$O(N_{MG} \times N_{IG} \times \frac{K^2}{2})$$

Pour la deuxième partie de l'algorithme correspondant à la ligne (4) de la procédure MATCH_GRAPH, nous savons que, pour le pire cas avec des labels constants, les ensembles F_i sont de cardinal N_{IG}. Nous avons aussi, après chaque itération i de cette ligne (4), F_{res} est de cardinal $A_i^{N_{IG}}$. Le second terme de tous les homomorphismes de F_i (correspondant aux affectations) contient K affectations et le second terme de tous les homomorphismes F_{res} contient $i \times K$ affectations. Notons aussi que, pour chaque itération i, la complexité introduite par la ligne (4) est équivalente à la complexité introduite par la procédure ADD_VERTEX. Dans ce cas, pour ADD_VERTEX, il existe $A_i^{N_{IG}} \times N_{IG}$ paires (f_1, f_2), et pour chaque paire, la ligne (1.b.i) (i.e. MERGE($f_1(N_r), f_2(n_r)$) introduit une complexité de $(i \times K) \times K$ correspondant au nombre d'affectations de f_1 multiplié par le nombre d'affectation de f_2. Ainsi, la complexité introduite par chaque itération i est $A_i^{N_{IG}} \times N_{IG} \times i \times K^2$. La complexité introduite par la ligne (4) est, donc, égale à :

$$N_{IG} \times K^2 \times \sum_{i=1}^{N_{MG}-1} i \times A_i^{N_{IG}}$$

***Proposition* 2** *Dans le pire cas, la complexité de l'algorithme de recherche d'homomorphismes en considérant des labels variables et $N_{MG} \ll N_{IG}$ est égale à :*

$$\boldsymbol{O(K^2 \times \sqrt{\frac{N_{IG}}{N_{IG} - N_{MG}}} \times (\frac{N_{MG}}{N_{IG} - N_{MG}}) N_{IG}^{N_{MG}+1})}$$

***Lemme* 2** *Si m et n tendent vers l'infini avec $n \ll m$, nous avons*

$$\boldsymbol{\sum_{i=0}^{n} i A_m^i \simeq \sqrt{\frac{m}{m-n}} \times (\frac{n}{m-n}) m^{n+1}}$$

Pour ne pas surcharger ce chapitre, la preuve du lemme 2 est aussi donnée dans l'annexe A du manuscrit.

***Preuve* 2 (Proposition 2)** *La complexité de l'algorithme est égale à :*

$$C2 = N_{MG} \times N_{IG} \times O(\frac{K^2}{2}) + N_{IG} \times K^2 \times \sum_{i=1}^{N_{MG}-1} i \times A_{N_{IG}}^i$$

En utilisant le lemme 2, nous obtenons :

$$C2 \quad = \quad N_{MG} \times N_{IG} \times O(\frac{K^2}{2}) + N_{IG} \times K^2 \times \sqrt{\frac{N_{IG}}{N_{IG} - N_{MG} - 1}} \times (\frac{N_{MG} - 1}{N_{IG} - N_{MG} - 1}) \times N_{IG}^{N_{MG}-1+1}$$

Puisque

$$(N_{IG} - N_{MG} - 1) \simeq (N_{IG} - N_{MG})$$

et $(N_{MG} \times N_{IG} \times \frac{K^2}{2})$ *est négligeable par rapport à*

$$(K^2 \times \sqrt{\frac{N_{IG}}{N_{IG} - N_{MG}}} \times \frac{N_{MG}}{N_{IG} - N_{MG}} \times N_{IG}^{N_{MG}+1})$$

nous obtenons :

$$C2 = O(K^2 \times \sqrt{\frac{N_{IG}}{N_{IG} - N_{MG}}}(\frac{N_{MG}}{N_{IG} - N_{MG}})N_{IG}^{N_{MG}+1})$$

□

4.5 Mesures expérimentales

4.5.1 Génération des entrées pour les expérimentations

Nous avons implémenté notre algorithme de recherche d'homomorphismes en *C++* et nous avons réalisé un nombre important de jeux de test pour obtenir des mesures expérimentales concernant le temps d'exécution. La génération des entrées de l'algorithme impliquant la génération du graphe d'entrée et du graphe modèle est réalisée de manière aléatoire en se basant sur une génération probabiliste uniforme et en considérant les paramètres définis ci-dessous. Pour ces expérimentations, nous avons considéré deux types de labels correspondant aux entiers et aux chaînes de caractères.

- n_M : Le nombre de nœuds dans le graphe modèle.
- n_G : Le nombre de nœuds dans le graphe d'entrée.
- $n_{i,M}$: Le nombre des valeurs possibles pour un label de type entier d'un nœud du graphe modèle. En utilisant une génération uniformément distribuée, pour chaque nœud, la probabilité d'obtenir la valeur i (avec $0 \leq i \leq n_{i,M} - 1$) pour un label est égale à $\frac{1}{n_{i,M}}$.
- $n_{s,M}$: Le nombre des valeurs possibles pour un label de type chaîne de caractères appartenant à un nœud du graphe modèle.
- $n_{i,G}$ et $n_{s,G}$: deux paramètres associés au graphe d'entrée respectivement équivalents à $n_{i,M}$ et $n_{s,M}$.
- $Prob_M$: La probabilité pour que deux nœuds du graphe modèle soient connectés.
- $Prob_I$: La probabilité pour que deux nœuds du graphe d'entrée soient connectés.

Afin d'éviter de prendre en compte des temps d'exécution non significatifs, nous rejetons toutes les expériences qui génèrent des entrées impliquant une absence d'homomorphismes.

4.5.2 Résultats expérimentaux

Influence du nombre de nœuds du graphe d'entrée

Afin de mesurer l'influence du nombre de nœuds dans le graphe d'entrée sur le temps d'exécution, nous faisons varier le paramètres n_G tout en fixant les autres paramètres avec $prob_M = \frac{1}{3}$, $prob_I = \frac{1}{10}$, $n_{i,M} = n_{s,M} = 3$, et $n_{i,G} = n_{s,G} = 5$.

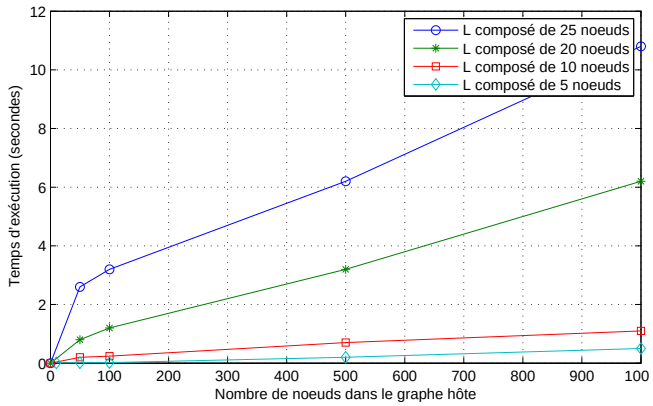


FIG. 4.12 – Influence de la taille du graphe hôte sur le temps d'exécution

Dans la figure 4.12, nous présentons les résultats moyens obtenus pour des graphes d'entrée contenant respectivement 5, 10, 20, et 25 nœuds. Les mesures obtenues montrent un temps d'exécution qui varie de manière polynômiale en fonction de la taille du graphe d'entrée.

Influence du nombre de nœuds dans le graphe modèle

Pour mesurer l'influence du nombre de nœuds dans le graphe modèle sur le temps d'exécution, nous considérons les valeurs suivantes $prob_M = \frac{1}{3}$, $prob_I = \frac{1}{5}$, $n_{i,M} = n_{s,M} = 3$, et $n_{i,G} = n_{s,G} = 5$ pour les entrées des expériences.

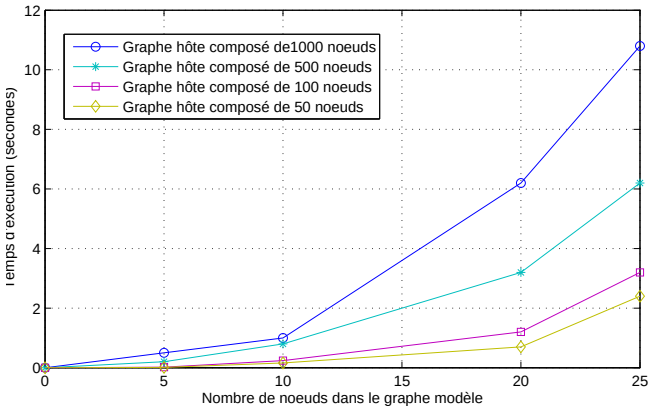


FIG. 4.13 – Influence de la taille du graphe modèle sur le temps d'exécution

Les temps d'exécution moyens, présentés dans la figure 4.13 et obtenus pour un graphe hôte HG comprenant respectivement 50, 100, 500 et 1000 nœuds, confirme que les temps d'exécution varient de manière exponentielle en fonction de la taille du graphe modèle[52].

Influence du nombre d'arcs

Dans l'objectif de mesurer l'influence du nombre d'arc (dans le graphe modèle et le graphe hôte) sur le temps d'exécution de l'algorithme, nous considérons deux ensembles de jeux d'expérimentation où les

[52] Ce qui était prévisible puisque le problème est NP-Complet.

paramètres n_M et n_G sont fixés aux valeurs $n_M = 10$ et $n_G = 500$.

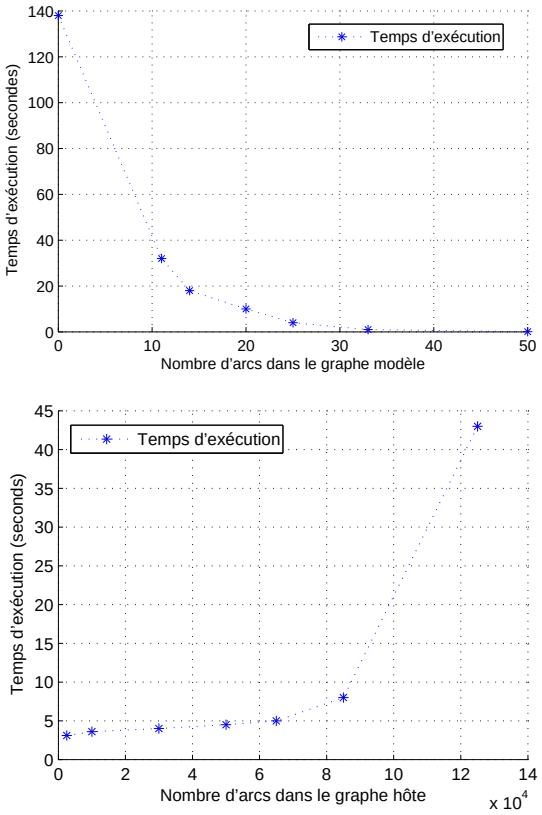


FIG. 4.14 – Influence du nombre d'arcs dans le graphe modèle / Influence du nombre d'arcs dans le graphe hôte

Nous pouvons conclure, à partir des résultats présentés dans la figure 4.14, que les temps d'exécution sont inversement proportionnels (de manière exponentielle) au nombre d'arcs dans le graphe modèle. Ceci était prévisible puisque, pour un graphe hôte donné HG, l'augmentation des arcs du graphe modèle diminue le nombre d'homomorphismes entre les sous graphes intermédiaires et le graphe hôte[53].

D'un autre côté, le temps d'exécution varie de manière exponentielle en fonction du nombre d'arc dans le graphe d'entrée (figure 4.14). Ceci est explicable par le fait que plus le graphe d'entrée est connecté, plus il y aura d'homomorphismes intermédiaires avec un graphe modèle donné.

4.6 Algorithmes basés sur la mise-à-jour des arbres de recherche

Nous introduisons dans cette section des algorithmes de recherche d'homomorphismes et de transformation de graphes en profitant du contexte d'utilisation vers lequel nous les destinons. En effet, que ce soit pour la génération des instances (correspondant à un graphe terminal) appartenant à un style architectural (correspondant à une grammaire de graphes) ou pour la reconfiguration de l'architecture en appliquant une règle de reconfiguration (correspondant à une règle de transformation) sur l'instance d'architecture courante (correspondant à un graphe), l'ensemble de règles à appliquer est stable[54].

[53] De manière intuitive, on peut voir que si un graphe intermédiaire G1 est homomorphe à un graphe G2, alors cet homomorphisme reste valide si on supprime un arc dans G1 mais peut devenir non valide si on introduit un autre arc dans G2.

[54] La seul exception concerne le cas où nous mettons à jour un protocole de reconfiguration pendant l'exécution en prenons en compte de nouveaux événements et de nouvelles règles de reconfiguration.

Nous avons présenté, dans les sections précédentes, un algorithme permettant de trouver tous les homomorphismes entre un graphe modèle et un graphe d'entrée donnés. Cet algorithme se base sur le principe domino pour construire un arbre de recherche où chaque nœud correspond à un homomorphisme intermédiaire impliquant un sous graphe du graphe modèle. Chaque ramification de cet arbre de recherche correspond à la prise en compte d'un nœud supplémentaire pour étendre, si cela est consistant, un homomorphisme intermédiaire. À la fin de ce processus, nous obtenons un arbre de recherche où les feuilles représentent les homomorphismes intermédiaires maximaux. Les solutions correspondent aux feuilles où l'algorithme à réussi l'intégration de tous les nœuds du graphe modèle.

Pour la réécriture de graphes, nous nous basons sur cet algorithme pour construire les ensembles d'homomorphismes entre 1) la partie L de la règle et le graphe hôte, et 2) entre le graphe $L \cup N$ de la règle et ce même graphe hôte. Les homomorphismes que nous considérons valides sont ceux qui appartiennent à la différence entre les deux ensembles.

Pour l'algorithme développé ici, nous profitons de la stabilité de l'ensemble des règles de réécriture (et par conséquent des graphes modèles L et $L \cup N$) et nous soutenons qu'il est judicieux de mettre à jour l'arbre de recherche générée (i.e. les homomorphismes intermédiaires) par l'algorithme entre deux transformations plutôt que de reconstruire l'ensemble de l'arbre de recherche et repartir à zéro à chaque transformation.

De manière intuitive, ce raisonnement est encore plus justifié dans le cas où le graphe d'entrée est très grand par rapport aux règles de réécriture. Dans ce cas, l'arbre de recherche ne serait que légèrement modifié d'une transformation à une autre puisque le nombre de nœuds et d'arcs supprimés ou introduits serait largement plus petit que les nœuds et les arcs du graphe initial. Les deux graphes avant et après l'application de la règle serait, donc, assez proches et par conséquent si nous considérons les mêmes graphes modèles, les arbres de recherche seraient aussi assez proches.

L'algorithme de construction de l'arbre de recherche suivra initialement le même processus que l'algorithme générique. Nous y introduirons des dispositifs permettant de stocker les arbres de recherche tout au long de leur construction. D'autre part, afin de pouvoir mettre à jour ces arbres, nous verrons qu'il est important de garder une trace indiquant l'origine (nœud dans le graphe d'entrée) des affectations des labels variables.

Dans ce qui suit, nous présentons de nouvelles versions des procédures de base de l'algorithme en considérant ces impératifs. Ensuite, nous donnerons les procédures permettant la mise à jour des arbres de recherche et réalisant la recherche d'homomorphismes et la transformation de graphes en considérant un ensemble stable de règles de réécriture et un graphe d'entrée initial donné.

Il est important de noter que pour la suite, nous parlerons de *nœuds de l'arbre de recherche* pour désigner les différents homomorphismes intermédiaires, les homomorphismes finaux, et les échecs développés pendant le processus de recherche d'homomorphismes. Pour éviter toute confusion nous rajouterons le qualificatif "nœud du graphe ..." pour désigner les nœuds du graphe d'entrée ou du graphe modèle.

4.6.1 Algorithmes de recherche d'homomorphismes

La procédure d'unification de labels est strictement la même que pour l'algorithme générique avec un léger changement impliquant la prise en compte des nœuds considérés pour ces labels dans les paramètres d'entrée et de sortie de la procédure. Cette considération est nécessaire pour permettre de tracer l'origine des différentes affectations. Nous obtenons, donc, la procédure décrite dans la figure 4.15.

UNIFY_PARAMS(l_1, l_2, n)
1- if l_1 is *constant* then
  1-1. if $l_1.type \neq l_2.type$ or $l_1.value \neq l_2.value$ then return(NULL)
  1-2. else return($\emptyset$)
2- if l_1 is *variable* then
  2-1. if $l_1.type = l_2.type$ then return($\{(l_1, l_2, n)\}$)
  2-2. else return(NULL)

FIG. 4.15 – Pseudo-code de la "nouvelle" procédure UNIFY_PARAMS.

Pour la procédure MERGE, nous avons besoin de continuer à tracer l'origine des valuations. Une affectation possédera maintenant la structure : (x, v, n) où x représente une variable, v correspond à la valuation attribuée à x, et n le nœud qui est à l'origine de cette affectation. La procédure MERGE qui, à part ce détail, reste inchangée, est présentée dans la figure 4.16.

MERGE(Val_1, Val_2)
0- Let $Val_{1.2} = \emptyset$
1- For all valuations $(x_1, v_1, n_1) \in Val_1$
  1-1. For all valuations $(x_2, v_2, n_2) \in Val_2$
    1-1-1. if $x_1 = x_2$ and $value_1 \neq value_2$ then return(NULL)
2- $Val_{1.2} = Val_1 \cup Val_2$
3- return($Val_{1.2}$)

FIG. 4.16 – Pseudo-code de la procédure MERGE.

Pour la définition de la procédure UNIFY_VERTICES, nous renvoyons maintenant un homomorphisme intermédiaire (i.e. entre les deux nœuds en entrée) au lieu de la seule affectation renvoyée précédemment. Afin de faciliter la mise à jour de l'arbre de recherche, nous avons besoin, en plus, d'enregistrer l'ensemble de nœuds visités (contenant dans ce cas un seul nœud) du graphe modèle.

UNIFY_VERTICES(n_1, n_2)
0- Let Val = $\emptyset$ and let $Lab(n_1) = (l_{1.1}, \ldots, l_{1.m})$ and let $Lab'(n_2) = (l_{2.1}, \ldots, l_{2.k})$
1- if $(m \neq k)$ then return(NULL)
2- Let i=1
3- While $i \leq m$
  3.1 V=UNIFY_PARAMS($l_{1.i}, l_{2.i}, n_1$)
  3.2 if V=NULL then return(NULL)
  3.3 Val=MERGE(Val, V)
  3.4 if Val=NULL then return(NULL)
  3.5 i=i+1
4- Let $f \in \{n_1\} \times \{n_2\}$ such that $f(n_1) = n_2$
5- return((f,$\{n_1\}$,Val))

FIG. 4.17 – Pseudo-code de la procédure UNIFY_VERTICES.

Afin de pouvoir mettre à jour et de compléter les homomorphismes intermédiaires nous implémentons deux procédures. La procédure TEST_MG_VERTEX correspond à la procédure TEST_VERTEX de l'algorithme générique en prenant en compte les nouveaux types des paramètres de sortie utilisés par

TEST_MG_VERTEX (n_M, MG, IG)

0- Let $MG = (N_M, A_M, Lab_M, D_{LabM}, Etiq_M, D_{EtiqM})$
and $IG = (N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$ and let F=$\emptyset$
1- For all $n_I \in N_I$, let F_{res}= UNIFY_VERTICES(n_M, n_I)
  1-1. if $F_{res} \neq NULL$ then
  1-2. if $[e = (n_M, n_M) \notin A_M]$ or
    if $[e = (n_M, n_M) \in A_M$ and $e' = (v_I, v_I) \in A_I$ and $Etiq_M(e) = Etiq_I(e')]$ then
  1-3. let $F = F \cup \{F_{res}\}$
2- return (F)

FIG. 4.18 – Pseudo-code de la procédure TEST_MG_VERTEX.

les nouvelles procédures UNIFY_PARAMS et MERGE. Elle permet de calculer toutes les unifications possibles entre un nœud du graphe modèle et tous les nœuds du graphe d'entrée.

TEST_IG_VERTEX (n_I, MG, IG)

0- Let $MG = (N_M, A_M, Lab_M, D_{LabM}, Etiq_M, D_{EtiqM})$
and $IG = (N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$ and let F=$\emptyset$
1- For all $n_M \in N_M$, let F_{res}= UNIFY_VERTICES(n_M, n_I)
  1-1. if $F_{res} \neq NULL$ then
  1-2. if $[e = (n_M, n_M) \notin A_M]$ or
    if $[e = (n_M, n_M) \in A_M$ and $e' = (n_I, n_I) \in A_I$ and $Etiq_M(e) = Etiq_I(e')]$ then
  1-3. let $F = F \cup \{F_{res}\}$
2- Return (F)

FIG. 4.19 – Pseudo-code de la procédure TEST_IG_VERTEX.

La procédure TEST_IG_VERTEX permet d'effectuer le calcul inverse de la procédure TEST_MG_VERTEX en cherchant toutes les unifications possibles entre chaque nœud du graphe modèle et un nœud donné appartenant au graphe d'entrée. Cette procédure sera très utile pour permettre une optimisation des calculs nécessaires pour la mise à jour des arbres de recherche.

Concernant la nouvelle procédure d'intégration d'un nouveau nœud (i.e. ADD_VERTEX), nous avons besoin de pouvoir tracer le chemin de toutes les branches (même celles qui ont échoué[55]) de l'arbre de recherche à l'état courant. Pour chaque feuille, à un état courant de développement donné, nous sauvegardons les homomorphismes intermédiaires pères au moment d'introduire un nœud avec la procédure ADD_VERTEX[56]. Aussi, le fait que l'ordre de l'introduction des nœuds n'est plus indifférent (du fait qu'il est dicté par l'ordre utilisé dans la construction des homomorphismes intermédiaires), il est nécessaire d'enregistrer les nœuds visités. L'arrêt d'une branche de l'arbre correspond à une valuation NULL de l'homomorphisme intermédiaire.

[55] En effet et dans l'optique d'une prochaine transformation, l'introduction d'un nouveau nœud par l'application d'une règle peut transformer le graphe de telle manière que cette branche va pouvoir être développée et réussir.

[56] Cette procédure prend, donc, en entrée l'arbre de recherche intermédiaire AR'_M au lieu de l'ensemble des homomorphismes intermédiaires F'_M comme dans l'algorithme générique.

ADD_VERTEX$(IG, MG', n_M, AR'_M, F, A_M)$

0- Let $MG' = (N'_M, A'_M, Lab'_M, D'_{LabM}, Etiq'_M, D'_{EtiqM})$
and let $IG = (N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$, and let AR_{res}=$\emptyset$
1- For every pair F_1, f_2 and Val_1, Val_2 where $F_1 \in AR'_M$ and $(f_2, \{n_M\}, Val_2) \in F$,
let $(f_1, N"_M, Val_1)$[57]=$Last(F_1)$[58]
1-1- if $Val_1 \neq$ NULL [59]
1-1-1. Test the conditions :
i. $f_1(N_M) \cap f_2(\{n_M\}) = \emptyset$ [60]
ii. For each edge $e_1 = (n'_M, n_M) \in A_M$ such that $n'_M \in N'_M$,
there exists an edge $e'_1 = (f_1(n'_M), f_2(n_M)) \in A_I$ such that $Etiq_M(e_1) = Etiq_I(e'_1)$
iii. For each edge $e_2 = (n_M, n'_M) \in A_M$ such that $n'_M \in N'_M$,
there exists an edge $e'_2 = (f_2(n_M), f_1(n'_M) \in A_I)$ such that $Etiq_M(e_2) = Etiq_I(e'_2)$
1-1-2. if i. and ii. and iii. hold then
1-1-2-1. MERGE$(Val_1, Val_2) = Val_{I,M}$
1-1-2-2. build $f \subseteq (N'_M \cup \{n_M\}) \times N_I$ such that,
$f(n) = f_1(n)$ if $n \in N'_M$ and $f(n_M) = f_2(n_M)$ otherwise
1-1-2-3. Let F'_1=F_1@$(f, N'_M \cup \{n_M\}, Val_{I,M})$
1-1-2-4. $AR_{res} = AR_{res} \cup \{F'_1\}$
1-1-3. else
1-1-3-1. build $f \subseteq (N_M \cup \{n_M\}) \times N_I$ such that,
$f(n) = f_1(n)$ if $n \in N_M$ and $f(n_M) = f_2(n_M)$ otherwise
1-1-2-3. Let F'_1=F_1@$(f, N"_M \cup \{n_M\}, NULL)$
1-1-2-4. $AR_{res} = AR_{res} \cup \{F'_1\}$
1-2. else[61] $AR_{res} = AR_{res} \cup \{F_1\}$
2- Return(AR_{res})

FIG. 4.20 – "Nouveau" pseudo-code de la procédure ADD_VERTEX.

