Modélisation formelle de systèmes dynamiques autonomes : graphe, réécriture et grammaire

Cédric Eichler

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *09/06/2015* **par :**

CÉDRIC EICHLER

Modélisation formelle de systèmes dynamiques autonomes : graphe, réécriture et grammaire

---

JURY

JEAN-MARC PIERSON	Professeur d'Université	Président du Jury
FLAVIO OQUENDO	Professeur d'Université	Rapporteur
LIONEL SEINTURIER	Professeur d'Université	Rapporteur
NOËL DE PALMA	Professeur d'Université	Examinateur
ÉRIC RUTTEN	Chargé de recherche HDR	Examinateur
KHALIL DRIRA	Directeur de recherche	Co-directeur de thèse
THIERRY MONTEIL	Maître de conférence HDR	Co-directeur de thèse
PATRICIA STOLF	Maître de conférence	Co-directeur de thèse

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :
LAAS-CNRS - (UPR 8001)
IRIT - (UMR 5505)

Directeur(s) de Thèse :
Patricia STOLF, *Thiery MONTEIL* et *Khalil DRIRA*

Rapporteurs :
Flavio OQUENDO et *Lionel SEINTURIER*

"I love deadlines. I love the whooshing noise they make as they go by."

Douglas Adams

Résumé

Les systèmes distribués modernes à large-échelle évoluent dans des contextes variables soumis à de nombreux aléas auxquels ils doivent s'adapter dynamiquement. Dans ce cadre, l'informatique autonome se propose de réduire (voire supprimer) les interventions humaines lentes et coûteuses, en leur préférant l'auto-gestion. L'adaptabilité autonome d'un système repose avant tout sur une description adéquate de ses composants, de leurs interactions et des différents aspects ou topologies qu'il peut adopter. Diverses approches de modélisation ont étés proposées dans la littérature, basées notamment sur des langages de descriptions spécifiques (e.g., les ADLs) ou des modèles génériques plus ou moins formels (e.g., profils UML, graphes). Ces représentations se concentrent en général sur certains aspects ou propriétés du système dynamique et ne permettent ainsi pas de répondre à chacune des problématiques inhérentes à l'auto-gestion.

Ce manuscrit traite de la modélisation basée graphes des systèmes dynamiques et de son adéquation pour la mise en œuvre des quatre propriétés fondamentales de l'informatique autonome : l'auto-optimisation, l'auto-protection, l'auto-guérison et l'auto-configuration. Cette thèse propose quatre principales contributions théoriques et appliquées.

La première est une méthodologie pour la construction et la caractérisation générative de transformations *correctes par construction* dont l'application préserve nécessairement la correction du système. Le maintien d'une application dans un état acceptable peut ainsi être efficacement garanti lors de son adaptation.

La seconde contribution consiste en une extension des systèmes de réécriture de graphe permettant de représenter, mettre à jour, évaluer et paramétrer les caractéristiques d'un système aisément et efficacement. Ces affirmations sont soutenues par des exemples illustratifs concrets reposant sur DIET, un répartiteur de charge distribué. Une étude expérimentale extensive révèle un net gain d'efficacité vis à vis de méthodes classiques, en particulier celles intégrées nativement aux outils AGG et GMTE.

La troisième contribution s'articule autour de l'élaboration d'un module de gestion de bout en bout pour des requêtes de traitement d'évènements complexes. Elle démontre l'intérêt des graphes en tant que représentation abstraite et haut niveau dans un contexte applicatif comprenant de multiples solutions fragmentées.

La quatrième et dernière contribution réside dans le design d'un gestionnaire autonome apte à piloter tout système Machine-à-Machine se conformant au standard ETSI M2M. Elle illustre la méthodologie relative à la correction par construction, mais également l'intégration de la représentation proposée à des approches multi-modèles incluant des problématiques de cohérence interne. La faisabilité de l'approche est démontrée expérimentalement en s'appuyant sur une application de compteur intelligent pour la domotique.

Table des matières

Table des figures

Liste des Algorithmes

Liste des tableaux

Abréviations

MAPE-K	**M**onitoring **A**nalyzing **P**lanning - **K**nowledge
RFC	**R**equest **F**or **C**hange
AGG	the **A**ttributed **G**raph **G**rammar System
GMTE	**G**raph **M**atching and **T**ransformation **E**ngine
DPO	**D**ouble **P**ush-**O**ut
SPO	**S**ingle **P**ush-**O**ut
NAC	**N**egative **A**pplication **C**ondition
AC	nested **A**pplication **C**ondition
CEP	**C**omplex **E**vent **P**rocessing
CEPaaS	**C**omplex **E**vent **P**rocessing **a**s **a** **S**ervice
DIET	**D**istributed **I**nteractive **E**ngineering **T**oolbox
OMNI	service de nommage **omni**RB
MA	**M**aster **A**gent
LA	**L**ayer **A**gent
SED	**SE**rver **D**aemon
M2M	**M**achine à (to/**2**) **M**achine
OM2M	**O**pen **M**achine to **M**achine
ETSI	The **E**uropean **T**elecommunications **S**tandards **I**nstitute
REST	**RE**presentational **S**tate **T**ransfer
SCL	**S**ervice **C**apabilities **L**ayer
NSCL	**N**etwork **S**ervice **C**apabilities **L**ayer