La procédure de recherche d'homomorphismes de base (i.e. BASIC_MATCH_GRAPH) suit la même logique globale (commencer par un nœud et intégrer les nœuds de graphe modèle un à un jusqu'à les intégrer tous). La différence se situe dans la prise en compte des nouveaux types des paramètres d'entrée/sortie des procédures appelées permettant l'enregistrement de l'arbre de recherche et les liens entre les affectations et les nœuds du graphe modèle. Cette procédure est utilisée pour construire l'arbre de recherche initial (i.e. pour L et $L \cup N$). Elle sera donc utilisée une seule fois à la première application de chaque règle. Pour les applications ultérieures, nous ferons appel à la mise à jour de cet arbre de recherche.

Le même discours est valable pour la procédure permettant la vérification de l'applicabilité d'une règle. L'algorithme général de la procédure BASIC_MATCH_RULE est très proche de la procédure MATCH_RULE de l'algorithme générique avec comme différence principale le type des paramètres de sortie. Ici, au lieu de renvoyer l'ensemble des homomorphismes valides en considérant les partie L et N de la règle, nous renvoyons un triplet correspondant aux arbres de recherche pour les homomorphismes entre L et HG, entre $L \cup N$ et HG, et l'arbre contenant les homomorphismes valides pour la règle (i.e. entre L et HG et non extensibles pour couvrir $L \cup N$).

[57] Nous utilisons N''_M au lieu de N'_M parce que cette branche peut avoir déjà échoué et son extension arrêtée avant d'atteindre N'_M.
[58] $Last(F1)$ correspond à la feuille de la branche, i.e. l'homomorphisme le plus développé de la branche qui correspond à celui qui a été introduit en dernier.
[59] F_1 est une branche qui n'a pas échoué.
[60] condition d'injectivité.
[61] F_1 a déjà échoué.

BASIC_MATCH_GRAPH (IG, MG)

0- Let $MG = (N_M, A_M, Lab_M, D_{LabM}, Etiq_M, D_{EtiqM})$
and let $IG = (N_I, A_I, Lab_I, D_{LabI}, Etiq_I, D_{EtiqI})$
Let $n = Card(N_M)$ and $m = Card(N_I)$ and $F_{res} = \emptyset$
1- If $n > m$ then return($\emptyset$)
2- For all $i \in [0..n-1]$
2-1. Let $n_{Mi} = N_M[i]$ and let F_i=TEST_MG_VERTEX(n_{Mi}, MG, IG)
2-2. if $F_i = \emptyset$ then return($\emptyset$)
3- Let $MG' = \{n_{M0}\}$ and let $AR_{res} = \{F_0\}$ and $AR'_{res} = \{F_0\}$
4- For all $i \in [1..n-1]$
4-1. AR'_{res}=ADD_VERTEX$(IG, MG', n_{Mi}, AR_{res}, F_i, A_M)$
4-2. if (for all $(f, N, v) = Last(F')$ such that $F' \in AR'_{res}$, we have $v = NULL$) then return(AR'_{res})[62]
4-3. $AR_{res} = AR'_{res}$
4-4. $MG' = MG' \cup \{n_{Mi}\}$
4-5. $i = i + 1$
5- Return(AR_{res})

FIG. 4.21 – Pseudo-code de la procédure basique pour la recherche d'homomorphismes de graphes

BASIC_MATCH_RULE (RG, HG)

0- Let $RG = (L, K, R, N, C)$, let $N \setminus L = (N_{N\setminus L}, A_{N\setminus L}, Lab_{N\setminus L}, D_{LabN\setminus L}, Etiq_{N\setminus L}, D_{EtiqN\setminus L})$
Let $L \cup N = (N_{L\cup N}, A_{L\cup N}, Lab_{L\cup N}, D_{LabL\cup N}, Etiq_{L\cup N}, D_{EtiqL\cup N})$,
let $HG = (N_H, A_H, Lab_H, D_{LabH}, Etiq_H, D_{EtiqH})$, let $F_{res} = \emptyset$
Let $n = Card(N_{L\cup N})$, $m = Card(N_H)$, $k = Card(N_{N\setminus L})$
1- AR_L = BASIC_MATCH_GRAPH(L, HG)
2- if (for all $(f, N, v) = Last(F)$ such that $F \in AR_L$, we have $v = NULL$) then return$(AR_L, AR_L, \emptyset)$[63]
3- if $n > m$ then return(AR_L, AR_L, AR_L)
4- For all $i \in [0..k-1]$
a. Let $n_{N\setminus L,i} = N_{N\setminus L}[i]$ and let F_i =TEST_MG_VERTICES$(n_{N\setminus L,i}, HG)$
b. if $F_i = \emptyset$ then return$((AR_L, AR_L, AR_L))$
5- Let $RG' = L$, let $AR_{N\cup L} = AR_L$
6- For all $i \in [0..k-1]$
a. $AR_{N\cup L}$ =ADD_VERTEX$(HG, RG', n_{N\setminus L,i}, AR_{N\cup L}, F_i, A_{L\cup N})$
b. if(for all $(f, N, v) = Last(F)$ such that $F \in AR_{N\cup L}$, we have $v = NULL$) then return$(AR_L, AR_{N\cup L}, AR_L)$
c. $RG' = RG' \cup \{n_{N\setminus L,i}\}$
7- Let $AR = AR_L$
8- For all $(f_1, N_1, v_1) = Last(F_1)$ and $(f_2, N_2, v_2) = Last(F_2)$ such that $F_1 \in AR_L$ and $F_2 \in AR_{N\cup L}$
a. If $v_1 \subset v_2$ and for all vertices $n \in L$, we have $f_1(n) = f_2(n)$ then $AR = AR \setminus F_2$
9- return$(AR_L, AR_{N\cup L}, AR)$

FIG. 4.22 – Pseudo-code de la procédure de vérification de l'applicabilité d'une règle.

La procédure basique de réécriture de graphes (i.e. BASIC_APPLY_RULE) implémente la réécriture de graphes en se basant sur l'arbre de recherche produit par la procédure BASIC_MATCH_RULE. En suivant la logique définie pour les règles de réécriture, ceci implique la suppression de la partie $L \setminus K$, l'ajout d'une copie valuée de la partie $R \setminus K$, la suppression des arcs suspendus et l'exécution des instructions

[62] En cas d'absence d'homomorphismes intermédiaires valables, nous renvoyons l'état courant de l'arbre de recherche.
[63] Ici, l'ensemble vide implique l'absence d'homomorphismes valides pour la réécriture.

de connexion. Cette procédure produit, en plus de la mise à jour du graphe hôte, les quatre ensembles correspondant aux nœuds supprimés et introduits et aux arcs supprimés et introduits. Ces ensembles seront utilisés pour mettre à jour les arbres de recherche de toutes les règles de réécriture considérées par l'algorithme final.

BASIC_APPLY_RULE (RG, HG, AR)

0- let $RG = (L, K, R, N, C)$, let $HG = (N_H, A_H, Lab_H, D_{LabH}, Etiq_H, D_{EtiqH})$
Let $HG' = HG$ and let note $HG' = (N'_H, A'_H, Lab'_H, D'_{LabH}, Etiq'_H, D'_{EtiqH})$ and
and $N_{Del} = \emptyset$, $A_{Del} = \emptyset$, $N_{Add} = \emptyset$, $A_{Add} = \emptyset$
1- if for all $(f, N, v) = Last(F)$ such that $F \in AR$, we have $v = NULL$, then return$(\emptyset, \emptyset, \emptyset, \emptyset, HG$,False)
2- Let $(f, N, v) = Last(F)$ such that $F \in AR$ and $v \neq NULL$
3- For all vertices $n_{L\setminus N} \in (L \setminus N)$
3-1. For all edges $e \in A_H$ such that $e = (f(n_{L\setminus N}), n)$ or $e = (n, f(v_{L\setminus N}))$ with $n \in HG$
3-1-1. $A'_H = A'_H \setminus \{e\}$
3-1-2. $A_{Del} = A_{Del} \cup \{e\}$
3-2. $N'_H = N'_H \setminus \{f(n_{L\setminus N})\}$
3-3. $N_{Del} = N_{Del} \cup \{f(n_{L\setminus N})\}$
4- For all vertices $n_R \in R \setminus K$, let $n_{R,H}$ a new vertex
a. let L=VALUATE_VERTEX$(n_R, val, R \setminus K)$
b. $N'_H = N'_H \cup \{n_{R,H}\}$
c. $Lab'_H = Lab'_H \cup (n_{R,H}, L)$
d. $N_{Add} = N_{Add} \cup \{f(n_{L\setminus N})\}$
e. $f = f \cup (n_R, n_{R,H})$
5- For all edges $e = (n_{R1}, n_{R2}) \in A_{R\setminus K}$
a. Let $e' = (f(n_{R1}), f(n_{R2}))$
b. $A'_H = A'_H \cup e'$ and $Etiq'_H = Etiq'_H \cup (e', Etiq_{R\setminus K}((n_{R1}, n_{R2})))$
c. $A_{Add} = A_{Add} \cup \{e'\}$
6- For all $c_i = (n, l_1, etiq_1/etiq_2, d, d')$ with $n \in R$
a. if $d = in$ then
i. For all $n_H \in N_H$ and $n_L \in L \setminus K$ such that $(n_H, f(n_L)) \in A_H$ and $Etiq_H((n_H, f(n_L))) = etiq1$
a. if $d' = in$ then
i. $A'_H = A'_H \cup (n_H, f(n))$
ii. $Etiq'_H = Etiq'_H \cup ((n_H, f(n)), etiq_2)$
iii. $A_{Add} = A_{Add} \cup \{(n_H, f(n))\}$
a. else
i. $A'_H = A'_H \cup (f(n), n_H)$
ii. $Etiq'_H = Etiq'_H \cup ((f(n), n_H), etiq_2)$
iii. $A_{Add} = A_{Add} \cup \{(f(n), n_H)\}$
b. if $d = out$ then
i. For all $n_H \in N_H$ and $n_L \in L \setminus K$ such that $(f(n_L), n_H) \in A_H$ and $Etiq_H((f(n_L), n_H)) = etiq1$
a. if $d' = in$ then
i. $A'_H = A'_H \cup (n_H, f(n))$
ii. $Etiq'_H = Etiq'_H \cup ((n_H, f(n)), etiq_2)$
iii. $A_{Add} = A_{Add} \cup \{(n_H, f(n))\}$
a. else
i. $A'_H = A'_H \cup (f(n), n_H)$
ii. $Etiq'_H = Etiq'_H \cup ((f(n), n_H), etiq_2)$
iii. $A_{Add} = A_{Add} \cup \{(f(n), n_H)\}$
7- $HG = HG'$
8- return$((N_{Del}, A_{Del}, N_{Add}, N_{Del}, TRUE))$

FIG. 4.23 – Pseudo-code de la procédure d'application des règles de réécriture

4.6.2 Mise à jour de l'arbre de recherche

L'idée principale de l'algorithme présenté ici est la suivante : si nous avons des applications successives d'un ensemble de règles sur un même graphe hôte et spécialement si nous considérons des graphes de grande taille, une grande partie des homomorphismes intermédiaires restent complètement ou partiellement valides.

Ici, l'objectif algorithme est d'enregistrer l'arbre de recherche global AR construit pour une recherche d'homomorphismes entre un graphe modèle MG et un graphe d'entrée IG. Ensuite, si IG est transformé en IG' (en supprimant et en introduisant des nœuds et des arcs), de mettre à jour AR pour obtenir le nouvel arbre de recherche AR' et les nouveaux homomorphismes entre MG et IG'.

Les procédures qui suivent (i.e. $UPDATE\_AR\_X\_X\_X$) permettent de mettre à jour l'arbre de recherche après une transformation du graphe d'entrée. Soit IG un graphe d'entrée et MG un graphe modèle et l'arbre de recherche AR correspondant à l'arbre de développement de tous les homomorphismes entre ces deux graphes. Nous présentons dans ce qui suit les impacts considérés sur l'arbre de recherche pour chacune des quatre actions de transformation élémentaires.

– Suppression de nœuds : L'impact de la suppression d'un ensemble de nœuds N_{Del} d'un graphe IG correspond à la suppression de tous les homomorphismes (intermédiaires ou finaux) tels que l'image de MG contient au moins un nœud appartenant à N_{Del}. Ce processus de suppression d'un nœud prend en compte, implicitement, la suppression de tous les arcs qui lient ce nœud à un autre (cf. ligne 1-1-1-1). Il n'est, donc, plus nécessaire de prendre en compte ce type d'arcs dans la mise à jour de l'arbre de recherche concernant la suppression des arcs. La procédure UPDATE_AR_VERTICES_DELETED présente la mise à jour relative à la suppression des nœuds.

UPDATE_AR_VERTICES_REMOVED$(RR, AR, N_{Del}, A_{Del}, IG_{ini})$

0- Let $AR' = AR$, let $A'_{Del} = A_{Del}$ and let $IG_{temp} = IG_{ini}$
1- For all $n_{Del} \in N_{Del}$
  1-1. For all $e \in A'_{Del}$
    1-1-1. If there exists $n' \in IG_{temp}$ such that $e = (n', n_{Del})$ or $e = (n_{Del}, n')$ then
      1-1-1-1. $A'_{Del} = A'_{Del} \setminus \{e\}$
  1-2. $IG_{temp} = IG_{temp} \setminus \{n_{Del}\}$
  1-3. For all $F_i \in AR'$
    1-3-1. Let $(f_i, set_i, val_i) = Last(F_i)$
      1-3-2. If there exists $n_{RG} \in set_i$ such that $f_i(n_{RG}) = n_{Del}$ then
        1-3-2-1. $AR' = AR' \setminus F_i$
2- return$(AR', E'_{Del}, IG_{temp})$

FIG. 4.24 – Pseudo-code de la procédure pour la mise à jour de l'arbre de recherche suite à la suppression de nœuds

– Suppression des arcs : L'impact de supprimer un arc $e = (n_{IG,begin}, n_{IG,end})$ sur la mise à jour d'une branche de l'arbre BR se terminant par un homomorphisme F est différent selon si $n_{RG,begin}$ et $n_{RG,end}$ (tel que $F(n_{RG,begin}) = n_{IG,begin}$ et $F(n_{RG,end}) = n_{IG,end}$) sont liés par un arc ou non. Si c'est le cas, l'homomorphisme F n'est plus valide, autrement, il le reste. Par conséquent, la mise à jour de l'arbre de recherche AR sera réalisée selon la démarche suivante : pour toutes les feuilles de AR, si $e_{RG} = (n_{RG,begin}, n_{RG,end})$ n'existe pas, alors la feuille et toute sa branche (de la racine à la feuille) restent valides. Sinon, la feuille n'est plus valide et la branche doit être mise à jour. Une solution serait de supprimer toute la branche et de la reconstruire depuis la racine. Cependant, nous pouvons constater que le chemin commençant à la racine jusqu'à l'introduction des nœuds $n_{RG,begin}$ et $n_{RG,end}$ reste valide. Pour garder le chemin valide le plus long possible de AR (et optimiser les calculs), nous remontons jusqu'à l'itération où le dernier nœud (entre $n_{RG,begin}$ et $n_{RG,end}$) à été intégré par la procédure ADD_VERTEX. La mise à jour de l'arbre de recherche consécutive à la suppression d'arcs est présentée dans la procédure UPDATE_AR_EDEGES_DELETED.

UPDATE_AR_EDEGES_REMOVED$(RR, AR, A_{Del}, IG_{ini})$

0- Let AR'=AR and let $IG_{temp} = IG_{ini}$
1- For all $e_{Del} = (n, n') \in A_{Del}$
  1-1. $IG_{new} = IG_{new} \setminus \{e_{Del}\}$
  1-2. For all $F_i \in AR'$
    1-2-3. Let $(f_i, set_i, val_i) = Last(F_i)$
    1-2-4. if there exists n_{RG} and n'_{RG} in set_i such that $f_i(n_{RG}) = n$ and $f_i(n'_{RG}) = n'$ then
      1-2-4-1. if there exists an edge e_{RG} such that $e_{RG} = (n_{RG}, n'_{RG})$
        1-2-4-1-1. Let $F'_i = Go\_Back\_To\_Insert(n_{RG}, n'_{RG}, F_i)$
        1-2-4-1-2. Let $(f'_i, set'_i, val'_i) = Last(F'_i)$
        1-2-4-1-3. $F'_i = F'_i \setminus \{(f'_i, set'_i, val'_i)\}$
        1-2-4-1-3. $F'_i = F'_i @ (f'_i, set'_i, NULL)$
        1-2-4-1-3. $AR' = AR' \setminus \{F_i\}$
        1-2-4-1-3. $AR' = AR' \cup \{F'_i\}$
2- return(AR', IG_{temp})

FIG. 4.25 – Pseudo-code de la procédure de mise à jour de l'arbre de recherche après la suppression d'arcs.

La procédure UPDATE_AR_EDEGES_DELETED fait allusion à la procédure GO_BACK _TO_ INSERT décrite ci-dessous. Cette procédure permet de remonter dans une branche jusqu'à l'introduction la plus récente d'un des deux nœuds passés en paramètre. L'application de cette procédure a pour effet de couper la branche de l'arbre de recherche à cet endroit.

GO_BACK_TO_INSERT(n_{RG}, n'_{RG}, F)

0- Let $F_{res} = F$ and let $F'_{res} = F \setminus Last(F)$ and Done=False
1- While not(Done)
  1-1. Let $(f, set, val) = Last(F_{res})$ and $(f', set', val') = Last(F'_{res})$
  1-2. if $set \setminus set' = \{n_{RG}\}$ or $set \setminus set' = \{n'_{RG}\}$
    1-2-1. then Done=True
    1-2-2. else $F_{res} = F'_{res}$ and
      and $F'_{res} = F_{res} \setminus Last(F_{res})$
2- return(F_{res})

FIG. 4.26 – Pseudo-code de la procédure GO_BACK_TO_INSERT

- Introduction de nœuds : Si l'on considère l'introduction d'un ensemble de nœuds N_{Add}, nous pouvons, tout d'abord, noter que tous les homomorphismes appartenant à l'arbre de recherche AR restent valides. Pour l'addition d'un nœud n_{Add} dans IG tel que n_{Add} est unifiable avec un nœud n_{MG} de MG, il est nécessaire de prospecter tous les homomorphismes intermédiaires qui ont échoué à l'intégration de n_{MG}. Dans ce cas, il faut réessayer d'étendre ses homomorphismes en utilisant la fonction f qui fait correspondre n_{MG} à n_{Add}. Pour tous les homomorphismes qui réussissent, nous devons supprimer le nœud n_{MG} et réessayer de le réinstancier en utilisant f. Pour toutes les fonctions qui se terminent par un échec et qui contiennent n_{MG} mais qui échouent à l'intégration d'un autre nœud que n_{MG}, nous devons revenir en arrière jusqu'à l'étape où n_{MG} a été intégré et regénérer la branche de l'arbre de recherche en considérant f seulement. Ceci est formalisé dans la procédure UPDATE_AR_VERTICES_ADDED.
- Introduction d'arcs : Concernant l'impact de l'ajout d'un arc $e = (n, n')$, nous pouvons dire que la mise à jour d'un arbre de recherche AR est limitée aux branches qui ont échoué pendant l'introduction d'un nœud n_{RG} tel que n ou n' sont unifiables avec ce nœud. Par conséquent, nous considérons les feuilles de AR et nous mettons à jour celles qui ont échoués à cause de cette situation spécifique. Nous développons les branches concernés en prenant en compte la présence de l'arc e dans le graphe hôte.

UPDATE_AR_VERTICES_ADDED$(RR, AR, N_{Add}, IG_{ini}, A_{Add})$

0- Let $RR = (L, K, R, N, C)$ and $AR' = AR$ and let $AR" = \emptyset$ and
let $A'_{Add} = A_{Add}$ and $IG_{temp} = IG_{ini}$
1- For all $n_{Add} \in N_{Add}$
1-1. Let $IG_{temp} = IG_{temp} \cup \{n_{Add}\}$
1-2. if there exists $e = (n_{Add}, n) \in A_{Add}$ or if there exists
$e = (n, n_{Add})$ such that $n \in IG_{temp}$
1-2-1. $IG_{temp} = IG_{temp} \cup \{e\}$
1-2-2. $A'_{Add} = A'_{Add} \setminus \{e\}$
1-2-3. Let $F_{n_{Add}} = TEST\_IG\_VERTEX(n_{Add}, L, IG_{temp})$
1-2-4. For all $(f^i_{n_{Add}}, \{n_{Add}\}, val^i_{n_{Add}}) \in F_{n_{Add}}$
1-2-4-1. let $n_L \in L$ such that $f^i_{n_{Add}}(n_L) = n_{Add}$
1-2-4-2. For all $F_j \in AR'$
1-2-4-2-1. Let $(f_j, set_j, val_j) = Last(F_j)$
1-2-4-2-2. if $n_L \in set_j$
1-2-4-2-2-1. Let F'_j =Go_Back_To_Insert(n_L, n_L, F_j)
1-2-3-2-2-2. Let $(f'_j, set'_j, val'_j) = Last(F'_j)$
1-2-3-2-2-3. Let $F"_j = F'_j \setminus \{(f'_j, set'_j, val'_j)\}$ [64]
1-2-3-2-2-4. Let $F'''_i =$
ADD_VERTEX$(L, NewIG, n_{RG}, F"_j, \{(f^i_{n_{Add}}, \{n_{Add}\}, val^i_{n_{Add}})\}, A_L)$
1-2-3-2-2-5. $AR" = AR" \cup F'''_i$
1-2-4-3. $AR' = AR' \cup AR"$
1-2-4-4. $AR" = \emptyset$
2- return$(AR', A'_{Add}, IG_{temp})$

FIG. 4.27 – Pseudo-code pour la mise à jour de l'arbre de recherche après l'ajout de nœuds

UPDATE_AR_EDGES_ADDED$(RR, AR, A_{Add}, IG_{ini})$

0- Let $RR = (L, K, R, N, C)$ and $AR' = AR$ and let $AR" = \emptyset$
and let $IG_{temp} = IG_{ini}$
1- For all $e_{Add} = (n, n') \in A_{Add}$
1-1. $IG_{temp} = IG_{temp} \cup \{e_{Add}\}$
1-2. For all $F_i \in AR'$
1-2-1. Let $(f_i, set_i, val_i) = Last(F_i)$
1-2-2. if there exists n_L and n'_L in set_i such that $f_i(n_L) = v$ and $f_i(n'_{RG}) = n'$ and
there exists an edge $e = (n_L, n'_L)$ such that $e \in L$ and $Val_i = NULL$[65] then
1-2-2-1. Let $F'_i = F_i \setminus \{(f_i, set_i, val_i)\}$
1-2-2-2. Let $(f'_i, set'_i, val'_i) = Last(F'_i)$ and let the vertex $n"_L$ such that
$set_i = set'_i \cup \{n"_L\}$ and let $n" = f_i(n"_L)$ [66]
1-2-2-3. Let $(f"_i, \{n"_{RG}\}, val"_i) = UNIFY\_VERTICES(n"_{RG}, n")$
1-2-2-4. Let $F"_i = ADD\_VERTEX(RG, IG_{temp}, n"_{RG}, F'_i, \{(f"_i, \{n"_{RG}\}, val"_i)\}, A_{RG})$
1-2-2-5. $AR" = AR" \cup F"_i$
1-2-2-6. $AR = AR' \setminus F_i$
1-3. $AR' = AR' \cup AR"$
1-4. $AR" = \emptyset$
2- return(AR')

FIG. 4.28 – Pseudo-code pour la mise à jour de l'arbre de recherche après l'ajout d'arcs

En se basant sur ces quatre opérations basiques, nous introduisons, dans la procédure UPDATE_ARset,

[64] La branche $F"_i$ est coupée juste avant l'intégration de n_{RG}.
[65] Cette dernière condition est toujours vraie mais est mise ici pour apporter plus de clarté.
[66] $v"_{RG}$ correspond ou bien à v_{RG} ou à v'_{RG}.

le traitement général relatif à la mise à jour des arbres de recherche de l'ensemble des règles considérées dans le système de réécriture. Cette procédure prend en entrée l'ensemble $ARset$ comprenant les règles de réécriture et leurs arbres de recherche associés (i.e. pour L, pour $L \cup N$ et pour la réécriture) en plus des quatre ensembles correspondant aux nœuds et aux arcs introduits et supprimés et du graphe hôte.

UPDATE_ARset $(ARset, N_{Add}, A_{Add}, N_{Del}, A_{Del}, IG)$

0- Let $ARset' = \emptyset$
1- For all $(RR_i, AR_i(L), AR_i(L \cup N)) \in ARset$, let $RR_i = (L, K, R, N, C)$
  1-1. Let $IG_{temp} = IG$ and let $(AR'_i(L), A'_{Del}, IG_{temp}) =$
    UPDATE_AR_VERTICES_DELETED$(L, AR_i(L), N_{Del}, A_{Del}, IG_{temp})$
  1-2. $(AR'_i(L), IG_{temp,i}) =$ UPDATE_AR_EDGES_DELETED$(L, AR'_i(L), A'_{Del}(L), IG_{temp})$
  1-3. $(AR'_i(L), A'_{Del}(L), IG_{temp,i}) =$ UPDATE_AR_VERTICES_ADDED$(L, AR'_i(L), N_{Add}, IG_{temp})$
  1-4. $AR'_i(L) =$ UPDATE_AR_EDGES_ADDED$(L, AR'_i(L), A'_{Add}, IG_{temp})$
  1-5. For all $F_{i,j} \in AR'_i(L)$[67]
    1-5-1. let $F'_{i,j} = F_{i,j}$ and let $(f'_{i,j}, set'_{i,j}, val'_{i,j}) = Last(F'_{i,j})$ and let $AR"_i(L) = \emptyset$
    1-5-2. Let $set"_{i,j} = N_L \setminus set'_{i,j}$
    1-5-3. if $val'_{i,j} \neq NULL$ and $set"_{i,j} \neq \emptyset$ then[68]
      1-5-3-1. Let $set'''_{i,j} = set"_{i,j}$
      1-5-3-2. For all $n_{i,j,k} \in set"_{i,j}$
        1-5-3-2-1. Let $F_{i,j,k}$ =TEST_MG_VERTEX$(n_{i,j,k}, L, IG_{temp})$
        1-5-3-2-2. Let $F'_{i,j}$ =ADD_VERTEX$(IG_{temp}, set'''_{i,j}, n_{i,j,k}, F_{i,j,k}, F'_{i,j}, A_L)$
        1-5-3-2-3. $set'''_{i,j} = set'''_{i,j} \cup \{n_{i,j,k}\}$
      1-5-3-3. $AR'_i(L) = AR'_i(L) \setminus F_{i,j}$
      1-5-4-4. $AR'_i(L) = AR'_i(L) \cup F'_{i,j}$
  1-6. Let $IG_{temp} = IG$ and let $(AR'_i(L \cup N), A'_{Del}, IG_{temp}) =$
    UPDATE_AR_VERTICES_DELETED$(L \cup N, AR_i(L \cup N), N_{Del}, A_{Del}, IG_{temp})$
  1-7. $(AR'_i(L \cup N), IG_{temp}) =$
    UPDATE_AR_EDGES_DELETED$(L \cup N, AR'_i(L \cup N), A'_{Del}(L \cup N), IG_{temp})$
  1-8. $(AR'_i(L \cup N), A'_{Add}, IG_{temp}) =$
    UPDATE_AR_VERTICES_ADDED$(L \cup N, AR'_i(L \cup N), N_{Add}, IG_{temp})$
  1-9. $AR'_i(L \cup N) =$
    UPDATE_AR_EDGES_ADDED$(L \cup N, AR'_i(L \cup N), A'_{Add}, IG_{temp})$
  1-10. For all $F_{i,j} \in AR'_i(L \cup N)$
    1-10-1. let $F'_{i,j} = F_{i,j}$ and let $(f'_{i,j}, set'_{i,j}, val'_{i,j}) = Last(F'_{i,j})$ and let $AR"_i(L \cup N) = \emptyset$
    1-10-2. Let $set"_{i,j} = N_{L \cup N} \setminus set'_{i,j}$
    1-10-3. if $val'_{i,j} \neq NULL$ and $set"_{i,j} \neq \emptyset$ then
      1-10-3-1. Let $set'''_{i,j} = set"_{i,j}$
      1-10-3-2. For all $n_{i,j,k} \in set"_{i,j}$
        1-10-3-2-1. Let $F_{i,j,k}$ =TEST_MG_VERTEX$(n_{i,j,k}, L \cup N, IG_{temp})$
        1-10-3-2-2. Let $F'_{i,j}$ =ADD_VERTEX$(IG_{temp}, set'''_{i,j}, n_{i,j,k}, F_{i,j,k}, F'_{i,j}, A_{L \cup N})$
        1-10-3-2-3. $set'''_{i,j} = set'''_{i,j} \cup \{n_{i,j,k}\}$
      1-10-3-3. $AR'_i(L \cup N) = AR'_i(L \cup N) \setminus F_{i,j}$
      1-10-3-4. $AR'_i(L \cup N) = AR'_i(L \cup N) \cup F'_{i,j}$
  1-11. $ARset' = ARset' \cup (R_i, AR'_i(L), AR'_i(L \cup N))$
1- return$(ARset')$

FIG. 4.29 – Pseudo-code de la mise à jour de l'arbre de recherche après une transformation de graphes décomposée en actions basiques

Nous définissons la procédure UPDATE_ARset qui permet de mettre à jour les arbres de recherche

[67] Cette boucle permet de terminer les branches qui ne sont pas encore totalement développées
[68] Ceci correspond à une branche non complètement développée qui est apparue suite à l'addition d'un nœud ou d'un arc

entre la partie L de la règle et le graphe d'entrée en considérant les nœuds et les arcs qui vont être introduits ou supprimés. Cette procédure est la base de l'application des règles de réécriture (présentée dans les procédures OPTIMIZED_APPLY_RULES et TRANSFORM_GRAPH). Pour cette application, nous distinguons deux cas : soit 1) la règle n'a jamais été appliquée au graphe. Dans ce cas nous devons construire la totalité de l'arbre de recherche pour les homomorphismes. Ensuite, si la règles est applicable, mettre à jour les arbres de recherches relatifs aux autres règles. Soit 2) la règle a déjà été appliquée au moins une fois et nous disposons de l'arbre de recherche obtenu après la dernière application d'une règle. Dans ce cas, nous appliquons directement la règle et nous mettons à jour les arbres de recherche des autres règles en conséquence.

OPTIMIZED_APPLY_RULES(R,ARset,IG)

0- Let $ARset_{new} = \emptyset$ and let $ARset' = ARset$
1- if there exists AR_1, AR_2, AR_3 such that $(R, AR_1, AR_2, AR_3) \in ARset$ then[69]
  1-1. Let $AR_L = AR_1$, $AR_{L \cup N} = AR_1$ and $AR = AR_3$
  1-2. else[70]
    1-2-1. Let $(AR_L, AR_{L \cup N}, AR) = BASIC\_MATCH\_RULE(R, IG)$
    1-2-2. Let $ARset' = ARset' \cup \{(R, AR_L, AR_{L \cup N}, AR)\}$[71]
2- Let $(N_{Del}, A_{Del}, N_{Add}, A_{Add}, IG_{new}, Result)$=
  BASIC_APPLY_RULE(R, IG, AR)
  2-1. if Result=$TRUE$
    2-1-1. Let $ARset_{new}$=UPDATE_MATCHINGS
      $(ARset', N_{Del}, A_{Del}, N_{Add}, A_{Add}, IG)$
    1-1-2. return$(ARset_{new}, IG_{new})$
  2-2. else return$(ARset', IG)$[72]

FIG. 4.30 – Pseudo-code pour la mise à jour des arbres de recherche de l'ensemble des règles

TRANSFORM_GRAPH(ER,IG)

0- Let $ARset = \emptyset$
1- While true
  1-1. Let R = GetRuleToApply(ER)
  1-2. Let $(ARset', IG_{new})$= OPTIMIZED_APPLY_RULE$(R, ARset, IG)$
  1-3. $ARset = ARset'$
  1-4. $IG = IG_{new}$

FIG. 4.31 – Réécriture de graphes en considérant un ensemble de règles.

4.7 Analyse de complexité, comparaison avec l'algorithme de base

Soit un graphe modèle composé de N_{MG} nœuds et soit un graphe d'entrée de N_{IG} nœuds. Dans cette section, nous allons faire une étude pour comparer l'efficacité de l'algorithme présenté ci-dessus avec l'algorithme générique présenté dans la première partie de ce chapitre.

Pour réaliser cette étude comparative et quantifier le gain obtenu, nous considérons des variables suivantes : 1) p_{uni}, la probabilité pour qu'un nœud donné du graphe modèle soit unifiable avec un nœud

[69] Ceci implique que R a été appelé au moins une fois
[70] C'est la première fois que R est appliquée
[71] R possède maintenant un arbre de recherche
[72] Ici, il est nécessaire de retourner $ARset'$ et non $ARset$ permettant, même si la règle n'est pas applicable de ne pas perdre les calculs effectués pour le développement de l'arbre de recherche de cette règle

donné du graphe d'entrée, et 2) p_{comp}, la probabilité pour que deux unifications soient compatibles. Nous notons, par conséquent, que :

- Pour chaque nœud du graphe modèle n_{MG}, il y a $p_{uni} \times N_{IG}$ nœuds dans le graphe d'entrée qui sont unifiables avec ce nœud n_{MG}.
- En exécutant la procédure ADD_VERTEX à l'itération 2 (recherche des homomorphismes d'un sous graphe de MG composé de deux nœuds vers le graphe d'entrée IG), il y a $(p_{uni} \times N_{IG})^2$ possibilités de combiner les unifications. Le concept induit par la probabilité p_{comp}, implique qu'il y a $p_{comp}(p_{uni}N_{IG})^2$ homomorphismes valides et $(1 - p_{comp})(p_{uni}N_{IG})^2$ échecs de combinaison.
- En exécutant la procédure ADD_VERTEX à l'itération 3 (impliquant trois nœuds de MG), il y a $p_{comp}(p_{uni} \times N_{IG})^2(p_{uni} \times N_{IG})$ combinaisons possibles. A cette étape, pour considérer un homomorphisme comme valide, il faut que le troisième nœud soit compatible avec les deux premiers. Ainsi, nous aurons $p^2_{comp}(p_{uni}N_{IG})$ nœuds compatibles et par la suite, à l'étape 3, il y a $p_{comp}(p_{uni} \times N_{IG})^2 \times p^2_{comp}(p_{uni} \times N_{IG}) = p^3_{comp} \times (p_{uni} \times N_{IG})^3$ homomorphismes valides.
- En exécutant ADD_VERTEX à l'itération 4, il y a $p^3_{comp}(p_{uni}N_{IG})^3.(p_{uni}N_{IG})$ possibilités de combinaison. Or, le quatrième nœud doit être compatible avec les trois premiers. Nous obtenons, donc, un nombre d'homomorphismes égale à : $p^3_{comp}(p_{uni}N_{IG})^3 p^3_{comp}(p_{uni}N_{IG})$
- Nous obtenons pour l'itération K (impliquant K nœuds de MG) un nombre de combinaisons égale à :

$$p_{comp}^{\sum_{i=1}^{K-2} I}(p_{uni}.N_{IG})^K = p_{comp}^{\frac{(K-2)(K-1)}{2}}(p_{uni}.N_{IG})^K$$

  et un nombre d'homomorphismes intermédiaires de[73] :

$$p_{comp}^{\sum_{i=1}^{K-1} I}(p_{uni}.N_{IG})^K = p_{comp}^{\frac{K(K-1)}{2}}(p_{uni}.N_{IG})^K$$

En résumé, nous avons :

- À chaque étape K (K nœuds à unifier), nous avons
  - $p_{comp}^{\frac{(K-1)(K-2)}{2}}(p_{uni}.N_{IG})^K$ combinaisons possibles.
  - $p_{comp}^{\frac{K(K-1)}{2}}(p_{uni}.N_{IG})^K$ homomorphismes intermédiaires.
  - et $(p_{comp}^{\frac{(k-1)(k-2)}{2}} - p_{comp}^{\frac{K(K-1)}{2}})(p_{uni}.N_{IG})^K$ échecs de combinaisons.
- L'ensemble de toutes les combinaisons possibles de l'arbre de recherche (i.e. correspondant à l'ensemble des nœuds de l'arbre de recherche) est égale à

$$C(N_{IG}) = \sum_{K=2}^{N_{MG}} p_{comp}^{\frac{(K-1)(K-2)}{2}}(p_{uni}.N_{IG})^K$$

Et nous avons donc,

$$\begin{aligned} C(N_{IG}) &= \sum_{K=0}^{N_{MG-2}} p_{comp}^{\frac{K(K+1)}{2}}(p_{uni}.N_{IG})^{K+2} \\ C(N_{IG}) &= (p_{uni}.N_{IG})^2 \sum_{K=0}^{N_{MG-2}} \sqrt{p_{comp}}^{K^2}(\sqrt{p_{comp}}.p_{uni}.N_{IG})^K \end{aligned}$$

4.7.1 Impact de la suppression de nœuds sur l'arbre de recherche

Le coût engendré par la suppression d'un nœud n_{Del} du graphe d'entrée est équivalent au coût engendré par le parcours de toutes les combinaisons de l'arbre de recherche à toutes les itérations et au coût de la suppression de toutes les combinaisons (échecs et homomorphismes intermédiaires) qui prennent en compte n_{Del} dans leurs images. Dans le cas de la suppression d'un ensemble contenant $N_{N,Del}$ nœuds,

[73] Cela peut être aisément prouvé avec un raisonnement par récurrence.

nous obtenons (en se basant sur la formule de C) un arbre de recherche avec une taille égale à :

$$C_1 = (p_{uni}.(N_{IG} - N_{N,Del}))^2 \sum_{K=0}^{N_{MG-2}} \sqrt{p_{comp}}^{K^2} (\sqrt{p_{comp}}.p_{uni}.(N_{IG} - N_{N,Del}))^K$$

4.7.2 Impact de la suppression des arcs sur l'arbre de recherche

Pour considérer l'impact de la suppression d'un arc $e_{Del} = (n_{IG}, n'_{IG})$ sur l'arbre de recherche AR, il y'a deux cas :

- e_{Del} est significatif pour un nœud H de l'arbre AR et, donc, il est nécessaire de supprimer ce nœud de AR. Ce cas de figure implique que n_{IG} et n'_{IG} appartiennent à l'image de H, et que les nœuds correspondants dans MG soient connectés par un arc.
- e_{Del} n'est pas significatif et H reste dans l'arbre de recherche.

Pour considérer le traitement de la suppression des arcs, nous introduisons la probabilité $p_{sig,e}$ qui exprime la probabilité pour qu'un arc soit réellement significatif pour un nœud donné de l'arbre de recherche en sachant que son origine appartient à l'image de H[74].

Par conséquent, l'impact de la suppression d'un arc $e_{Del} = (n_{IG}, n'_{IG})$ implique la suppression de tous les nœuds de l'arbre de recherche contenant dans leur image le nœud n_{IG} avec une probabilité de $p_{sig,e}$. Ainsi, la suppression d'un ensemble contenant $N_{E,Del}$ arcs $e_{Del,i} = (n_{IG,i}, n'_{IG,i})$ impliquerait la suppression, dans l'arbre de recherche, d'un nombre de nœuds qui est équivalent au nombre de nœuds contenant dans leur image au moins un nœud $n_{IG,i}$ multiplié par la probabilité $p_{sig,e}$. En se basant sur la formule C, l'arbre de recherche contient après la suppression de $N_{E,Del}$ arcs un nombre de combinaisons égale à :

$$C_2 = (p_{uni} \times (N_{IG} - p_{sig,e}.N_{E,Del}))^2 \sum_{K=0}^{N_{MG-2}} \sqrt{p_{comp}}^{K^2} (\sqrt{p_{comp}}.p_{uni}.(N_{IG} - p_{sig,e}.N_{N,Del}))^K$$

4.7.3 Impact de l'introduction de nœuds sur l'arbre de recherche

Introduire $N_{N,Add}$ nœuds au graphe IG implique que tous les nœuds développés dans l'arbre de recherche restent valides et que cet arbre doit être développé avec des combinaisons supplémentaires qui découlent des nœuds nouvellement introduits. Le nombre des nœuds dans l'arbre de recherche est, donc, égale à :
C_3 = nombre de nœuds avec IG qui contient $(N_{IG} + N_{N,Add})$ nœuds
et alors, en se basant sur la formule C, nous obtenons

$$C_3 = (p_{uni}.(N_{IG} + N_{N,Add})^2) \sum_{K=0}^{N_{MG-2}} \sqrt{p_{comp}}^{K^2} (\sqrt{p_{comp}}.p_{uni}.(N_{IG} + N_{N,Add}))^K$$

4.7.4 Impact de l'introduction d'arcs sur l'arbre de recherche

En considérant $p_{sig,e}$ la probabilité définie dans la section 4.7.2 et par analogie avec l'impact de la suppression d'un ensemble d'arcs, l'introduction d'un ensemble de $N_{E,Add}$ arcs dans le graphe IG impliquerait l'introduction d'un nombre de nœuds dans l'arbre de recherche égal à[75] :

$$C_4 = (p_{uni}.(N_{IG} + p_{sig,e}.N_{E,Add}))^2 \sum_{K=0}^{N_{MG-2}} \sqrt{p_{comp}}^{K^2} (\sqrt{p_{comp}}.p_{uni}.(N_{IG} - p_{sig,e}.N_{E,Add}))^K$$

Ceci reste cohérent puisqu'en appliquant les deux expressions relatives à la suppression puis à l'introduction de deux ensembles identiques d'arcs, nous retrouvons que la taille de l'arbre de recherche reste la même.

[74] Cette probabilité est égale à la probabilité que le nœud d'arrivée est dans l'image de H multipliée par la probabilité que cet arc soit significatif.

[75] La preuve peut être aisément réalisée par un raisonnement par l'absurde en utilisant la formule de C_2.

4.7.5 Gain obtenu avec le nouvel algorithme

Dans cette section, l'objectif est d'estimer le gain obtenu par l'algorithme avec mise à jour de l'arbre de recherche par rapport à l'algorithme initial. Nous considérons l'introduction de $N_{N,Add}$ nœuds et de $N_{E,Add}$ arcs dans IG et la suppression de $N_{N,Del}$ nœuds et de $N_{E,Del}$ arcs de IG.

Nous exprimons ce gain par le ratio entre les nombres de nœuds développés dans l'arbre de recherche pour chaque algorithme. Dans le cas du deuxième algorithme avec mise à jour de l'arbre de recherche, nous supposons que le coût de suppression de nœuds dans l'arbre de recherche (i.e. suite à la suppression d'arcs et de nœuds dans le graphe d'entrée) est négligeable en comparaison au coût de développement de branches supplémentaires dans cet arbre de recherche (suite à l'introduction de nœuds et d'arcs dans le graphe modèle).

Dans le cas de l'algorithme générique, le nombre de nœuds développés dans l'arbre de recherche correspond à la totalité des nœuds de cette arbre. Nous obtenons en considérant la formule C :

$$C_{Algo1} = C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del} + N_{N,Add} + p_{sig,e}.N_{E,Add})$$

Dans le cas de l'algorithme avec mise à jour de l'arbre de recherche, le nombre de nœuds développés dans l'arbre de recherche correspond à la différence entre 1) le nombre total des nœuds développés pour le graphe final (après suppression et introduction des nœuds et des arcs dans le graphe d'entrée) et 2) le nombre total de nœuds dans l'arbre de recherche après la prise en compte de la suppression des arcs et des nœuds dans le graphe d'entrée. Nous obtenons, par conséquent, un nombre de nœuds développés dans l'arbre de recherche égal à :

$$C_{Algo2} = C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del} + N_{N,Add} + p_{sig,e}.N_{E,Add}) - C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del})$$

Le gain obtenu entre les deux algorithmes peut, donc, être exprimé par la formule :

$$G = \frac{C_{Algo2}}{C_{Algo1}}$$

Ceci implique que :

$$\begin{aligned} G &= \frac{C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del} + N_{N,Add} + p_{sig,e}.N_{E,Add}) - C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del})}{C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del} + N_{N,Add} + p_{sig,e}.N_{E,Add})} \\ G &= 1 - \frac{C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del})}{C(N_{IG} - N_{N,Del} - p_{sig,e}.N_{E,Del} + N_{N,Add} + p_{sig,e}.N_{E,Add})} \end{aligned}$$

Notons $N_{IG,Del} = N_{IG} - N_{N,Del} - p_{sig,e}.N_{e,Del}$
et notons $N'_{N,Add} = N_{N,Add} + p_{sig,e}.N_{E,Add}$
Nous avons, donc,

$$\begin{aligned} G &= 1 - \frac{C(N_{IG,Del})}{C(N_{IG,Del} + N'_{N,Add})} \\ G &= 1 - \frac{(p_{uni}.N_{IG,Del})^2 \sum_{K=0}^{N_{MG}-2} \sqrt{p_{comp}}^{K^2} (\sqrt{p_{comp}}.p_{uni}.N_{IG,Del})^K}{(p_{uni}.(N_{IG,Del} + N'_{N,Add}))^2 \sum_{K=0}^{N_{MG}-2} \sqrt{p_{comp}}^{K^2} (\sqrt{p_{comp}}.p_{uni}.(N_{IG,Del} + N'_{N,Add}))^K} \end{aligned}$$

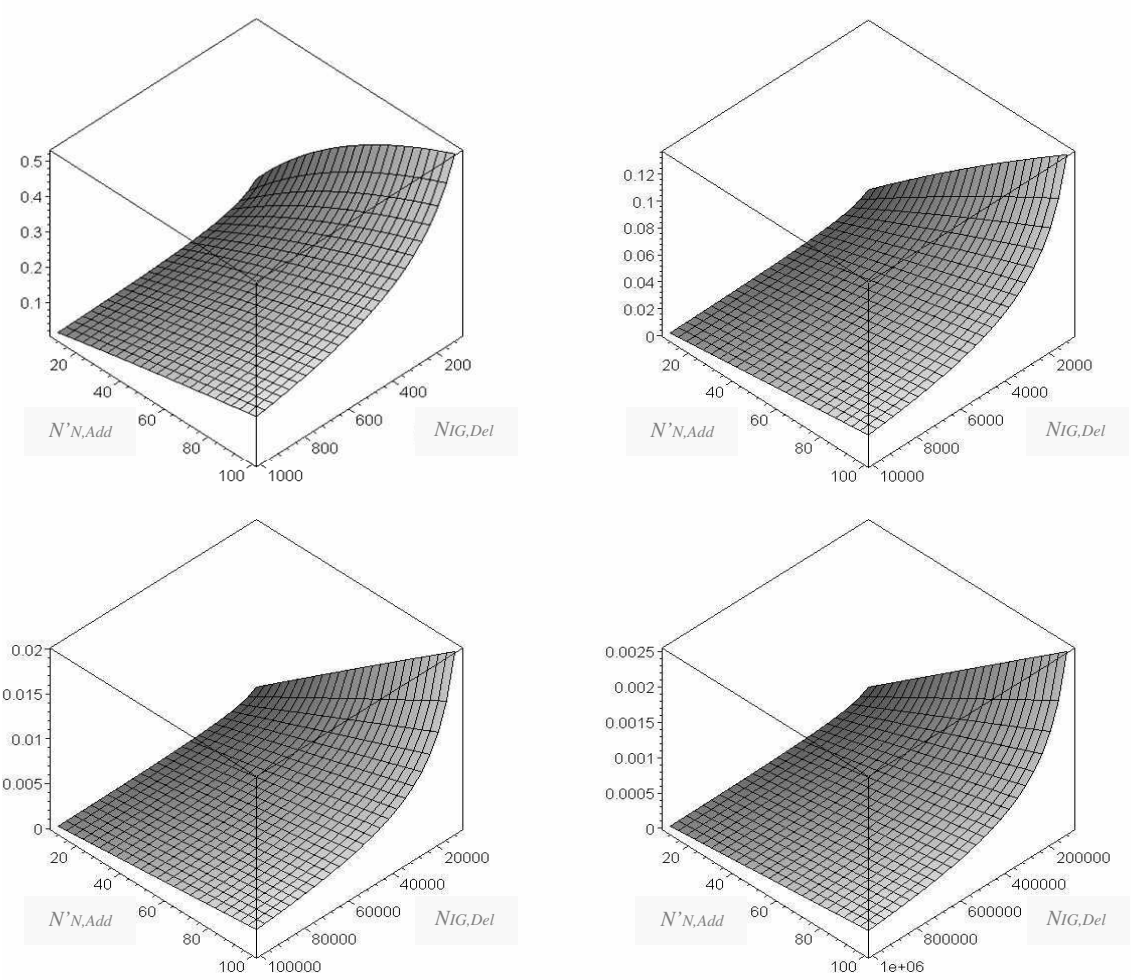


FIG. 4.32 – Ratio de la complexité de l'algorithme avec mise-à-jour de l'arbre de recherche sur la complexité de l'algorithme générique

La figure 4.32 présente les courbes[76] représentant le ratio G en fonction des variables $N_{IG,Del}$ et $N'_{N,Add}$. Pour obtenir ces courbes nous fixons la probabilité $\sqrt{p_{comp}}.p_{uni}$ à 0,1 et nous considérons, dans l'expression de G, $N_{MG} = 2N'_{N,Add}$. Ce dernier choix correspond, plus ou moins, à la situation médiane où les parties L et R ont une même taille t et où K possède la taille $\frac{t}{2}$.