GSCL	**G**ateway **S**ervice **C**apabilities **L**ayer
DSCL	**D**evice **S**ervice **C**apabilities **L**ayer
AC	**A**ttribué et **C**ontraint
VM	**V**irtual **M**achine

Pour Alfred,

Chapitre 1

Introduction

1.1 Contexte

L'informatique et ses usages se sont véritablement transfigurés au cours des dernières décades. Les terminaux isolés sont devenus connectés. Encouragées par le succès de l'informatique en nuage et du paradigme d'accès aux ressources en tant que services, les applications sont devenues connectées. Avec l'avènement de l'internet des objets, les objets du quotidien eux-mêmes sont devenus connectés. *Nous* sommes devenus hyper-connectés. Partout et tout le temps, nous accédons sans effort, parfois même inconsciemment, à une multitude d'informations, de ressources et de services. Sans effort ? Pour l'utilisateur peut être.

L'ouverture, la dispersion et la complexification des systèmes informatiques ont démultiplié leurs exigences en terme d'administration. En particulier, il est devenu crucial de s'adapter dynamiquement afin de faire face efficacement aux attaques, aux pannes logicielles et matérielles, à l'évolution du contexte et des besoins utilisateurs... Considérant la complexité des adaptations induites, par soucis de réactivité, de réduction de coûts et de baisse de la probabilité d'erreur, elles ne peuvent raisonnablement pas être conduites par des opérateurs humains. L'informatique autonome se propose de déléguer ce pilotage aux systèmes eux-mêmes en les rendant capables d'auto-gestion.

L'adaptabilité autonome d'un système repose avant tout sur une description adéquate de ses composants, de leurs interactions et des différents aspects ou topologies qu'il peut adopter. En cela, l'informatique autonome rejoint les problématiques de spécification et d'études des architectures logicielles dynamiques. Parmi les méthodes de modélisation existantes se trouvent un certain nombre d'approches basées sur les graphes. Ces derniers constituent un modèle universel pouvant représenter une grande variété de systèmes. Ils possèdent un caractère formel couplé à un aspect visuel, permettant ainsi de concevoir, étudier et gérer rigoureusement un système tout en facilitant une compréhension instinctive de situation et concepts complexes.

Nous nous intéressons ainsi à la représentation basée sur les graphes de systèmes dynamiques pour leur auto-gestion. Les contextes abordés sont multiples, reflétant les nombreux domaines applicatifs potentiels : traitement d'évènements complexes, gestion de tâches dans les nuages et systèmes machine-à-machine.

1.2 Problématiques

Les problématiques de cette thèse découlent de deux constats.

Premièrement, la représentation d'un système comporte plusieurs facettes souvent considérées séparément : correction et performance [1] .
À son tour, le caractère dynamique des applications considérées apporte ses propres contraintes en terme de représentation des évolutions et de leur comportement vis à vis des propriétés du système. Or, ce caractère n'est souvent qu'insuffisamment supporté [2, 3].
Enfin, l'application de ces représentations à la gestion autonome amène encore de nouvelles exigences. En effet, celle-ci possède plusieurs aspects ayant chacun ses propres besoins en terme de modélisation, ces derniers étant largement insatisfaits par le passé [4].

Les approches reposant sur les graphes permettent de répondre à certaines de ces exigences, mais sont elles adéquates pour l'implémentation de chacune des propriétés d'auto-gestion ? Si non, est-il possible de proposer des solutions pour qu'elles le soient ?

Deuxièmement, les approches à base de réécriture de graphe pour la transformation de modèles et de systèmes en général sont profondément clivées [5]. On retrouve d'un côté des solutions théoriques s'intéressant à des propriétés et des systèmes de réécriture de plus en plus abstraits et haut niveau. De l'autre se situent des considérations plus concrètes visant l'implémentation et l'automatisation de transformations dans le cadre de domaines applicatifs spécifiques.

Les réponses à la première problématique sont tenues de prendre en compte cette dichotomie.

Des approches et solutions génériques répondant à des besoin applicatifs concrets sont-elles envisageables ?