Pour apprécier l'évolution du gain par rapport à la taille du graphe d'entrée et du graphe modèle, nous pouvons noter que la variable $N_{IG,Del}$ correspond à la taille du graphe modèle moins une taille du même ordre que la taille du graphe modèle (i.e. la taille de sa partie $L \setminus K$ plus le nombre des arcs de ce sous graphe multipliant une certaine probabilité). Dans le cas où nous considérons que la taille du graphe d'entrée est très grande comparée à la taille du graphe modèle, nous pouvons considérer que cette variable $N_{IG,Del}$ et la taille du graphe d'entrée sont du même ordre. D'un autre côté, nous avons vu que nous considérons la variable $N'_{N,Add}$ comme étant du même ordre que N_{MG}.

En se basant sur ces considérations, nous pouvons conclure à une réelle amélioration obtenue pour l'algorithme de recherche d'homomorphismes par la mise à jour des arbres de recherches. En comparant les quatre courbes, nous pouvons noter aussi que le gain observé est d'autant plus important que le graphe

[76] Produites en utilisant l'outil MATLAB 7.1.

d'entrée est plus grand et le graphe modèle plus petit.

Ces résultats théoriques correspondent, cependant, à une seule règle de réécriture. Si nous voulons prendre en compte un ensemble contenant n règles de réécriture, le gain devrait être divisé par n puisqu'il est nécessaire de mettre à jour les arbres de recherche pour chaque règle. Mais d'un autre côté, l'algorithme est très facile à paralléliser puisque cette mise à jour est complètement indépendante d'une règle à une autre.

Nous pouvons aussi noter concernant la comparaison des deux algorithmes les remarques suivantes :

- Si nous employons une approche avec calcul parallèle, le calcul pour le second algorithme est équivalent à développer une partie d'un arbre de recherche. Pour l'algorithme de base qui n'est pas (de manière efficace) parallélisable, ce calcul correspond toujours à construire l'arbre de recherche en totalité.
- Pour ce qui concerne une approche *offline* impliquant une mise à jour des arbres de recherche à priori, le calcul pour le second algorithme est équivalent à développer une partie de l'arbre de recherche de toutes règles de transformation. Pour l'algorithme de base, il faut construire la totalité de l'arbre de recherche pour toutes les règles.
- En conclusion, si nous n'employons ni une approche parallèle ni une approche en différé, le calcul de l'algorithme de base est équivalent à construire tout l'arbre de recherche en totalité de la règle à appliquer, alors pour l'algorithme avec mise à jour, il correspond à développer une partie de l'arbre de recherche de la règle à appliquer en plus de développer une partie de l'arbre de recherche pour toutes les autres règles. Ceci rend le deuxième algorithme plus sensible à la fréquence des transformations si notre système inclut un grand nombre de règles considérées.

Suite à ces quelques remarques, nous pouvons ajouter que l'algorithme utilisant une mise à jour de l'arbre de recherche est encore plus optimal si nous nous nous plaçons dans un contexte où : 1) il est parallélisé, 2) les calculs sont réalisés à priori, 3) le nombre de règles potentiellement applicables n'est pas trop grand, ou 4) la fréquence des applications des règles n'est pas très élevée.

4.8 Conclusion

Dans ce chapitre, nous avons présenté un premier algorithme de recherche d'homomorphismes et de transformation de graphes. Cet algorithme prend en compte toutes les extensions introduites pour notre méta-modèle en considérant des nœuds multi-labélés, des labels variables, des conditions d'application négatives et des instructions de connexion.

Pour cet algorithme, nous avons réalisé une étude de complexité et une séries d'expérimentations. Les résultats obtenus ont permis de faire ressortir, comme cela était prévisible, le caractère NP-Complet du problème. Nous avons constaté, pour la résolution de notre problème, une évolution polynômiale et exponentielle respectivement par rapport à la taille du graphe modèle et du graphe d'entrée.

Étant donné que le passage à l'échelle de notre méta-modèle se situe essentiellement au niveau de la taille de l'architecture dynamique traitée, notre approche reste réaliste dans la mesure où cette taille correspond à la taille du graphe d'entrée et sachant que la complexité de l'algorithme est polynômiale en fonction de ce paramètre. D'autre part, les graphe modèles et les règles de réécritures correspondent aux règles de reconfiguration et aux productions des grammaires pour la description des styles architecturaux et des systèmes de transformation verticale. Les règles de réécriture sont donc généralement d'assez petite taille[77].

Ensuite, nous avons profité du caractère stable de l'ensemble des règles de réécriture dans les systèmes que nous modélisons. Nous avons, ainsi, défini un nouvel algorithme basé sur la mise à jour de l'arbre de recherche généré à chaque application d'une règle de réécriture. Nous avons réalisé une analyse comparative qui a permis de démontrer et de quantifier le gain obtenu.

Dans le chapitre qui suit, nous présentons les réalisations logicielles basées sur les algorithmes de recherche d'homomorphismes et de transformation de graphes. Nous verrons, par exemple, comment nous avons utilisé ces algorithmes afin de générer des modules génériques pour la gestion des architectures logicielles.

[77] e.g. pour l'ensemble du cas d'étude présenté dans le chapitre 3 et de tous les autres cas développés dans nos travaux antérieurs, la taille max de la partie L de toutes les règles de réécriture est égale à 12.

Chapitre 5

Implémentations et Réalisations

5.1 Introduction

Dans ce chapitre, nous introduisons les implémentations logicielles réalisées dans le cadre de nos travaux. Nous considérons, dans ce cadre, les algorithmes de recherche d'homomorphismes et de transformation de graphes définis dans le chapitre précédent ainsi que le méta-modèle défini dans le chapitre2 pour la gestion de la reconfiguration de l'architecture. Nous utilisons le langage C++ préféré à Java pour optimiser les performances en terme d'utilisation de mémoire et de temps d'exécution. Ce choix est basé sur plusieurs travaux comparatifs basés sur différents Benchmarks [SPT02, VP02, ben06][78].

Nous présentons, dans un premier lieu, les structures de données utilisées pour implémenter les différentes classes de base relatives aux nœuds, aux arcs, aux graphes et aux règles de réécriture du modèle.

Ensuite, nous présentons une première couche logicielle correspondant à l'implémentation des algorithmes de recherche d'homomorphismes et de transformation de graphes (ATG). Nous nous basons, pour cela, sur les algorithmes génériques définis dans le chapitre précédent. Pour cette première couche, nous présentons aussi une comparaison de performances avec l'outil AGG.

Nous avons vu, dans les chapitres relatifs au méta-modèle et au cas d'étude pour les opérations d'intervention d'urgence (i.e. chapitre 2 et 3), la nécessité de prendre en compte plusieurs règles de réécriture (pouvant, par exemple, intervenir dans les combinaisons des règles pour la reconfiguration) et de prendre en compte plusieurs graphes hôtes (pouvant, par exemple, correspondre à l'instance courante de l'architecture décrite à différents niveaux d'abstraction). Nous présenterons, donc, notre implémentation du moteur de transformation de graphes (MTG) permettant d'enregistrer les instances courantes d'un ensemble de graphes et de les transformer en considérant un ensemble de règles de réécriture.

Finalement, nous présenterons une troisième couche correspondant au protocole de reconfiguration de l'architecture (PRA). La conception de ce protocole s'appuie sur sa définition donnée au chapitre relatif au méta-modèle. Par conséquent, nous prenons en compte des événements de reconfiguration, des combinaisons de règles de reconfiguration, et des graphes représentant les instances courantes de l'architecture à différents niveaux d'abstraction. Nous considérons aussi des mécanismes pour l'instanciation des règles de reconfiguration en se basant sur les paramètres transportés par les événements de reconfiguration.

Pour ces trois couches, nous présenterons des API permettant, d'un côté, leur programmation et leur mise-à-jour pendant l'exécution, et d'un autre côté, leur exécution en permettant la transformation des graphes. Nous illustrons nos réalisations en utilisant la notation UML et en présentant des diagrammes de classes et des diagrammes de séquence.

5.2 Structures générales des données

Dans cette section, nous faisons une description des structures de données des entités élémentaires considérées. Nous proposons des structures pour décrire les nœuds des graphes hôtes et des règles de

[78] Selon les benchmarks, on retrouve un gain entre 20% et 300% en terme de temps d'exécution entre C++ et Java.

réécriture. Ensuite, nous présentons les choix que nous avons fait pour caractériser les graphes et les règles de réécriture.

5.2.1 Structures de données pour les nœuds

Dans notre méta-modèle, nous faisons la distinction entre, d'un côté, les nœuds des graphes hôtes et d'un autre côté, les nœuds des graphes modèles et des règles de réécriture. Dans le premier cas, les labels sont exclusivement constants alors que nous considérons aussi des labels variables dans le deuxième cas.

Dans les deux cas, nous considérons, pour la définition des labels des nœuds, quatre types de labels correspondant dans C++ aux types simples *int, string, bool* et *double*. Étant donné que le nombre de labels n'est pas limité par le méta-modèle, nous utilisons quatre tableaux correspondant à ces quatre types.

Pour les nœuds des graphes hôtes, les éléments de chaque tableau sont des constantes appartenant au type auquel correspond le tableau. Par contre, pour les nœuds des règles de réécriture, les éléments de chaque tableau peuvent correspondre à une constante appartenant au type du tableau ou bien à une variable de ce même type. Nous introduisons, par conséquent, quatre nouveaux types de labels (i.e. ExtendedString, ExtendedInt, ExtendedBool et ExtendedDouble) correspondant aux quatre types considérés et permettant de considérer des valeurs constantes et des valeurs variables. Nous utiliserons, pour chacune de ces classes, deux attributs *kind* et *value* où, par exemple, la constante 1 est définie par une instance de la classe ExtendedInt telle que *kind="const"* et *value=1*, et la variable "x" est définie par une instance telle que *kind="x"* et *value=-1*.

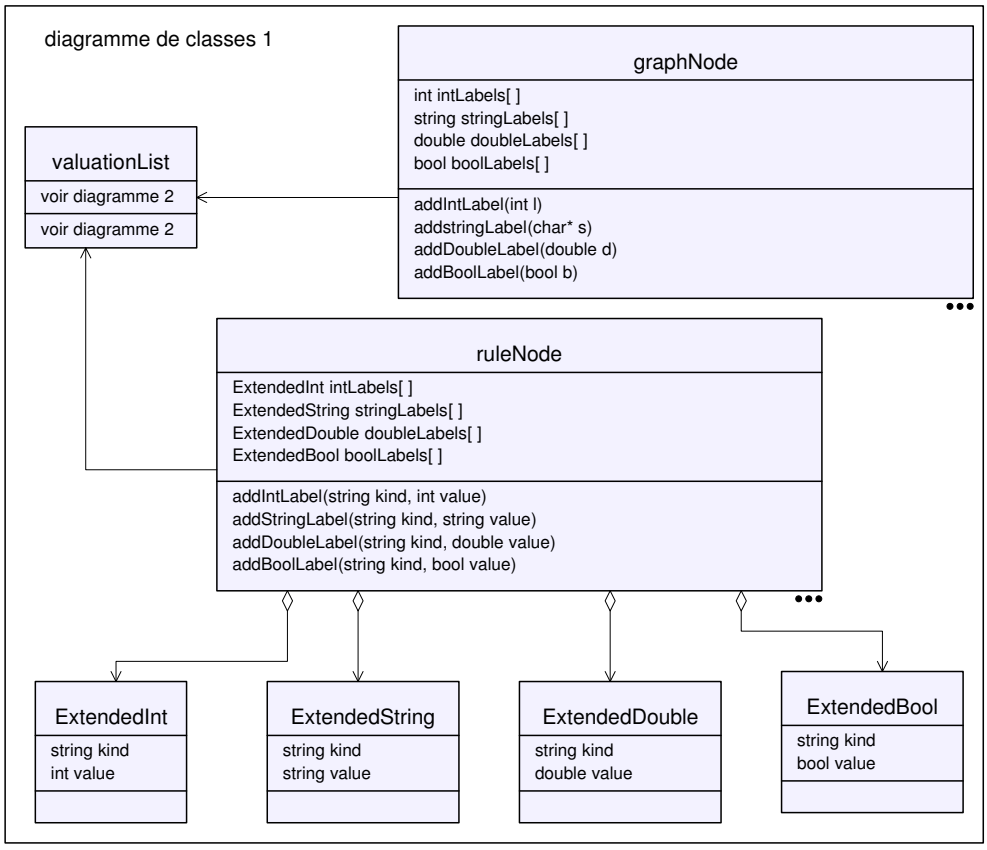


FIG. 5.1 – Diagramme de classes pour les nœuds des graphes modèles et des graphes d'entrée

L'unification d'un nœud appartenant à un graphe modèle et d'un nœud appartenant au graphe d'entrée produit quatre tableaux correspondant aux affectations obtenues par les différentes unifications des labels variables avec les labels constants.

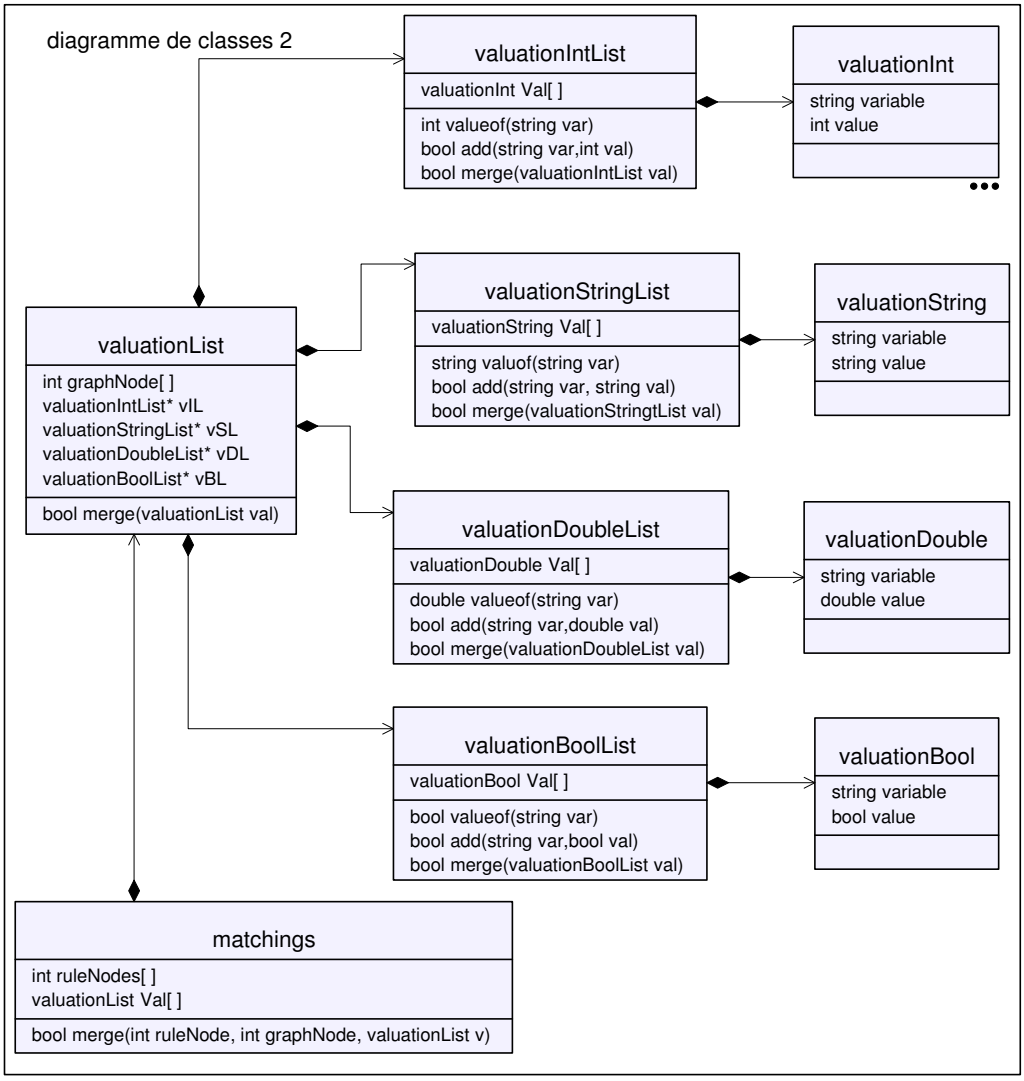


FIG. 5.2 – Diagramme de classes relatif aux structures caractérisant les résultats de la recherche d'homomorphismes

La Figure 5.2 donne un aperçu du diagramme de classes relatif au type des valuations obtenues après l'unification des nœuds. Au final, le résultat d'une recherche d'homomorphismes appartient au type "*matchings*" présenté dans le diagramme de classes de cette figure. Ce type contient deux attributs représentant les homomorphismes intermédiaires obtenus à chaque itération de l'algorithme. Le premier attribut (i.e. "*ruleNodes*") est un tableau de nœuds du graphes modèle qui correspondent aux nœuds déjà traités à l'étape courante de l'algorithme. Le deuxième attribut (i.e. "*Val*") correspond à l'ensemble des homomorphismes intermédiaires obtenus pour le sous graphe du graphe modèle contenant l'ensemble des nœuds appartenant au tableau du premier attribut (i.e. "*ruleNodes*").

De manière informelle, une instance m de la classe *matchings* telle que $m = ([n1, n3, n2], [val1, val2])$

avec, par exemple, $val1 = ([N1, N2, N4], V1, V2, V3, V4)$ où $V1 = [("x", 1), ("y", 2)]$ et $V2 = [("z","toto")]$ indique qu'il existe deux homomorphismes intermédiaires entre le sous graphe contenant les nœuds $n1$, $n2$, et $n3$ et le graphe hôte. H, le premier de ces deux homomorphismes est défini tel que $H(n1) = N1$, $H(n3) = N2$, et $H(n2) = N4$. Cet homomorphisme produit des affectations où les variables entières x et y correspondent respectivement aux valeurs 1 et 2 et où la variable z est affectée par la chaîne de caractères "$toto$".

5.2.2 Structures de données pour les graphes hôtes

La représentation des graphes implique, de manière générale, deux structures de données. La première utilise une structure matricielle alors que la seconde concerne une représentation basée sur les listes chaînées.

Une description basée sur les matrices (un tableau de tableaux) sous entend que l'élément appartenant à $i^{\text{ème}}$ ligne et à la $j^{\text{ème}}$ colonne représente l'étiquette de l'arc reliant le nœud correspondant à la ligne i au nœud correspondant à ligne j. Une valeur spécifique est choisie pour indiquer, le cas échéant, l'absence d'arcs (e.g. pour les entiers, c'est généralement -1 et pour les chaînes de caractères c'est généralement la chaîne vide "").

Un exemple de l'utilisation de cette représentation est donnée dans la figure 5.3 où le graphe G est décrit par : 1) une matrice M qui défini l'ensemble de ses arcs ainsi que leurs étiquettes et, 2) un tableau décrivant les labels de ses nœuds.

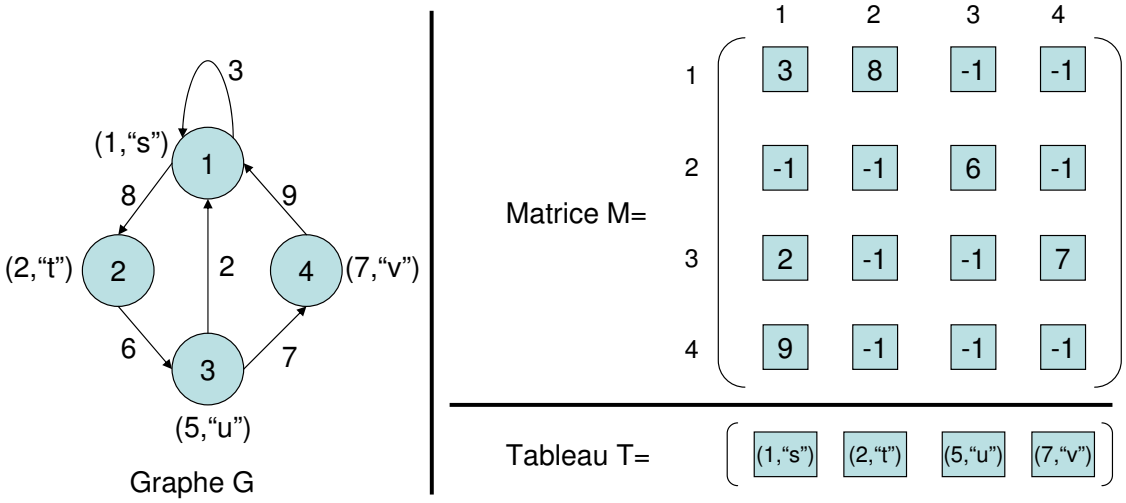


FIG. 5.3 – Caractérisation des graphes par des structures matricielles

Une description basée sur les files ou les listes chaînées à deux dimensions suggère que l'étiquette de l'arc reliant le $i^{\text{ème}}$ nœud au $j^{\text{ème}}$ nœud du graphe est obtenue en réalisant, à partir de l'origine de la structure, $i - 1$ sauts verticaux et $j - 1$ sauts horizontaux.

Un exemple de représentation est donnée dans la figure 5.4 où le même graphe G de la figure 5.3 est décrit par : 1) "L" la liste chaînée à deux dimensions qui défini les arcs et leurs étiquettes, et par 2) "l" une liste chaînée à une dimension qui donne les labels des nœuds.

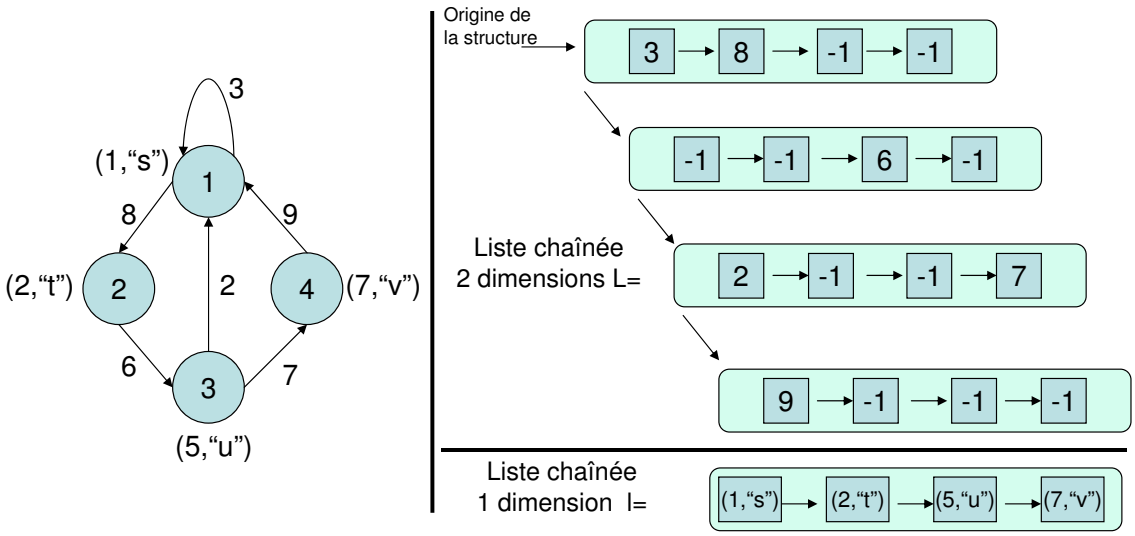


FIG. 5.4 – Caractérisation des graphes par des structures basées sur les listes chaînées

Les deux structures de données précitées présentent des avantages et des inconvénients pour le traitement de la transformation de graphes. En ce qui concerne l'accession à l'étiquette d'un arc donné reliant deux nœuds n_i et n_j donnés, une structure matricielle s'avère bien plus efficace qu'une structure de liste chaînée : dans le cas d'une matrice M, l'accès est immédiat par les éléments M[i][j] et M[j][i], alors que pour une structure de liste chaînée cela nécessite de faire $(i-1)+(j-1)$ opérations "$next$" pour effectuer les $i-1$ sauts verticaux et les $j-1$ sauts horizontaux.

D'un autre côté, s'agissant de la réécriture et de la transformation d'un graphe la structure de liste chaînée est bien plus efficace qu'une structure matricielle. Si nous prenons l'exemple de la suppression d'un nœud n_i, dans le cas d'une liste chaînée, il suffit de rediriger le pointeur vertical de la $(i-1)^{\text{ème}}$ liste vers la $(i+1)^{\text{ème}}$ liste pour ne pas prendre en compte la liste i, et de rediriger les pointeurs horizontaux des $(i-1)^{\text{ème}}$ cellules de chaque liste vers les $(i+1)^{\text{ème}}$ cellules de cette même liste [79]. Dans le cas d'une matrice, la suppression de n_i implique beaucoup plus d'opérations puisqu'il est nécessaire de décaler d'un cran toutes les lignes[80] j et toutes les colonnes k telles que $j > i$ et $k > i$ (i.e. M[j][l] :=M[j-1][l] et M[l][k] :=M[l][k-1])[81].

La différence d'efficacité entre les deux structures concernant la mise à jour et la réécriture des graphes nous pousse à opter pour une caractérisation par les listes chaînées. Cependant et afin de profiter de l'efficacité d'accès offerte par les structures matricielles, nous utiliserons, pour les étapes de calcul une caractérization parallèle basée sur les matrices. L'utilisation des deux structures (une liste chaînée permanente utilisée pour la mise à jour et une matrice créée provisoirement pour réaliser les calculs) permet de tirer profit des avantages qu'offre chaque structure tout en évitant les inconvénients de son utilisation.

[79] Ce qui fait un total de N opérations pour une liste chaînée représentant un graphe avec N nœuds.
[80] Pour une matrice de taille $N \times N$, le coût de cette opération est de N pour chaque ligne décalée.
[81] Ce qui fait un total de $N(N-i)+N(N-i-1)$ opérations pour une matrice représentant un graphe de N nœuds.

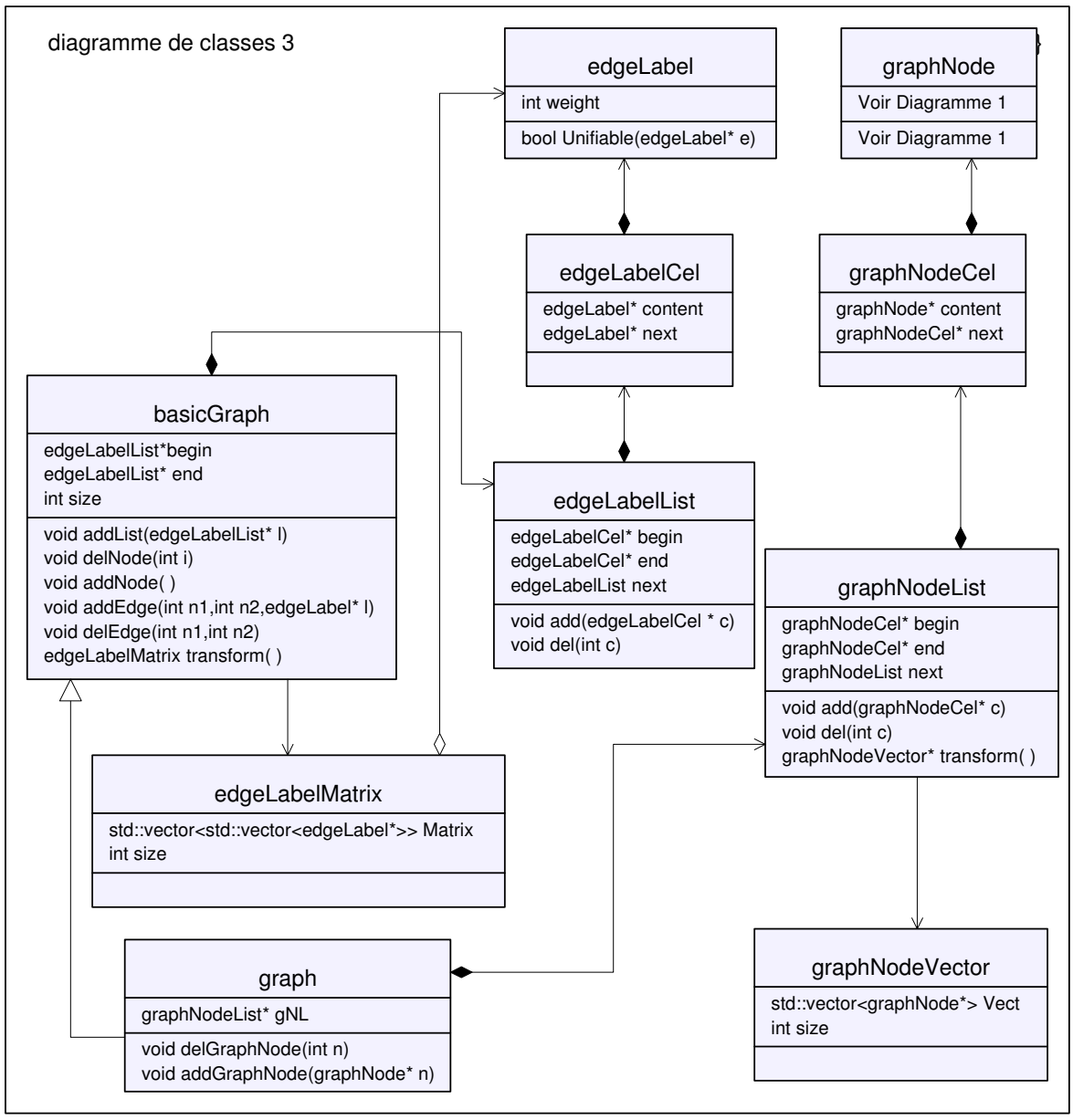


FIG. 5.5 – Diagramme de classes pour les graphes hôtes

La figure 5.5 donne une partie du diagramme de classes de la première couche relative aux structures de données utilisées pour les graphes hôtes. Ce diagramme se réfère à la classe "*graphNode*" introduite dans la figure 5.1. À partir de cette classe, nous définissons les cellules ainsi que la classe des listes chaînées (i.e. respectivement les classes "*graphNodeCel*" et "*graphNodeList*") permettant de caractériser les attributs et les nœuds appartenant au graphe hôte. Ce diagramme exhibe aussi les classes relatives à la caractérisation des arcs et de leurs labels [82] (e.g. "*edgeLabel*" pour les labels des arcs, "*edgeLabelList*" pour définir une liste chaînée d'étiquettes).

[82] Nous notons, ici, que nous considérons des labels entiers pour les arcs.

La classe "*basicGraph*", dont vont hériter les deux classes relatives aux graphes modèles et aux graphes d'entrée, permet de définir la liste chaînée à deux dimensions qui caractérise les étiquettes des arcs. La classe des graphes hôtes (i.e. "*graph*") hérite de cette classe et l'étend en introduisant la liste des nœuds du graphe ainsi que leurs labels.

Pour finir, nous introduisons les deux classes "*graphNodeVector*" et "*edgeLabelMatrix*" qui correspondent à une représentation du graphe sous format matricielle. Notons la présence des deux méthodes "*transform*" (de la classe mère "*basicGraph*") et "*transformList*" (de la classe fille "*graph*") permettant de passer d'une structure en liste chaînée vers une structure matricielle. Ces deux dernières classes sont utilisées exclusivement pour les besoins des calculs internes et ne sont pas visibles aux utilisateurs via l'API de la première couche. Nous les montrons ici simplement pour rappeler l'utilisation des deux structures de données.

5.2.3 Structure de données pour les règles de réécriture

Nous utilisons une structure de données similaire à celle des graphes hôtes pour caractériser les différentes zones des règles de réécriture (i.e. L, K, R, N). Étant donné l'éventualité, pour une règle, de contenir des arcs reliant des nœuds appartenant à des zones différentes, nous choisissons de représenter la règle par un graphe global comprenant tous les nœuds de toutes ses zones (i.e. $L \cup R \cup N$). Les étiquettes de cet arbre globale seront décrites, comme pour les graphes hôtes, par des listes chaînées.

La délimitation des différentes zones est donnée par quatre listes chaînées comprenant les nœuds de chaque zone. Nous considérons, aussi, l'éventualité d'avoir des arcs appartenant à une zone différente de celle de leurs nœuds de départ et d'arrivée. Cette situation correspondant au trois cas de figure suivants : 1) les deux nœuds dans K et l'arc dans $R \setminus K$[83], 2) les deux nœuds dans K et l'arc dans $L \setminus K$[84] et 3) les deux nœuds dans L et l'arc dans N[85]. Nous introduisons, alors, trois tableaux "*EdgeVectR*", "*EdgeVectK*", et "*EdgeVectN*" correspondant à ces trois types d'arcs.

La figure 5.6 présente un diagramme de classes qui introduit la classe relative aux règles de réécriture (i.e. la classe "*rule*"). Cette classe hérite aussi de la classe "*basicGraph*" étant donné que nous retenons la même structure que celle des graphes hôtes pour la caractérisation du graphe global de la règle. Elle étend la classe mère en considérant les quatre listes chaînées correspondant aux quatre zones de la règle, la liste des instructions de connexion, ainsi que les trois tableaux correspondant aux arcs particuliers appartenant à une zone de la règle différente de la zone des nœuds qu'ils relient.

L'API de la classe "*rule*" permet de construire des règles de réécriture en introduisant, un à un, les nœuds et les arcs qui les constituent. Un contrôle est réalisée, à chaque introduction d'un élément, permettant de garantir la cohérence de ces règles en interdisant, par exemple, d'introduire un arc se trouvant dans la zone $R \setminus K$ entre deux nœuds se trouvant dans la zone $L \setminus K$.

[83] Ceci exprime l'introduction d'une interdépendance entre deux composants qui existent déjà.
[84] Ceci exprime la suppression d'une interdépendance entre deux composants qui ne seront pas supprimés.
[85] Ceci exprime une condition d'application négative exigeant l'absence d'un lien entre deux composants pour appliquer la règle.

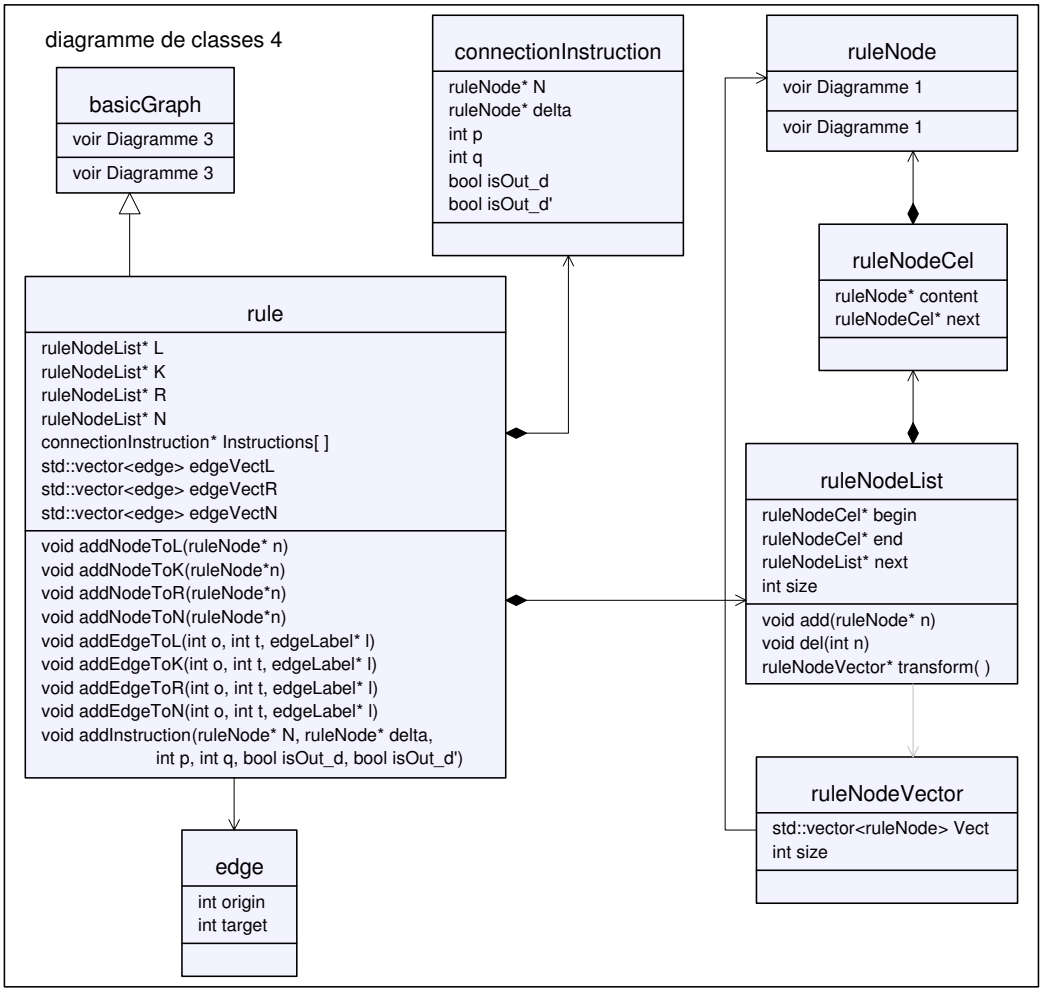


FIG. 5.6 – Diagramme de classes pour les règles de réécriture

5.3 Première couche logicielle : l'Algorithme de Transformation de Graphes (ATG)

La couche correspondant à l'algorithme de transformation de graphes, présentée dans la figure 5.7, constitue la couche la plus basse sur laquelle seront construites toutes nos autres briques logicielles. Pour son implémentation, nous nous basons sur les structures de données permettant la description des règles de réécriture et des graphes hôtes. Nous utilisons la structure *matchings* pour caractériser l'ensemble des homomorphismes valides obtenus par l'algorithme. La figure 5.7 présente le diagramme de classes impliquant l'ATG dont l'API est constituée essentiellement par deux méthodes permettant respectivement de rechercher tous les homomorphismes valides[86] entre une règle et un graphe, et d'appliquer une règle de réécriture à un graphe[87].

[86] Pour rappel, ces homomorphismes sont constitués par les homomorphismes entre la zone L et le graphe hôte avec la restriction qu'ils ne soient pas extensibles pour couvrir le sous graphe $L \cup N$.

[87] Dans le cas où la règle est applicable, cette méthode renvoie l'homomorphisme choisi parmi l'ensemble des homomorphismes valides.

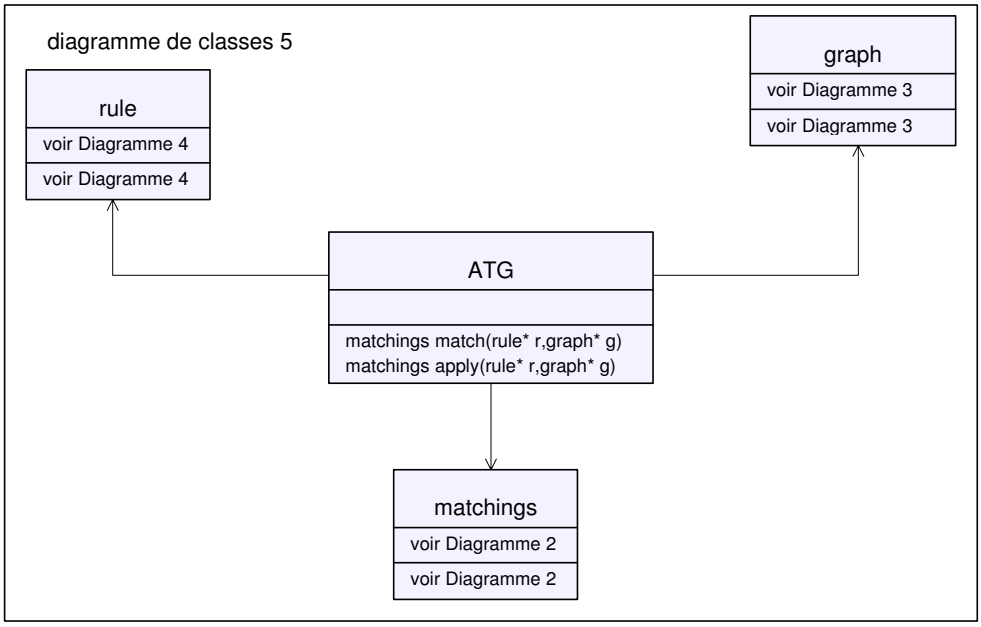


FIG. 5.7 – Diagramme de classes pour l'algorithme de transformation des graphes

5.3.1 Comparaison de performances avec l'outil AGG

AGG (Attributed Graph Grammar system) [AGG06, Tae03] est un environnement de développement basé sur les transformation des graphes. Cet outil, réalisé à l'Université Technique de Berlin (Technische Universität Berlin) et développé depuis 1997 [88], est un des outils de transformation de graphes parmi les plus aboutis et les plus cités dans la littérature.

AGG est développé en utilisant le langage de programmation Java. Il offre une interface graphique permettant de construire les graphes et les règles de transformation et de réaliser des simulations pas à pas et quelques vérifications élémentaires.

Les graphes considérés sont orientés et possèdent des nœuds et des arcs labélés et étiquetés par des objets Java. Les règles de transformation sont définies selon l'approche SPO avec une structure de type (L, R).

Tous les nœuds et les arcs appartenant à la partie gauche (i.e. L) de la règle et qui n'ont pas de contrepartie dans la partie droite (i.e. K) sont supprimés à l'application de la règle. Tous les nœuds et les arcs du côté droit de la règle qui ne possèdent pas de contrepartie dans le côté gauche sont introduits dans le graphe hôte. Les arcs suspendus sont supprimés selon l'approche SPO. Le modèle permet aussi de spécifier des règles d'application négatives.

[88] l'outil est actuellement à la version 1.5.0.

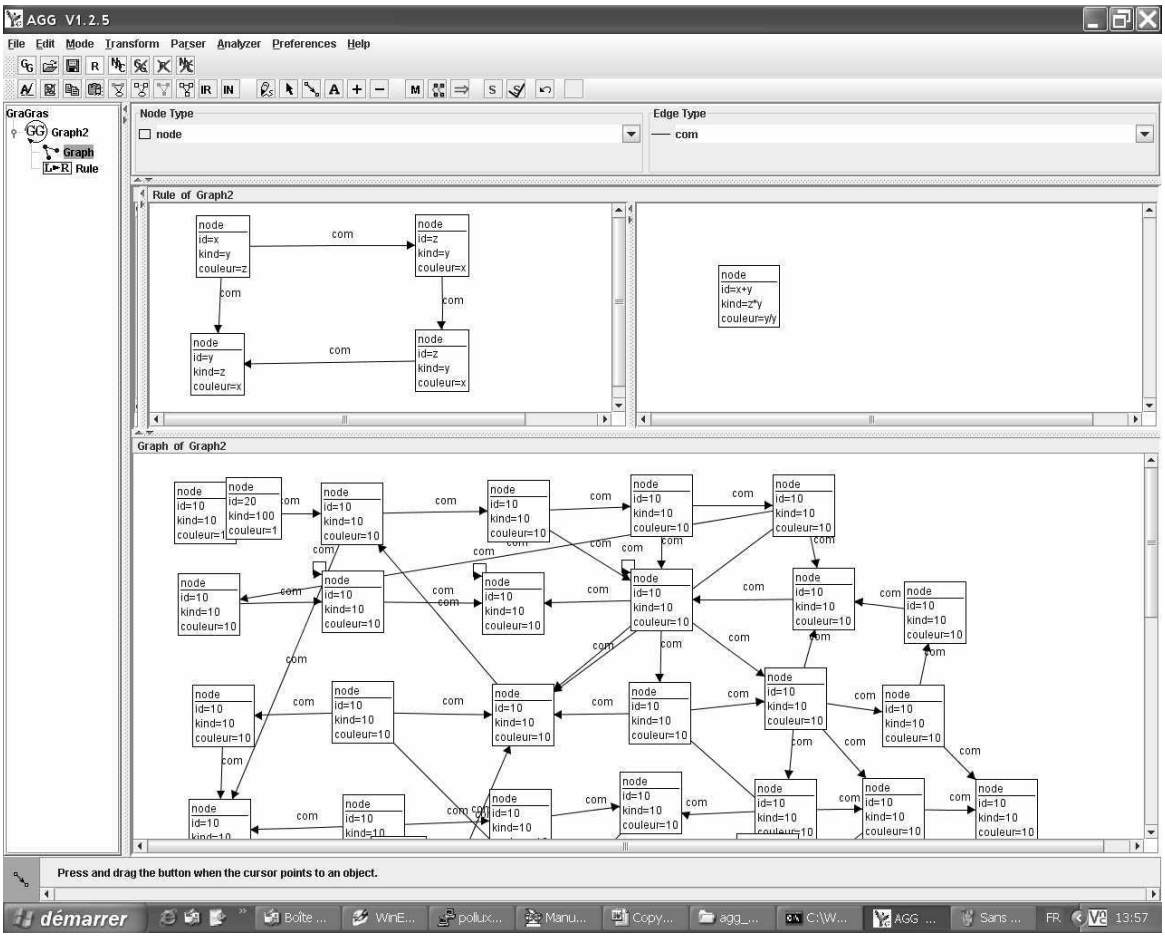


FIG. 5.8 – Interface Graphique de l'outil AGG

L'approche utilisée pour résoudre le problème de recherche d'homomorphismes se base sur sa traduction sous la forme d'un problème de satisfaction de contraintes (*Constraint Satisfaction Problem* (CSP))[Rud98]. Les auteurs de l'outil avancent que l'utilisation de cette approche combinée avec des mécanismes de backtracking intelligents réduisent la complexité dans les cas moyens d'exécution.

Cependant, il est légitime de se demander si ce gain n'est pas affecté par les calculs nécessaires pour transformer le problème de recherche d'homomorphismes en un problème de type CSP, et par la considération de structures de données supplémentaires pour le backtracking intelligent.

Nous avons comparé les résultats expérimentaux obtenus pour la première couche correspondant à l'algorithme de transformation de graphes avec les résultats expérimentaux obtenus en utilisant l'outil AGG. La figure 5.9 présente ces résultats comparatifs qui démontrent une efficacité nettement plus grande de la couche ATG [89].

[89] Ces résultats expérimentaux sont à tempérer puisqu'ils sont obtenus en utilisant l'interface graphique d'AGG qui peut se révéler très gourmande en ressources, alors que pour l'ATG nous utilisons l'API.

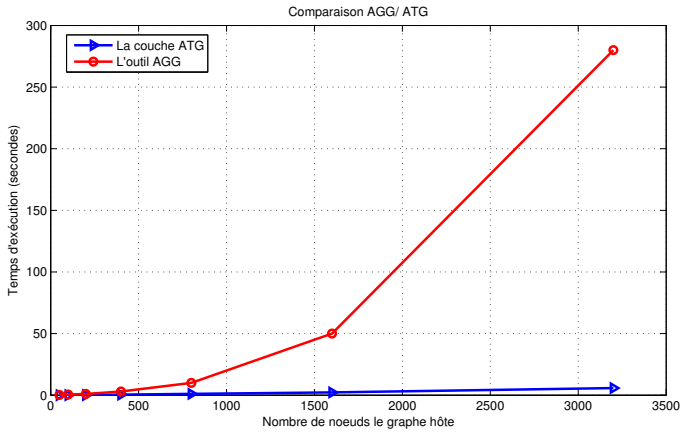


FIG. 5.9 – Comparaison des temps d'exécution entre AGG et l'ATG

5.4 Deuxième couche logicielle : le Moteur de Transformation de Graphes (MTG)

Le moteur de transformation de graphes (MTG) constitue la deuxième couche logicielle implémentée. Son objectif est de permettre l'utilisation de l'algorithme de transformation de graphes en considérant un ensembles de règles de réécriture et un ensemble de graphes hôtes afin de coller à la spécification du méta-modèle. En effet, nous considérons des ensembles de règle pour la définition des styles architecturaux et pour les transformations verticales et horizontales. D'un autre côté, la prise en compte de différents niveaux architecturaux (i.e. telle que cela a été illustré dans le chapitre3) nécessite de considérer plusieurs graphes hôtes.

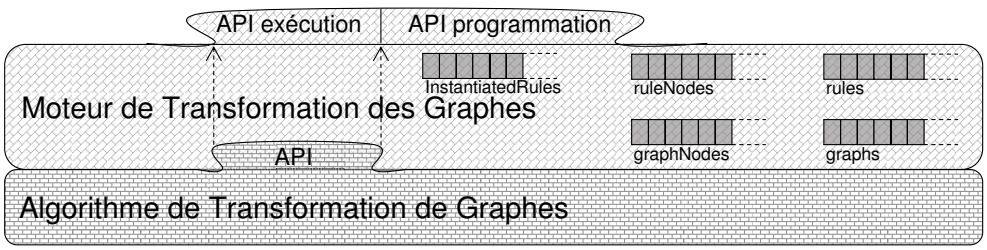


FIG. 5.10 – Les deux premières couches logicielles

Le MTG possède une API C++ que nous classons en deux parties. La première partie que nous appelons "*API de programmation*" permet, pendant l'exécution, de réaliser deux objectifs : 1) initialiser les graphes hôtes et les règles de réécriture considérés en construisant les nœuds et les arcs ainsi que leurs labels et leurs étiquettes, et 2) mettre à jour la liste des règles et des graphes en introduisant de nouvelles règles et de nouveaux graphes, et en supprimant des graphes et des règles des listes considérées.

La deuxième partie que nous appelons "*API d'exécution*" permet de vérifier l'applicabilité d'une règle de réécriture à un graphe considérés par le système et de transformer ce graphe dans le cas où la règle de réécriture est applicable. Ces opérations sont réalisées en faisant appel à l'API de la couche relevant de l'ATG.

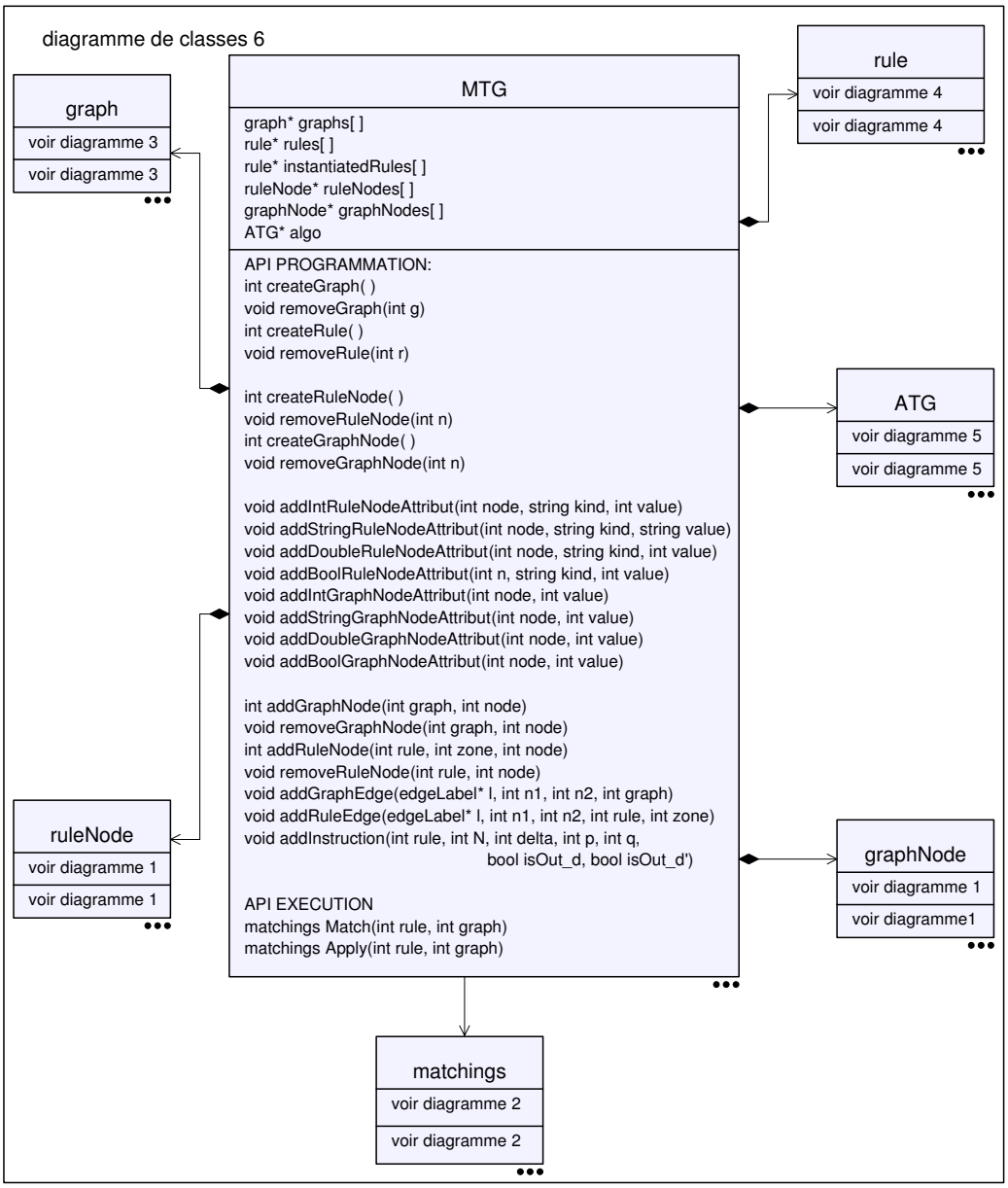


FIG. 5.11 – Diagramme de classes pour le MTG

La figure 5.11 présente un diagramme de classes qui illustre les liens de la classe MTG avec les autres classes de la couche logicielle inférieure. Nous notons, dans cette figure, les éléments des deux types de méthodes appartenant à l'API de cette classe. Dans l'API de programmation, nous remontons les constructeurs et les méthodes d'initialisation des classes relatives aux graphes hôtes (i.e. "*graph*"), aux règles de réécriture (i.e. "*rule*"), aux nœuds des graphes hôtes (i.e. "*graphNode*"), et aux nœuds des règles de réécriture (i.e. "*rule*"). Nous remontons les méthodes de la classe ATG pour construire l'autre partie concernant l'API d'exécution.

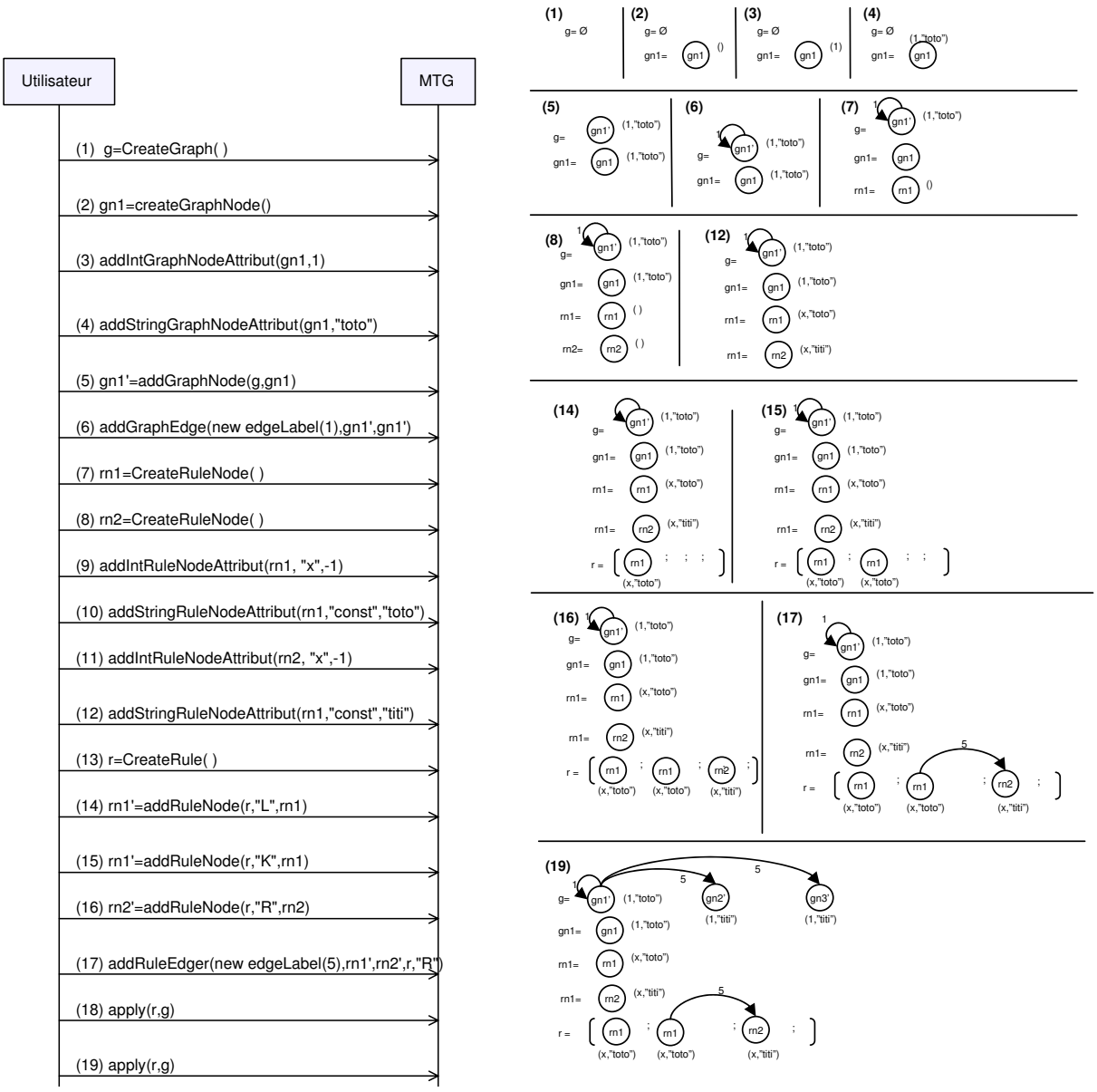


FIG. 5.12 – Un scénario pour les interactions impliquant le MTG et l'ATG

Nous présentons, dans la figure 5.12, un diagramme de séquence qui expose un scénario d'utilisation de l'API offerte par le moteur de transformation de graphes. Nous présentons, en parallèle, les différents éléments construits grâce à l'API de programmation (étapes de (1) à (17)). D'un côté, nous construisons le graphe hôte g ainsi que les nœuds et leurs labels et les arcs et leurs étiquettes qui le constituent (étapes de (1) à (6)). Et d'un autre côté nous construisons la règle de réécriture r et les nœuds et les arcs qui constituent ses différentes zones (étapes de (7) à (17)). Nous présentons ensuite le résultat obtenu après utilisation de l'API d'exécution en appliquant deux fois la règle r au graphe g (étapes (18) et (19)).

5.5 Le Protocole de Reconfiguration de l'Architecture (PRA)

Nous implémentons, au dessus du moteur de transformation de graphes, une brique logicielle permettant de définir le protocole de reconfiguration de l'architecture (PRA) tel que nous l'avons spécifié dans le chapitre 2.

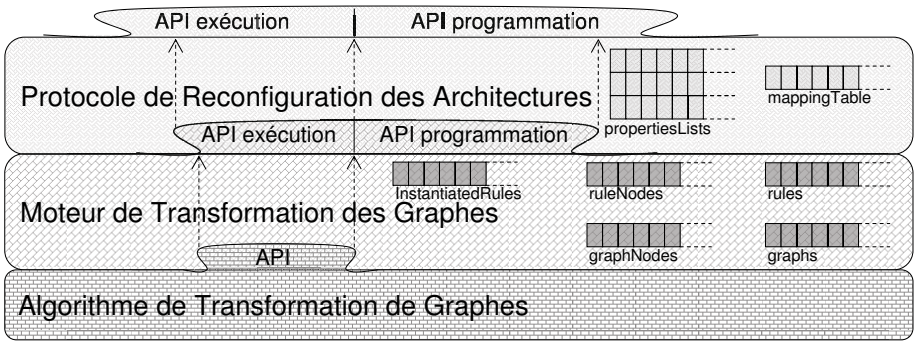


FIG. 5.13 – Les trois couches implémentées

Nous considérons, par conséquent, un nouveau type correspondant aux événements de reconfiguration ainsi que les structures permettant de décrire les politiques d'instanciation. Ceci a pour objectif de répercuter les valeurs transportées par les événements de reconfiguration sur les différents labels des nœuds des règles de réécriture. La classe "*instantiator*" permet de spécifier les variables concernées par les affectations. Nous distinguons, par conséquent, quatre tableaux regroupant ces variables selon leurs types. Pour chacune de ces variables, nous désignons la position de la valeur qui doit lui être affectée. Si nous prenons par exemple une instance I de cette classe telle que $intVariables = [x, y]$ et $intValues = [4, 3]$, cela caractérise une instanciation où nous affecterons la valeur du quatrième et du troisième paramètre entier transporté par l'événement respectivement aux variables x et y [90].

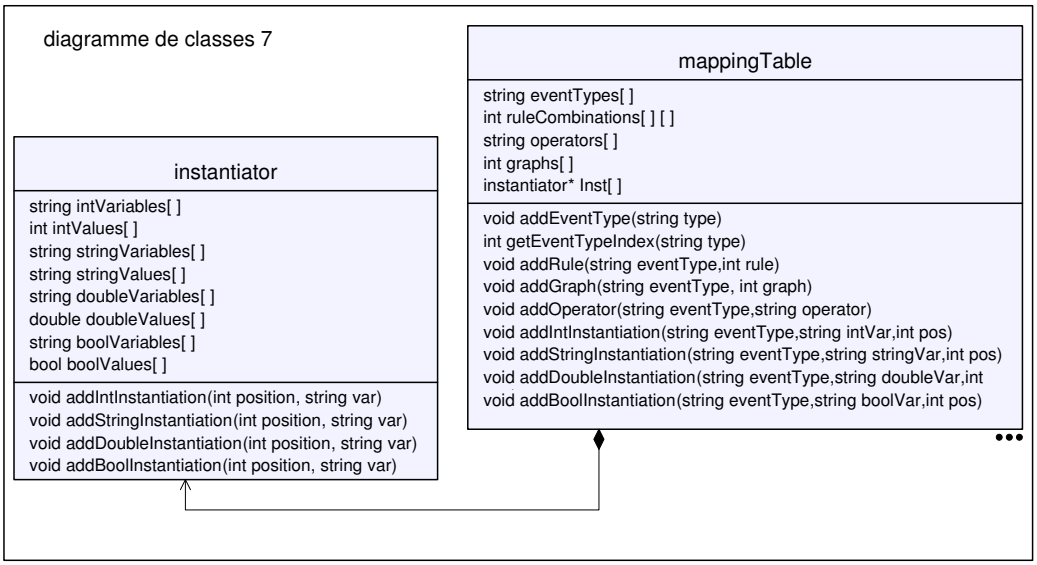


FIG. 5.14 – Règles d'instanciation et table de correspondance entre événements, règles et graphes

Nous introduisons, aussi, une classe supplémentaire permettant de définir la correspondance entre les événements de reconfiguration et les combinaisons des règles de reconfiguration. Pour chaque type

[90] e.g. si nous avons un événement e1(1,5,"titi",4,"toto",7), cela impliquerait l'affectation de la variable x par la valeur 7 et la variable y par la valeur 4.

d'événements, nous définissons la combinaison de règles à appliquer (i.e. grâce aux deux attributs "*ruleCombinations*" et "*operators*") et sur quel graphe il faut l'appliquer. L'attribut "*inst*" permet d'instancier correctement, pour chaque type d'événement, les labels des règles appartenant aux combinaisons à appliquer.

Dans la figure 5.15, nous donnons un diagramme de classes qui implique la caractérisation et l'exécution du protocole de reconfiguration des architectures (PRA). Nous remontons, à ce niveau, les API de programmation et d'exécution définies au niveau du MTA. Nous récupérons, ainsi, les méthodes permettant de construire les graphes hôtes et les règles de réécriture.

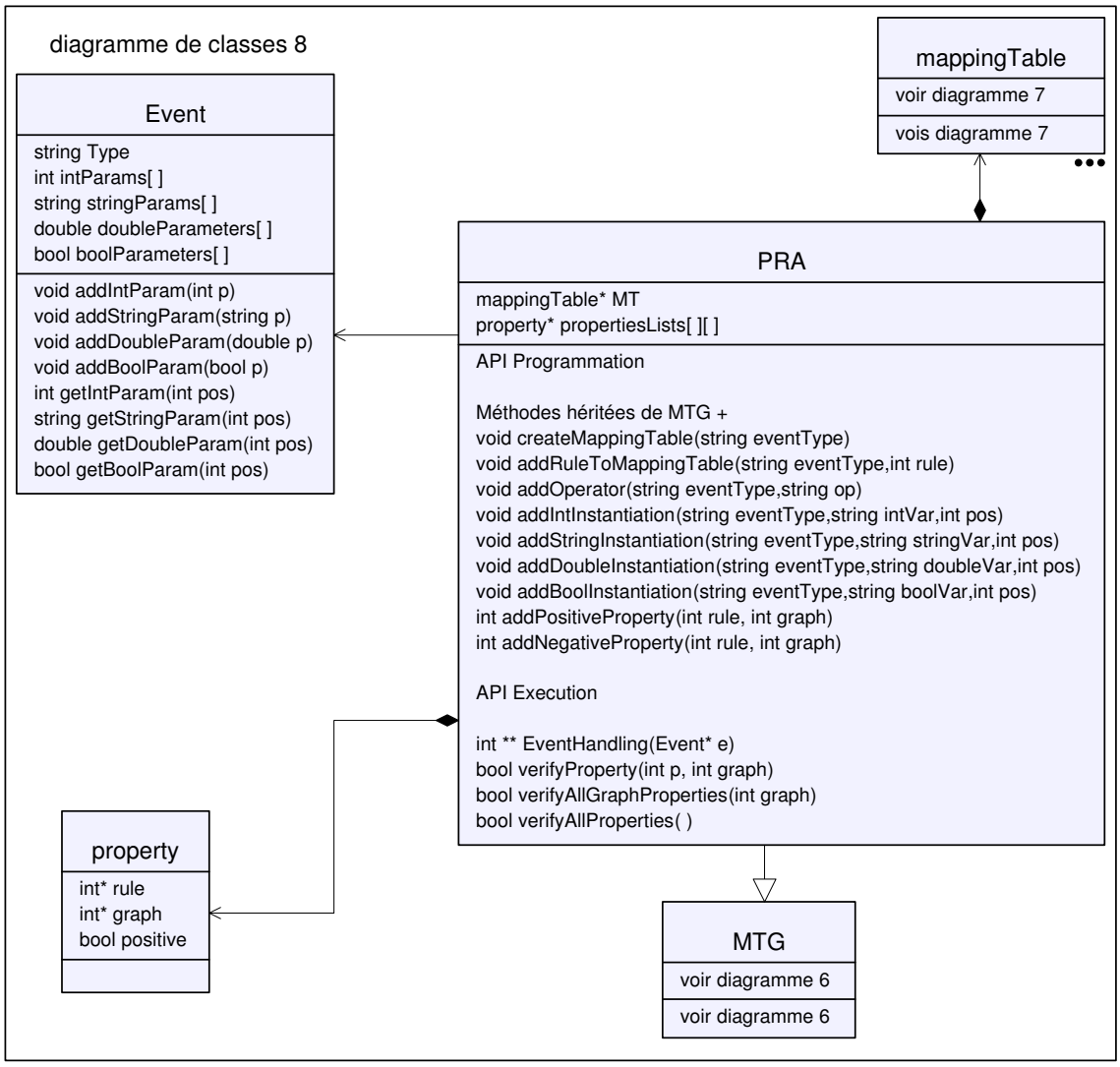


FIG. 5.15 – Le protocole de reconfiguration

Nous enrichissons, par la suite, l'API de programmation par des méthodes permettant de construire la correspondance entre événements, combinaisons de règles et graphes, et des méthodes qui définissent les liens entre les paramètres transportés par les événements et les labels des règles. Cette API permet d'actualiser, pendant l'exécution, le protocole de reconfiguration. Par exemple, nous pouvons prendre

en compte, sans avoir à arrêter l'application ni à la recompiler, un nouvel événement, et définir les combinaisons de reconfiguration qui lui correspondent. Cette API permet aussi de modifier, pendant l'exécution, les actions de reconfiguration relatives à un événement donné.

L'API d'exécution concernera principalement, en plus de celle héritée de la couche inférieure, deux méthodes. La première méthode (i.e. "*eventHandling*") correspond au traitement des événements en instanciant correctement les règles de réécriture et en appliquant la combinaison correspondante sur le graphe désigné par la table de correspondance. La deuxième méthode concerne la vérification des propriétés architecturales en testant l'applicabilité des règles (positives et négatives correspondantes). Nous disposons de trois options permettant 1) ou bien de cibler une propriétés particulière, ou 2) de vérifier la validité de l'ensemble des propriétés définies pour un graphe, ou 3) de vérifier la validité de toutes les propriétés définies pour tous les graphes du système.

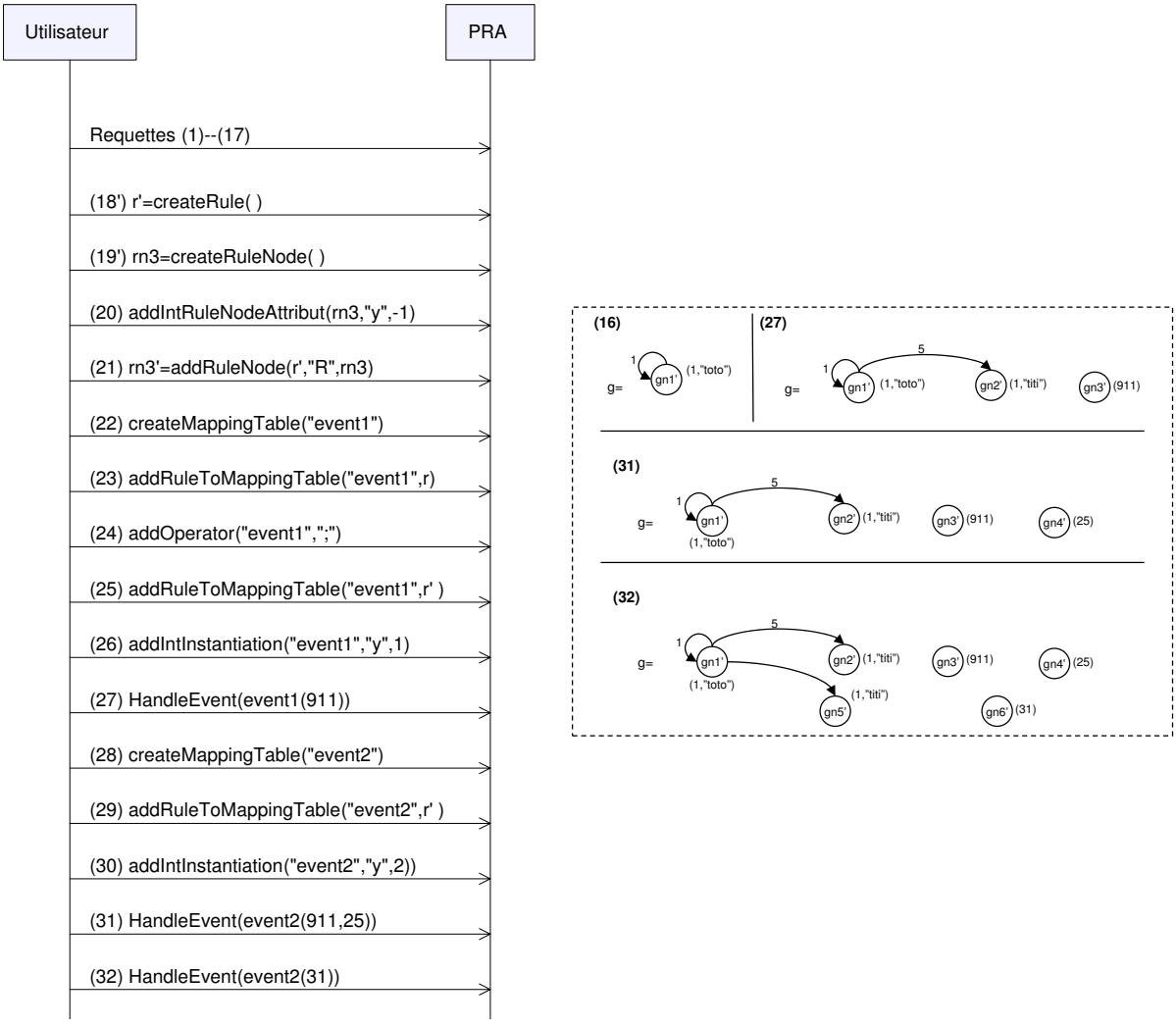


FIG. 5.16 – Le protocole de reconfiguration

La figure 5.16 présente, sous forme de diagramme de séquence, un scénario d'utilisation de la brique logicielle correspondant à la programmation et à l'exécution du PRA. En considérant les étapes (1) à (17) présentées dans la figure, nous complétons le processus de définition du protocole de reconfiguration par les étapes (18') jusqu'à (26). Les requêtes de (22) à (26) permettent de définir le type d'événements "*event1*". À cet événement correspondra la combinaison "$r; r'$". Ces requêtes définissent aussi une règle d'instanciation qui implique que la variable y sera affectée par le premier paramètre entier que trans-

porte l'événement (requête (26)). Les requêtes de (28) à (30) permettent de considérer un nouveau type d'événement "*event2*". À cet événement, correspond la combinaison constituée de l'unique règle r'. La règle d'instanciation spécifiée implique que la variable y est affectée par le second paramètre entier de l'événement.

5.6 Conclusion et perspectives

Nous avons présenté, dans ce chapitre, les trois couches logicielles que nous avons développées pour la mise en œuvre de notre méta-modèle. Nous avons implémenté les premiers algorithmes de recherche d'homomorphismes et de transformation de graphes présentés dans le chapitre 4.

Ensuite, nous avons implémenté une couche intermédiaire qui servira de base pour la simulation et gestion de l'évolution des architectures. Enfin, nous avons présenté la mise en œuvre d'un protocole de reconfiguration programmable. Nous avons introduit les différentes facettes de cette implémentation impliquant : 1) la définition du protocole par la spécification de ces éléments comprenant les graphes initiaux, les règles et les combinaisons de reconfiguration ainsi que les événements, et les tables de correspondance, 2) l'exécution de ce protocole et la génération des actions de reconfiguration à réaliser en réaction à la production des événements de reconfiguration, et 3) le contrôle de la reconfiguration par la définition et la vérification de la validité des propriétés architecturales.

Afin de pouvoir intégrer les trois briques logicielles présentées ici avec des applications Java, nous avons développée une API Java grâce à la technologie Java Native Interface (JNI)[JNI06]. Cette option présente deux principaux avantages : 1) nous évitons de tout réécrire en java, et 2) nous obtenons de meilleurs performances que si nous avons tout réécrit. Au final, nous disposons de deux API Java et C++ permettant aux trois briques logicielles d'être intégrées avec des programmes utilisant les deux langages.

Les perspectives de ces travaux d'implémentation concernent différents axes. Le premier axe concerne l'intégration de l'outil avec une application concrète afin de tester le passage à l'échelle de l'approche. Nous avons développé, en se basant sur une approche qui utilise les intercepteurs, un composant logiciel dédié aux Services Web permettant d'exécuter les actions élémentaires sur les entités logicielles de l'architecture (e.g. supprimer, introduire, remplacer des Services Web)[Del06b]. Nous projetons d'intégrer ce composant et le PRA avec des applications de revue coopératives et de magasin alimentaire développés dans le cadre du projet européen WS-DIAMOND. Nous planifions de réaliser des expérimentations sur la grille de calcul GRID5000 [GRI06] afin d'éprouver l'approche et observer ses performances dans le cas où l'application comprend un grand nombre de services inter-connectés.

Un deuxième axe concerne le développement de briques additionnelles permettant de réaliser des expérimentations et des simulations de manière automatique en définissant des paramètres de génération pour les entrées de simulation de manière générique. Le but est d'offrir un outil permettant de valider une spécification du protocole de reconfiguration par de larges simulations en se basant sur la définition de propriétés architecturales[91].

Un autre axe est de ré-implémenter la couche la plus basse en utilisant l'algorithme de transformation de graphes basé sur la mise à jour des arbres de recherche, et de confronter, par des expérimentation, l'efficacité des deux approches. À terme, l'objectif est d'offrir à l'utilisateur la possibilité de choisir entre les deux algorithmes (et éventuellement de basculer de l'un vers l'autre) tout en gardant la même interface pour les deux couches supérieures.

[91] au lieu de réécrire ce simulateur de manière ad-hoc et pour chaque applications comme nous le faisons actuellement.

Conclusion

1 Rappel des contributions

Nous avons abordé, dans ce manuscrit, le corps des travaux et des contributions réalisés dans le cadre de la thèse.

Dans un premier lieu, nous avons passé en revue l'état de l'art et étudié la littérature concernant les domaines de recherche abordés. Nous avons introduit les concepts relatifs aux architectures dynamiques en nous basant sur les standards et les travaux de recherche. Puis, nous avons présenté les langages de description des architectures les plus utilisés et les plus publiés. Nous avons mis l'accent sur le traitement et les mécanismes de spécification définis, pour chacun de ces langages, afin d'adresser la problématique des architectures dynamiques et de l'adaptabilité architecturale. Nous avons, ainsi, mis en lumière les forces et les insuffisances de ces différentes approches.

D'un autre côté, nous avons présenté les formalismes de graphes et de grammaires de graphes et les différents mécanismes pour la caractérisation rigoureuse de leur réécriture. Nous avons introduit les approches algorithmiques classiques pour la recherche d'homomorphismes de graphes dans un contexte considérant des modèles basiques.

Dans un deuxième chapitre, nous avons défini un méta-modèle qui traite plusieurs problématiques identifiées de l'adaptation architecturale. Ce méta-modèle utilise des formalismes basés sur les grammaires de graphes offrant à la fois une caractérisation visuelle et des spécifications mathématiques rigoureuses.

Les composants ainsi que leurs paramètres sont décrits par des nœuds possédant des labels multiples alors que les interdépendances liant ces composants sont décrites par des arcs étiquetés. Une instance de l'architecture est, par conséquent, décrite par un graphe énumérant les entités logicielles et leurs interdépendances.

La description des styles, correspondant à la caractérisation de l'ensemble des instances consistantes de l'architecture, est réalisée en utilisant des grammaires de graphes dont la structure et la sémantique des productions a été étendue en combinant les approches SPO et DPO et en considérant des labels variables et des productions paramétrées.

Nous avons démontré, en l'illustrant par un exemple simple, l'utilité de distinguer plusieurs niveaux architecturaux. Nous avons caractérisé les systèmes de transformations horizontales permettant de raffiner ou d'abstraire les descriptions en passant d'un niveau à un autre. Pour cela, nous avons étendu nos grammaires de graphes en introduisant des instructions de connexion très utiles pour traduire les interdépendances entre les différents niveaux, et en considérant des conditions d'application négatives pour augmenter la puissance d'expression des modèles.

Les actions élémentaires de transformation de l'architecture sont spécifiées par l'intermédiaire de règles de transformation de graphes intégrant aussi des instructions de connexion et des conditions d'application négatives. Nous avons enrichi cette spécification par l'introduction d'opérateurs logiques permettant de combiner les règles de transformation. Une combinaison correspond, alors, aux actions de reconfiguration à réaliser suite à la production d'un événement nécessitant une adaptation architecturale.

Le méta-modèle offre, d'autre part, le moyen de définir des propriétés architecturales exigeant la présence et/ou l'absence de sous-configurations dans les instances produites tout au long du processus de reconfiguration. La vérification de la validité de ces propriétés est réalisée par la recherche d'homomorphismes.

Différentes problématiques peuvent être abordées en se basant sur les aspects traités par le méta-modèle. Nous avons présenté, à la fin du chapitre 2, les vérifications et les validations qui peuvent être produites, à priori et pendant l'exécution, en se basant sur la consistance et la compatibilité des différentes descriptions formelles.

Nos approches ont été éprouvées dans différents travaux cités dans le manuscrit concernant des études réalisées dans le cadre des projets européens WS-DIAMOND et NetQoS et dans le cadre de travaux internes relevant du domaine des applications coopératives telles que l'édition partagée.

Afin de démontrer la faisabilité de l'approche, nous avons pris le choix de présenter un traitement détaillé du cas d'étude des opérations d'intervention d'urgence relevant du projet NetQoS. Dans le cadre de ce cas d'étude, nous avons considéré des exigences d'adaptabilité dues aux changements du contexte d'exécution, à la panne des composants et à des contraintes liées au provisionment de la QdS.

Pour les trois niveaux architecturaux considérés et correspondant aux niveaux application, middleware et réseau, nous avons élaboré les descriptions des styles architecturaux. Nous avons présenté les spécifications formelles des systèmes de raffinement et d'abstraction entre les deux premiers niveaux architecturaux ainsi que les protocoles de reconfiguration et les propriétés architecturales.

Dans les deux derniers chapitres, nous avons abordé l'implémentation d'outils relevant du méta-modèle. Nous avons, ainsi, défini un algorithme de recherche d'homomorphismes et de transformation de graphes qui supporte les extensions considérées pour les graphes et les règles de réécriture. Nous avons réalisé une analyse de sa complexité et donné des résultats expérimentaux concernant les temps d'exécution obtenus. Dans un deuxième temps, nous avons présenté un nouvel algorithme qui se base sur la mise à jour de l'arbre de recherche. Nous avons réalisé une étude théorique afin de comparer l'efficacité des deux algorithmes. Les résultats obtenus expriment, pour le deuxième algorithme, un gain conséquent.

Le dernier chapitre aborde les réalisations logicielles implantées durant la thèse. Ces réalisations touchent l'implémentation d'une première couche relative aux premiers algorithmes de recherche d'homomorphismes et de transformation de graphes définis dans le manuscrit. Une comparaison expérimentale a permis de montrer l'efficacité de cette implémentation par rapport à un des outils les plus reconnus, en l'occurrence, AGG.

Une deuxième couche logicielle a été définie afin de considérer l'ensemble des règles et des graphes impliqués dans les différents points de vues de description. Une dernière brique logicielle, permettant la programmation générique du protocole de reconfiguration, est aussi présentée.

Nous avons défini, pour les trois couches, des API C++ classées en API de programmation et en API d'exécution. En complément, nous avons aussi développé les API Java correspondantes en utilisant la technologie JNI.

2 Perspectives

Les perspectives des travaux présentés dans ce mémoire suivent différents axes de réflexion et se situent dans différents contextes de recherche.

Dans le cadre du projet WS-DIAMOND, nous positionnons la reconfiguration des architectures du point de vue de la réparation dans les applications à base de Services Web et nous considérons principalement le niveau architectural correspondant au niveau application.

Dans ce cadre, d'autres aspects sont considérés par les autres participants et en particulier les actions de réparation comportementales au niveau *workflow*. Ces actions impliquent, par exemple, une réémission des requêtes et des actions de compensation au niveau des processus BPEL. Un des objectifs est de coordonner les actions au niveau architectural et au niveau comportemental afin d'éviter des sur-réactions de l'application ou l'exécution d'actions qui s'opposent ou s'annihilent.

Dans le cadre du projet NetQoS, la reconfiguration est surtout un moyen pour le provisionnement de la QdS considérée principalement aux niveaux middleware et transport. La coordination de la reconfiguration entre les différents niveaux architecturaux constitue la perspective principale de nos travaux dans ce cadre.

Un des objectifs est de définir des outils qui produisent toutes les instances correspondant à une configuration de l'architecture au niveau le plus haut, évaluer la QdS pour chaque instance, extraire la ou les instances qui maximisent cette QdS, et finalement produire les actions de reconfiguration à exécuter pour atteindre cette instance maximale.

D'autre perspectives au niveau du développement de l'outil ont été présentés dans le chapitre 5. Elles concernent principalement l'implémentation de l'algorithme amélioré de transformation de graphes et la réalisation de briques logicielles permettant la validation par la simulation des modèles. Des études expérimentales sur la faisabilité à grande échelle utilisant la grille de calcul GRID 5000 sont programmées à court terme.

À moyen terme, des travaux théoriques sont planifiés concernant, en partie, la définition d'algorithmes et d'approches génériques afin de vérifier la compatibilité des points de vues de description, et la préservation des propriétés architecturales.

Annexe A

Preuves pour le Chapitre 4

A.1 Preuve du lemme 1

Calculons un équivalent de :

$$\sum_{i=1}^{n} A_m^i$$

Nous avons :

$$\begin{aligned}
\sum_{i=1}^{n} A_m^i &= \sum_{i=1}^{n} \frac{m!}{(m-i)!} \\
&= m! \sum_{i=1}^{n} \frac{1}{(m-i)!} \\
&= m! \sum_{i=m-n}^{m-1} \frac{1}{i!} \\
&= m! (\sum_{i=0}^{m-1} \frac{1}{i!} - \sum_{i=0}^{m-n-1} \frac{1}{i!})
\end{aligned}$$

Calculons maintenant un équivalent de $\sum_{i=1}^{k} \frac{1}{i!}$. En utilisant la formule de Taylor avec reste intégral[92] et en considérant la fonction e^t de classe C^∞ sur l'interval $[0,1]$, nous obtenons :

$$\sum_{i=0}^{K} \frac{1}{i!} = e - \frac{e}{K!} \int_0^1 t^K e^{-t}\, dt \qquad \text{(I)}$$

Cherchons une expression équivalente au second terme correspondant à l'intégral. Nous avons :

$$\begin{aligned}
\int_0^1 t^K e^{-t}\, dt &= \frac{1}{K+1} \int_0^1 ((K+1)t^K) e^{-t}\, dt \\
&= \frac{1}{K+1} \int_0^1 (t^{K+1})' e^{-t}\, dt
\end{aligned}$$

[92] Soit f une fonction dérivable $n+1$ sur l'interval $[a,b]$ avec f^{n+1} est continue sur $[a,b]$ (i.e. f est de classe C^{n+1}), alors : $f(b) = f(a) + \sum_{k=1}^{n} \frac{f^{(k)}(a)}{k!}(b-a)^k + \frac{1}{k!} \int_a^b (b-t)^n f^{n+1}(t)\, dt$

Nous obtenons en utilisant une *intégration par parties* [93]

$$\begin{aligned}\int_0^1 t^K e^{-t}\,dt &= \frac{1}{K+1}([t^{K+1}e^{-t}]_0^1 + \int_0^1 t^{K+1}e^{-t}\,dt) \\ &= \frac{1}{K+1}(e^{-1} + \int_0^1 t^{K+1}e^{-t}\,dt)\end{aligned}$$

Considérons

$$S_K = \int_0^1 t^K e^{-t}\,dt$$

Nous obtenons

$$S_K = \frac{e^{-1}}{K+1} + \frac{S_{K+1}}{K+1}$$

Et

$$S_{K+1} = \frac{e^{-1}}{K+2} + \frac{S_{K+2}}{K+2}$$

Et ensuite

$$S_K = \frac{e^{-1}}{K+1} + \frac{\frac{e^{-1}}{K+2} + \frac{S_{K+2}}{K+2}}{K+1}$$

Puisque

$$S_K \leq S_{K-1} \ , \ \forall 1 \leq K$$

Et puisque

$$\forall t \in [0,1], \text{ nous avons, } t^K e^{-t} \leq t^{K-1}e^{-t}$$

Alors

$$\int_0^1 t^K e^{-t}\,dt \leq \int_0^1 t^{K-1}e^{-t}\,dt$$

Et par conséquent

$$\int_0^1 t^K e^{-t}\,dt \leq \int_0^1 t^0 e^{-t}\,dt$$

Et ensuite :

$$\int_0^1 t^K e^{-t}\,dt \leq 1 - e^{-1}$$

Nous avons

$$S_K = \frac{e^{-1}}{K+1} + \frac{e^{-1}}{(K+1)(K+2)} + \frac{S_{K+2}}{(K+1)(K+2)}$$

[93] La formule-type est la suivante, où f et g sont deux fonctions dérivables, de dérivées continues et a et b deux réels de leur intervalle de définition. $\int_a^b f(x)g'(x)\,dx = ([f(x)g(x)]_a^b + \int_a^b f'(x)g(x)\,dx)$

Et

$$0 \leq S_{K+2} \leq 1 - e^{-1}$$

Alors

$$\frac{e^{-1}}{K+1} + \frac{e^{-1}}{(K+1)(K+2)} \leq S_K$$

And

$$S_K \leq \frac{e^{-1}}{K+1} + \frac{e^{-1}}{(K+1)(K+2)} + \frac{1-e^{-1}}{(K+1)(K+2)}$$

Ce qui donne

$$\frac{e^{-1}}{K+1} + \frac{e^{-1}}{(K+1)(K+2)} \leq S_K \leq \frac{e^{-1}}{K+1} + \frac{1}{(K+1)(K+2)}$$

Et ensuite

$$S_K = \frac{e^{-1}}{K} + o(\frac{1}{K}) \qquad \text{(II)}$$

(I) et (II) impliquent

$$\sum_{i=0}^{K} \frac{1}{i!} = e - \frac{e}{K!}(\frac{e^{-1}}{K} + o(\frac{1}{K})) \qquad \text{(III)}$$

Nous avons

$$\sum_{i=1}^{n} A_m^i = m!(\sum_{i=0}^{m-1} \frac{1}{i!} - \sum_{i=0}^{m-n-1} \frac{1}{i!}) \qquad \text{(IV)}$$

Nous obtenons d'après (III)

$$\begin{aligned}
\sum_{i=1}^{n} A_m^i &= m![(e - \frac{e}{m-1!}(\frac{e^{-1}}{m-1} + o(\frac{1}{m}))) \\
&\quad -(e - \frac{e}{(m-n-1)!}(\frac{e^{-1}}{m-n-1} + o(\frac{1}{m-n-1})))] \\
&= m![\frac{e}{(m-n-1)!}(\frac{e^{-1}}{m-n-1} + o(\frac{1}{m-n-1})) - \frac{e}{m-1!}(\frac{e^{-1}}{m-1} + o(\frac{1}{m-1}))] \\
&= m![\frac{1}{(m-n-1)!(m-n-1)} + o(\frac{1}{(m-n-1)!(m-n-1)}) \\
&\quad -\frac{1}{(m-1)!(m-1)} - o(\frac{1}{(m-1)!(m-1)})]
\end{aligned}$$

Nous avons

$$\frac{1}{(m-1)!(m-1)} = o(\frac{1}{(m-n-1)!(m-n-1)})$$

Et alors

$$\sum_{i=1}^{n} A_m^i = m![\frac{1}{(m-n-1)!(m-n-1)} + o(\frac{1}{(m-n-1)!(m-n-1)})]$$

et ensuite[94]

$$\sum_{i=1}^{n} A_m^i \simeq \frac{m!}{(m-n-1)!(m-n-1)} \qquad \text{(V)}$$

Calculons maintenant un équivalent pour la fraction précédente. Si nous prenons M=m et N=n+1,

$$\frac{m!}{(m-n-1)!(m-n-1)} = \frac{M!}{(M-N)!(M-N)}$$

[94] $\simeq$ est utilisé pour indiquer que le rapport entre les deux expressions tend vers 1 quand m et n tendent vers l'infini

La formule de *Stirling* donne pour K très grand

$$K! \simeq \sqrt{2\pi K}(\frac{K}{e})^K$$

Nous obtenons donc

$$\begin{aligned}\frac{M!}{(M-N)!(M-N)} &\simeq \frac{\sqrt{2\pi M}(\frac{M}{e})^M}{\sqrt{2\pi(M-N)}(\frac{M-N}{e})^{M-N}(M-N)} \\ &\simeq \sqrt{\frac{M}{M-N}} \times \frac{1}{M-N} \times e^{-N} \times \frac{M^M}{(M-N)^{M-N}} \qquad \text{(VI)}\end{aligned}$$

Calculons maintenant une expression équivalente à

$$\frac{M^M}{(M-N)^{M-N}}$$

Nous avons,

$$\begin{aligned}\frac{M^M}{(M-N)^{M-N}} &= \frac{\exp(M\ln(M))}{\exp((M-N)\ln(M-N))} \\ &= \exp(M\ln(M) - (M-N)\ln(M-N)) \\ &= \exp(N\ln(M) + (M-N)(\ln(M) - \ln(M-N)) \\ &= \exp(N\ln(M) + (M-N)\ln(\frac{M}{M-N})) \\ &= \exp(N\ln(M) + (M-N)\ln(1+\frac{N}{M-N}))\end{aligned}$$

Nous supposons que $M \gg N$. Cette supposition est motivée par le fait, qu'en pratique, les graphes d'entrée sont beaucoup plus grand en taille que les graphes modèles. [95]Ainsi, nous avons $\frac{N}{M-N} \longmapsto 0$ quand N et M tendent vers l'infini. Et alors

$$\ln(1+\frac{N}{M-N}) \simeq \frac{N}{M-N}$$

Et

$$\begin{aligned}\frac{M^N}{(M-N)^{M-N}} &\simeq \exp(N\ln(M) + (M-N)\frac{N}{M-N}) \\ &\simeq \exp(N\ln(M) + N) \\ &\simeq \exp(N\ln(M))\exp(N) \\ &\simeq M^N e^N \qquad \text{(VII)}\end{aligned}$$

(VI) et (VII), donnent

$$\begin{aligned}\frac{M!}{(M-N)!(M-N)} &\simeq \sqrt{\frac{M}{M-N}}\frac{1}{M-N}e^{-N}M^N e^N \\ &\simeq \sqrt{\frac{M}{M-N}}\frac{1}{M-N}M^N\end{aligned}$$

[95]Sans cette supposition nous obtenons, au lieu d'un équivalent, une borne max de la complexité de l'algorithme puisque $ln(1+x) \leq x$, $\forall x \geq 0$.

Nous avons $M = m$ et $N = n + 1$, alors

$$\begin{aligned} \frac{m!}{(m-n-1)!(m-n-1)} &\simeq \sqrt{\frac{m}{m-n-1}}\frac{1}{m-n-1}m^{n+1} \\ &\simeq \sqrt{\frac{m}{m-n}}\frac{1}{m-n}m^{n+1} \qquad \text{(VIII)} \end{aligned}$$

Et ensuite

$$\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{A_m^i} \simeq \sqrt{\frac{\boldsymbol{m}}{\boldsymbol{m-n}}}\frac{\mathbf{1}}{\boldsymbol{m-n}}\boldsymbol{m^{n+1}} \qquad \text{(IX)}$$

□

A.2 Preuve du lemme 2

Calculons une expression équivalente pour :

$$\sum_{i=0}^{n} iA_m^i$$

Considérons

$$T_{n,m} = \sum_{i=0}^{n} iA_m^i$$

Nous avons

$$\begin{aligned} T_{n,m} &= \sum_{i=0}^{n} i\frac{m!}{(m-i)!} \\ &= m!(\sum_{i=0}^{n}\frac{i}{(m-i)!}) \\ &= m!(\sum_{i=m}^{m-n}\frac{m-i}{i!}) \\ &= m!(\sum_{i=m}^{m-n}\frac{m}{i!} - \sum_{i=m}^{m-n}\frac{i}{i!}) \\ &= m!(m\sum_{i=m}^{m-n}\frac{1}{i!} - \sum_{i=m}^{m-n}\frac{1}{i-1!}) \\ &= m!(m\sum_{i=m-n}^{m}\frac{1}{i!} - \sum_{i=m-n-1}^{m-1}\frac{1}{i!}) \\ &= m!(\frac{m}{m!} + m\sum_{i=m-n}^{m-1}\frac{1}{i!} - \frac{1}{(m-n-1)} - \sum_{i=m-n}^{m-1}\frac{1}{i!}) \\ &= m!(\frac{m}{m!} - \frac{1}{(m-n-1)!} + (m-1)\sum_{i=m-n}^{m-1}\frac{1}{i!}) \\ &= m - \frac{m!}{(m-n-1)!} + (m-1)m!\sum_{i=m-n}^{m-1}\frac{1}{i!}) \end{aligned}$$

Sachant que

$$\sum_{i=1}^{n} A_m^i = m!\sum_{i=m-n}^{m-1}\frac{1}{i!}$$

Nous obtenons

$$T_{n,m} = m - \frac{m!}{(m-n-1)} + (m-1)\sum_{i=1}^{n} A_m^i$$

Nous avons

$$m \ll \frac{m!}{(m-n-1)!}$$

Et, par la formule (IX), nous avons

$$\sum_{i=1}^{n} A_m^i \simeq \sqrt{\frac{m}{m-n}}\frac{1}{m-n}m^{n+1}$$

Alors

$$T_{n,m} = -\frac{m!}{(m-n-1)!} + \sqrt{\frac{m}{m-n}}\frac{1}{m-n}m^{n+1}$$

nous avons par (VIII)

$$\frac{m!}{(m-n-1)!} \simeq \sqrt{\frac{m}{m-n}}m^{n+1}$$

Donc, nous obtenons

$$\begin{aligned} T_{n,m} &= -\sqrt{\frac{m}{m-n}}m^{n+1} + (m-1)\sqrt{\frac{m}{m-n}}\frac{1}{m-n}m^{n+1} \\ &= \sqrt{\frac{m}{m-n}}m^{n+1}(-1+\frac{m-1}{m-n}) \\ &= \sqrt{\frac{m}{m-n}}m^{n+1}(\frac{-m+n+m}{m-n}) \\ &= \sqrt{\frac{m}{m-n}}m^{n+1}\frac{n}{m-n} \end{aligned}$$

Et finalement,

$$\boldsymbol{T_{n,m} \simeq \sqrt{\frac{m}{m-n}}(\frac{n}{m-n})m^{n+1}}$$

□

Résumé

L'adaptabilité des applications logicielles peut être séparée en deux catégories. La première concerne l'adaptation comportementale appelée aussi adaptation algorithmique. Cette adaptation traite la redéfinition du comportement de l'application et de ses composants et implique, par exemple, l'introduction d'une nouvelle méthode dans l'interface d'un composant ou le changement du protocole d'orchestration qui coordonne un ensemble de services.

Nos travaux, que nous classons dans une deuxième catégorie, traitent l'adaptation structurelle et considèrent une reconfiguration au niveau architectural. Ce type de reconfiguration traite l'organisation de l'architecture et consiste, par exemple, à remplacer un composant défaillant par un autre composant qui possède les mêmes fonctionnalités ou rediriger un client d'un service qui ne respecte pas le contrat de QdS vers un service susceptible d'offrir de meilleures garanties.

Dans ce mémoire, nous définissons un méta-modèle relatif à la description et la gestion automatique des architectures dynamiques. Les instances des architectures sont décrites par des graphes étendus où les composants (ou les services) sont représentés par des nœuds, et les interdépendances (e.g. les connexions, les relations de contrôle ...etc) sont décrites par des arcs. Les styles architecturaux sont spécifiés par des grammaires de graphes étendues. Le méta-modèle admet des descriptions en considérant différents niveaux d'abstraction et offre des mécanismes pour raffiner ou abstraire les descriptions selon des points de vues spécifiques. Il permet, aussi, de décrire le protocole de gestion de l'architecture et de caractériser les propriétés architecturales à préserver pour chaque niveau architectural considéré.

Nous avons développé un algorithme de recherche d'homomorphismes de graphes et un algorithme de transformation de graphes pour les grammaires de graphes étendus définis pour notre méta-modèle. L'analyse de complexité des algorithmes ainsi que les résultats expérimentaux obtenus ont permis de conclure à leur efficacité. Une deuxième version des deux algorithmes a été définie profitant de la spécificité du contexte de la transformation de graphes. L'analyse de complexité de ces nouvelles versions donne des résultats encore plus performants pour le passage à l'échelle.

Notre approche a été éprouvée en l'appliquant à un cas d'étude dans le contexte des activités d'opération d'intervention d'urgence. L'application implique des groupes structurés de robots ou de personnel militaire qui disposent de ressources inégales en capacités de communication, en CPU et en énergie. Les besoins d'adaptabilité découlent des changements du contexte d'exécution, de l'occurrence de pannes et du provisionnement de la QdS.

Mots-clés: Adaptabilité architecturale, Architectures Dynamiques, Composants, SOA, Grammaires de graphes

Abstract

Adaptability for software applications can be separated into two categories. First relates to the behavioral adaptation also called algorithmic adaptation. This adaptation addresses the redefinition of the behavior of the application and its components and implies, for example, adding a new method into the interface of a component or updating the orchestration protocol that coordinates a set of services.

Second category, in which we can classify our work, relates to the structural adaptation and implies a reconfiguration at the architectural level. This kind of reconfiguration deals with the organization of the architecture and involve, for instance, the substitution of a failing component by another with same functionalities or the redirection for a customer of a service which does not respect the QoS contract towards a service likely to offer better guarantees.

In this thesis, we specify a meta-model relating to the description and the automatic management of dynamic architectures. Architecture instances are described by extended graphs where components (or services) are represented by vertices, and interdependencies (e.g. connections, relations of control ... etc) are described by edges. The architectural styles are specified by extended graph grammars. The meta-model considers descriptions by admitting various abstraction levels and offers mechanisms to either abstract or refine descriptions according to specific points of view. It, also, makes it possible to describe architecture management protocols and to characterize the architectural properties to be preserved for each considered architectural level.

We developed an algorithm for graph homomorphisms building and an algorithm for graph transformation in the context of extended graph grammars defined for our meta-model. Complexity analysis of these algorithms as well as the experimental results obtained made it possible to show their effectiveness. A second version of the two algorithms was defined benefiting from the specificity of our graph transformation context. Complexity analysis of these new versions gives results even more powerful when considering their scalability.

Our approach was applied to a case study in the context of the activities of emergency operations. The related system implies structured groups of robots or soldiers that have unequal resources for communication capacities, CPU and energy. The needs for adaptability rise from the changes in the execution context, from fault occurrence, and to achieve QoS provisioning.

Keywords: Architectural adaptation, Dynamic architectures, Components, SOA, Graph grammars

Publications de l'auteur

Revues internationales avec actes

▷ K. Guennoun, K. Drira and M. Diaz. Considering Topological Constraints for the Description of Dynamic Service-Oriented Orchestrated Architectures. International Journal of Computer Information Sciences. Vol. 5, NO. 1, April 2007, 45-51.

▷ C. Chassot, K. Guennoun, K. Drira, F. Armando, E. Exposito and A. Lozes. Towards autonomous management of QoS through model-driven adaptability in communication-centric systems[96]. International Transactions on Systems Science and Applications, Vol.2, N.3, pp.255-264.

▷ K. Guennoun and K. Drira. Using graph grammars for interaction style description. Application to service-oriented architectures. International Journal on Computer Systems Science Engineering, vol21, n4, July 2006, 293-299.

Revues internationales électroniques

▷ K.Guennoun, K.Drira and M.Diaz. Une approche orientée modèle pour la gestion des architectures logicielles distribuées dynamiques. Information Sciences for Decision Making, Article N°238, N°19, 2005

Conférences internationales avec actes et comité de lecture

▷ K. Guennoun, K. Drira and C. Chassot. Architectural Adaptability Management for Mobile Cooperative Systems. Accepted in the 1st International Conference on Multimedia and Ubiquitous Engineering (MUE'07), Seoul (Korea), April 2007.

▷ C. Chassot, K. Guennoun, K. Drira, F. Armando, E. Exposito and A. Lozes. Towards autonomous management of QoS through model-driven adaptability in communication-centric systems. In International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS2006), Erfurt (Germany), September 2006.

▷ C. Chassot, K. Guennoun, K. Drira, F. Armando, E. Exposito and A. Lozes. Architecture transformation and refinement for model-driven adaptability management. Application to QoS provisioning in group communication. In European workshop on software architecture, LNCS 4344, Nantes (France), September 2006. Springer Verlag.

[96] Ce papier apparaît en doublon aussi dans la partie conférences internationales puisque sa parution en revue est consécutive à sa sélection dans le cadre de la conférence SOAS'06

▷ K. Guennoun, K. Drira and M. Diaz. Adressing the conformity of dynamic and distributed architecture management protocols. In 6th IEEE International Symposium and School on Advanced Distributed Systems (ISSADS'2006), Guadalajara (Mexico), January 23-27 2006.

▷ K. Guennoun and K. Drira. Specifying reference styles for service orchestration and composition. In 1st International Workshop on Design of Service-Oriented Applications (WDSOA'05), Amsterdam (Netherlands), 12 December 2005, pp.67-74.

▷ G. Lussier, H. Waeselynck and K. Guennoun. Proof-guided testing : an experimental study. In 28th Annual International Computer Software and Applications Conference (COMPSAC'2004), Hong Kong (China), 28-30 September 2004, pp.528-533.

▷ K. Guennoun, K. Drira and M. Diaz. A proved component-oriented approach for managing dynamic software architectures. In 7th IASTED International Conference on Software Engineering and Applications, Marina del Rey (USA), 3-5 November 2003, pp.657-662.

Conférences Francophones avec actes et comité de lecture

▷ K. Guennoun, K. Drira and M. Diaz. Description des architectures pour une orchestration hiérarchique des applications orientées-services. In 6èmes Journées Scientifiques des Jeunes Chercheurs en Génie Electrique et Informatique (GEI'06), Sfax (Tunisie), Mars 2006, pp.135-142.

▷ K. Guennoun, K. Drira, M. Diaz. Une approche orientée modèle pour la gestion des architectures logicielles distribuées dynamiques. 8ème Conférence Maghrébine sur le Génie Logiciel et l'Intelligence Artificielle (MCSEAI'04), Sousse (Tunisie), 9-12 Mai 2004, pp.149-160

Manifestations d'audience nationale avec comité de lecture

▷ K. Guennoun, K. Drira and M. Diaz. Architectures logicielles distribuées dynamiques, caractérisation de quatre styles de base. In Journées "Formalisation des Activités Concurrentes" (FAC'2004), Toulouse (France), 9-10 Mars 2004, 13p.

Manifestations d'audience nationale

▷ K.Guennoun. Caractérisation formelle des styles architecturaux spécialisés. Application aux topologies hiérarchiques. 5ème Congrès de l'Ecole Doctorale Systèmes (EDSYS), Toulouse (France), 13-14 Mai 2004, 8p.

Délivrables WS-DIAMOND

▷ K.Guennoun, F. Moo-Meena and K. Drira. Requirements, application scenarios, overall architecture, and test/validation specication, common working environment and standards at Milestone M1, chapter 3. In Web Services - DIAgnosability, MONitoring and Diagnosis (WS-Diamond) IST-516933, Delivrable D1.1, pages 76-106, February 2006.

▷ F. Moo-Meena, K. Guennoun, K. Drira, L. Trave-Massuyes and X. Pucel. Specification of execution mechanisms and composition strategies for self-healing web services - phase 1. In Web Services - DIAgnosability, MONitoring and Diagnosis (WS-Diamond) IST-516933, Delivrable D3.1, pages 1-103, September 2006.

Bibliographie

[AAD06] AADL. *Avionics Architecture Description Language*. http ://www.sae.org/technical/ standards/AS5506, Last Consultation : October 2006.

[ACKM04a] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*, chapter II.8 : Service Composition, pages 245–294. Springer-Verlag, 2004.

[ACKM04b] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications, ISBN 3540440089*. Springer-Verlag, 2004.

[ADG97] R. Allen, R. Douence, and D. Garlan. Specifying dynamism in software architecture. In *Foundations of Component-Based Systems Workshop*, 1997.

[ADG98] R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In *1998 Conference on Fundamental Approaches to Software Engineering*, Lisbon, Portugal, March 1998.

[AGD97] Robert Allen, David Garlan, and Remi Douence. Specifying dynamism in software architectures. In *Proceedings of the Workshop on Foundations of Component-Based Software Engineering*, Zurich, Switzerland, September 1997.

[AGG06] *AGG home page*. http ://tfs.cs.tu-berlin.de/agg, Last Consultation : October 2006.

[All97] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 1997.

[BB76] H. G. Barrow and Rod M. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Inf. Process. Lett.*, 4(4) :83–84, 1976.

[ben06] *Performance comparison C++, C and Java*. http ://www.tommti-systems.de/main-Dateien/reviews/languages/benchmarks.html, Last Consultation : October 2006.

[BPE06] *Business Process Execution Language for Web Services Version 1.1*. http ://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel/, Last Consultation : October 2006.

[BR75] J.R. Bitner and E.M. Reingold. Backtrack programming techniques. *Communication ACM*, 18(11) :651–656, 1975.

[BSHL01] I. Ben-Shaul, O. Holder, and B. Lavva. Dynamic adaptation and deployment of distributed components in hadas. *IEEE Transactions on Software Engineering*, 27(9) :769–787, September 2001.

[Bun00] H. Bunke. Graph matching for visual object recognition. *Saptial vision*, 13 :333–340, 2000.

[C2S06] C2SADL. *C2 Software Architecture Description Language.* http ://www.isr.uci.edu/architecture/adl /SADL.html, Last Consultation : October 2006.

[CDK+02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web : an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2), March-April 2002.

[CGD+06a] C. Chassot, K. Guennoun, K. Drira, F. Armando, E. Exposito, and A. Lozes. Architecture transformation and refinement for model-driven adaptability management. application to QoS provisioning in group communication. In *European workshop on software architecture*, Nantes,France, September 2006.

[CGD+06b] C. Chassot, K. Guennoun, K. Drira, F. Armando, E. Exposito, and A. Lozes. Towards autonomous management of QoS through model-driven adaptability in communication-centric systems. In *International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS2006) (published in International Transactions on Systems Science and Applications, Vol.2, N°3, pp.255-264)*, Erfurt, Germany, September 2006.

[Cho56] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2 :113–124, 1956.

[Cho06a] *Web services choreography in practice.* http ://www-128.ibm.com/developerworks/webservices/library/ws-choreography/, Last Consultation : October 2006.

[Cho06b] *Web Services Choreography Working Group.* http ://www.w3.org/2002/ws/chor/, Last Consultation : October 2006.

[COT06] COTRE. *COmposants Temps REel (COTRE) official web site.* http ://www.laas.fr/cotre, Last Consultation : October 2006.

[CPT99] C. Canal, E. Pimentel, and J. M. Troya. Specification and refinement of dynamic software architectures. *Software Architecture, Kluwer Academic Publishers*, pages 107–126, 1999.

[Del06a] Requirements, application scenarios, overall architecture, and test/validation specification, common working environment and standards at Milestone M1, chapter 3. In *Web Services - DIAgnosability, MONitoring and Diagnosis (WS-Diamond) IST-516933, Delivrable D1.1*, pages 76–106, February 2006.

[Del06b] Specification of execution mechanisms and composition strategies for self-healing web services - phase 1. In *Web Services - DIAgnosability, MONitoring and Diagnosis (WS-Diamond) IST-516933, Delivrable D3.1*, pages 1–103, September 2006.

[EHP+96] W. Ellis, R. Hilliard, P. Poon, D. Rayford, T. Saunders, B. Sherlund, and R. Wade. Toward a recommended practice for architectural description. In 2^{nd} *IEEE International Conference on Engineering of Complex Computer Systems*, pages 21–25, Montreal, Canada, October 1996.

[EK90] H. Ehrig and H.-J. Kreowski, editors. *Graph grammars and their application to computer Science, ISBN 3-540-54478-X.* Springer-Verlag, 1990.

[EKL90] H. Ehrig, M. Korff, and M. Löwe. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In 4^{th} *International Workshop on Graph Grammars and Their Application to Computer Science, LNCS 532, ISBN 3-540-54478-X,*, pages 24–37, Bremen, Germany, March 1990. Springer-Verlag.

[ER97] J. Engelfriet and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, ISBN 981-02-2884-8*, chapter Node Replacement Graph Grammars, pages 1–94. World Scientific Publishing, 1997.

[FBB+03] J.M. Farines, B. Berthomieu, J.P. Bodeveix, P. Farail, M. Filali, P.Gaufillet, H. Hafidi, and F. Vernadat J.L. Lambert, P. Michel. The COTRE project : rigorous software development for real time systems in avionics. In *27th IFAC/IFIP/IEEE Workshop on Real Time programming*, Swiebodzin,Poland, May 14-17 2003.

[FH00] H. Fahmy and R. Holt. Software architecture transformations. In *International Conference on Software Maintenance*, pages 88–96, San Jose, CA, USA, October 2000.

[Gar06] D. Garlan. *Wright Web Site, Carnegie Mellon University.* http ://www-2.cs.cmu.edu/ able/wright, Last Consultation : October 2006.

[GB65] S.W. Golomb and L.D. Baumert. Backtrack programming. *Journal of ACM*, 12(4) :516–524, 1965.

[GDD05] K. Guennoun, K. Drira, and M. Diaz. Une approche orientée modèle pour la gestion des architectures logicielles distribués dynamiques. *Information Sciences for Decision Making*, (19), 2005.

[GDD06] K. Guennoun, K. Drira, and M. Diaz. Adressing the conformity of dynamic and distributed architecture management protocols. In *6th IEEE International Symposium and School on Advanced Distributed Systems (ISSADS'2006)*, Guadalajara, Mexico, January 23-27 2006.

[Geo02] I. Georgiadis. *Self-Organising Distributed Component Software Architectures.* PhD thesis, the Faculty of Engineering of the university of London, January 2002.

[GP95] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions On Software Engineering*, 21(4) :269–274, April 1995.

[GRI06] *Grid'5000 official web site.* http ://www.grid5000.fr, Last Consultation : October 2006.

[Gue06] K. Guennoun. Opérations d'intervention d'urgence : spécification des transformations verticales et horizontales au niveau architectural réseau. In *Rapport LAAS*, Novembre 2006.

[HIM99] D. Hirsch, P. Inverardi, and U. Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In 1^{st} *Working IFIP Conference on Software Architecture*, pages 127–142, San Antonio, TX, USA, February 1999. ISBN 0-7923-8453-9, Kluwer.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[JNI06] *Java Native Interface*. http ://java.sun.com/j2se/1.4.2/docs/guide/jni/, Last Consultation : October 2006.

[KTP00] C. Kotropoulos, A. Tefas, and I. Pitas. Frontal face authentication using morphological elastic graph matching. *IEEE Transactions on Image Processing*, 9(4) :555–560, 2000.

[LCG04] J. Liu, J. Cui, and N. Gu. Composing web services dynamically and semantically. In *IEEE International Conference on E-Commerce Technology for Dynamic E-Business, ISBN 0-7695-2206-8*, pages 234–241, Beijing , China, September 2004.

[LKA+95] D.C. Luckham, J.J. Kenney, L.M. Augustin, L.M. Vera, J. Bryan, and W. Mann. Specification and analysis of system architecture using rapide. *IEEE Transactions On Software Engineering*, 21(4) :336–354, April 1995.

[LRS91] S.W. Lu, Y. Ren, and C.Y. Suen. Hierarchical attributed graph representation and recognition of handwritten chinese characters. *Pattern Recognition*, 24 :617–632, 1991.

[LS03] J. Llados and G. Sanchez. Symbol recognition using graphs. In *IEEE International Conference on Image Processing*, volume 3, pages II– 49–52, Barcelona, Spain, September 2003.

[Luc06] D. C. Luckham. *The Standford Rapide Project, Stanford University, Stanford, CA, USA*. http ://pavg.stanford.edu/rapide/, Last Consultation : October 2006.

[LV95] D.C. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transactions On Software Engineering*, 21(9) :717–734, September 1995.

[MB00] B. Messmer and H. Bunke. Efficient subgraph isomorphism detection : a decompostion approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2) :307–323, 2000.

[McD01] J.H. McDuffie. Using the architecture description language MetaH for designing and prototyping an embedded spacecraft attitude control system. In *The 20th Conference on Digital Avionics Systems*, volume 2, pages 8E3/1–8E3/9, Daytona Beach, Florida,USA, October 2001.

[MDEK95] J. Magee, N. Dulay, S. Eisenbagh, and J. Kramer. Specifying distributed software architectures. In *Fifth European Software Engineering Conference, ESEC' 95, LNCS989, ISBN 3-540-60406-5*, Barcelona, Spain, September 1995. Springer-Verlag.

[MDK92] J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. In *International Workshop on Configurable Distributed Systems*, pages 102–117, London, England, March 1992.

[Met98] D. Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions On Software Engineering*, 24(7) :521–533, July 1998.

[MK06] J. Magee and J. Kramer. *Darwin Web Site, Imperial College London.* http ://www.dse.doc.ic.ac.uk/Research /architecture.html, Last Consultation : October 2006.

[MM04] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6) :51–59, November-December 2004.

[MR97] M. Moriconi and R. Riemenschneider. Introduction to SADL 1.0. Technical Report SRI-CSL-97-01, SRI Computer Science Laboratory, March 1997.

[MRT99] N. Medvidovic, D.S. Rosenblum, and R.N. Taylor. A language and environment for architecture-based software development and evolution. In 21^{st} *International Conference on Software Engineering*, pages 44–53, Los Angeles, California,USA, May 1999.

[MT00] N. Medvidivic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions On Software Engineering*, 26(1) :70–93, January 2000.

[NL91] N.M. Nasrabadi and W. Li. Object recognition by a hopfield neural network. *IEEE Transactions on systems, man, and cybernetics*, 21(6) :1523–1535, 1991.

[NSDC04] T. Nadour, N. Simoni, and G. Du-Chene. Towards dynamic vertical self-organization. In 12^{th} *IEEE International Conference on Networks (ICON'04), ISBN 0-7803-8783-X*, volume 1, pages 432–436, Singapore, November 2004.

[OT98] P. Oreizy and R. Taylor. On the role of software architectures in runtime system reconfiguration. In *International Conference on Configurable Distributed Systems*, pages 61–70, 1998.

[Pel03] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10) :46–52, october 2003.

[RBC+03] Y. Ren, D. Bakken, T. Courtney, M. Cukier, A. Karr, P. Rubel, C. Sabnis, W. Sanders, R. Schantz, and M. Seri. AQUA : An adaptative architecture that provides dependable distributed objects. *IEEE Transactions on Computers*, 52(1) :31–50, January 2003.

[RK57] L.C. Ray and R.A. Kirsch. Finding chemical records by digital computers. *Science*, 126(3278) :814–819, 1957.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation.* World Scientific Publishing, 1997.

[RS02] M. Riveill and A. Senart. Aspects dynamiques des langages de description d'architecture logicielle. *L'Objet : Coopération dans les systèmes à objet*, 8(3) :109–129, 2002.

[Rud98] M. Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In *International Workshop on Theory and Application of Graph Transformations (TAGT)*, Paderborn, Germany, November 1998.

[SDK+95] M. Shaw, R. Deline, D. Klein, T. Ross, D. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions On Software Engineering*, 21(4) :314–335, April 1995.

[SOA06] *Simple Object Access Protocol Version 1.2.* http ://www.w3.org/TR/soap/, Last Consultation : October 2006.

[SPT02] Sudhir Sangappa, K. Palaniappan, and Richard Tollerton. Benchmarking java against c/c++ for interactive scientific visualization. In *the 2002 joint ACM-ISCOPE conference on Java Grande*, Seattle, Washington, USA, November 03-05 2002.

[Std90] In *IEEE Std 610.12-1990, Glossary of Software Engineering Terminology,*, pages 1–83, 1990.

[Std00] In *IEEE Std 1471-2000, IEEE Recommended practice for architectural description of software-intensive systems*, pages i–23, 2000.

[Tae03] G. Taentzer. AGG : A graph transformation environment for modeling and validation of software. In *Applications of Graph Transformations with Industrial Relevance : Second International Workshop, AGTIVE 2003,LNCS 3062*, pages 446–453, Charlottesville, VA, USA, September-October 2003.

[TMA+96] R.N. Taylor, N. Medvidovic, K.M. Anderson, E.J. Whitehead, J.E Robbins, K.A. Nies, P. Oreizy, and D.L. Dubrow. A component and message-based architectural style for GUI software. *IEEE Transactions on Software Engineering*, 22(6) :390–406, 1996.

[TMO06] R.N. Taylor, N. Medvidovic, and P. Oreizy. *C2SADL Web Site, ISR-University of California, Irvine.* http ://www.isr.uci.edu/architecture/c2.html, Last Consultation : October 2006.

[Tor02] R. Torkar. *Building Reliable Component-based Systems*, chapter Dynamic software Architectures. extended report for I. Crnkovic and M. Larsson, Artech House, July 2002.

[Ull76] J.R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the association for computing Machinery*, 23(1) :31–52, 1976.

[VB93] S. Vestal and P. Binns. Scheduling and communication in MetaH. In *Real-Time Systems Symposium*, pages 194–200, Raleigh Durham, North Carolina,USA, December 1993.

[VP02] R. Vivanco and N. Pizzi. Computational performance of Java and C++ in processing large biomedical datasets. *Canadian Conference on Electrical and Computer Engineering, IEEE CCECE 2002.*, 2 :691–696, 12-15 May 2002.

[VPL99] James Vera, Louis Perrochon, and David C. Luckham. Event-based execution architectures for dynamic software systems. In *The 1st Working IFIP Conference on Software Architecture*, pages 303–318, San Antonio, Texas, 1999. Kluwer.

[WFKM97] L. Wiskott, J. Fellous, N. Krüger, and C. Von Der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7) :775–779, 1997.

[WHK+00] A. Wolf, D. Heimbigner, J. Knight, P. Devanbu, M. Getz, and A. Carzaniga. Bend, Don't Break : Using reconfiguration to achieve survivabibility. In *The 3rd Information Survivability workshop*, Boston,Ma,USA, October 2000. Kluwer Academic Publishers, ISBN : 1-4020-7043-8 2002.

[WS06] *Web Services Activity*. http ://www.w3.org/2002/ws/, Last Consultation : October 2006.

Glossaire

AADL : Avionics Architecture Description Language
ADL : Architecture Description Language
AGG : the Attributed Graph Grammar system
API : Application programming interface
ATG : Algorithme de Transformation de Graphes
BPEL : Business Process Execution Language
CPU : Central Processing Unit
CSP(1) : Communicating Sequential Processes
CSP(2) : Constraint Satisfaction Problem
dNCE : directed Neighbourhood Controlled Embedding
dNLC : directed Node Label Controlled
edNCE : edge directed Neighbourhood Controlled Embedding
edNLE : edge directed Node Label Controlled
EJB : Enterprise Java Beans
eNCE : edge Neighbourhood Controlled Embedding
eNLC : edge Node Label Controlled
JNI : Java Native Interface
MTG : Moteur de Transformation de Graphes
NAC : Negative Application Condition
NCE : Neighbourhood Controlled Embedding
NLC : Node Label Controlled
NP : Non-deterministic Polynomial time
OIU : Opérations d'Intervention d'Urgence
OSI : Open Systems Interconnection
PRA : Protocole de Reconfiguration de l'Architecture
QdS : Qualité de Service
SESC : Software Engineering Standard Committee
SOAP : Simple Object Access Protocol
SPO : Single PushOut
UDDI : Universal Description, Discovery and Integration
UML : Unified Modeling Language
WS-DIAMOND : Web Services - DIAgnosability, MONitoring and Diagnosis