À cette fin et pour en faciliter la compréhension et l'usage, les solutions proposées dans ce manuscrit seront également illustrées à l'aide d'applications réelles.

1.3 Structure du document

Le présent document est constitué de sept chapitres. Les deux premiers, l'introduction et l'état de l'art, présentent le contexte de notre étude et ses concepts sous-jacents. Les défis liés à la représentation de systèmes dynamiques, les approches existantes et les objectifs de nos travaux sont détaillés dans le chapitre deux.

Les quatre chapitres suivants décrivent nos contributions, chacun visant un des quatre aspects majeurs de l'auto-gestion en particulier.

Le troisième chapitre expose dans un premier temps la représentation utilisée tout au long de ce manuscrit. Ce méta-modèle, basé sur les graphes et leur réécriture, est élaboré en formalisant et étendant une approche existante [6]. Nous illustrons par la suite son utilisation en détaillant la phase de modélisation et de **conception d'un module de gestion de bout-en-bout pour des requêtes de traitement d'évènements complexes**. Ce dernier démontre l'intérêt des graphes en tant que représentation abstraite et haut niveau dans un contexte applicatif comprenant de multiples solutions fragmentées. Nous détaillons finalement quelques transformations intervenant dans le cadre de politiques d'auto-optimisation.

Le quatrième chapitre présente une **méthodologie innovante pour la caractérisation générative et la construction de reconfigurations *correctes par construction*** pour un style architectural. L'utilisation de telles transformations répond avantageusement à une des deux principales problématiques de l'auto-protection, la protection du système contre lui-même. Ici, il s'agit d'efficacement garantir le maintien d'un système dans un état correct lors de ses reconfigurations.

Dans le cinquième chapitre, nous concevons un **gestionnaire autonome apte à piloter tout système M2M se conformant au standard ETSI M2M**. Ce faisant, nous illustrons l'intégration de notre modèle au sein d'une **vue architecturale multi-modèles intégrant des problématiques de cohérence interne**, de même que son adéquation vis à vis de l'auto-guérison du système.

Le sixième chapitre décrit une **extension des systèmes de réécriture de graphe permettant de représenter, mettre à jour, évaluer et paramétrer les caractéristiques d'un système aisément et efficacement**. Ceci facilite, comme détaillé par des exemples et scénarios concrets, l'évaluation de la qualité d'un système, sa gestion et son auto-configuration. Enfin, une étude expérimentale extensive révèle un net gain d'efficacité vis à vis de méthodes classiques.

Le dernier chapitre est dédié aux conclusions et perspectives de ces travaux. Finalement, les annexes décrivent les démonstrations et preuves les plus longues ainsi que le projet ANR SOP "penser Services globaux pour les Ordinateurs Personnels" [1], projet mené à bien dans le cadre de cette thèse.

1. ANR-11-INFR-001, homepage : sop-project.org/

Chapitre 2

État de l'art

Ce chapitre présente le contexte dans lequel s'inscrit les travaux décrits plus loin dans ce manuscrit.

La première partie est dédiée à l'informatique autonome. Nous présentons tout d'abord ses motivations et ses origines, puis introduisons les principales propriétés sous-tendant la gestion autonome, ainsi que la boucle de contrôle permettant de les implémenter. Enfin, nous nous positionnons vis à vis de ce paradigme en fixant le cadre de notre étude à la modélisation de systèmes pour leur gestion autonome.

Ceci va naturellement nous amener à nous intéresser dans un deuxième temps à la représentation de systèmes dynamiques. Nous en détaillons les considérations inhérentes, dont se dégagent notamment la nécessité de formalisation et de prise en compte des aspects dynamiques du système. Ces aspects soulèvent en particulier des problématiques liées à la correction, à sa vérification en général et sa conservation lors d'évolutions en particulier. Un rapide tour d'horizon des approches existantes et de leurs limitations est également proposé.

Par la suite, nous introduisons une famille d'approches particulières, basées sur les graphes, pour ce type de représentation. Nous détaillons en particulier les réponses qu'elles proposent aux diverses problématiques identifiées, ainsi que les différentes fondations théoriques de chacun des concepts sous-jacents. Ce faisant, nous nous positionnons et fixons nos objectifs vis à vis de ces derniers.

Finalement, nous présenterons les principaux outils et environnements de développement pour la manipulation et la réécriture de graphes, ainsi que leurs bases théoriques. Nous identifions également les outils qui serviront à l'implémentation et au test de certaines des contributions élaborées.

2.1 Contexte : l'informatique autonome

2.1.1 La gestion autonome

C'est en 2001 qu'IBM introduit le principe de l'informatique autonome au travers d'un manifesto [7] désormais célèbre. Horn y fait un constat aujourd'hui largement accepté : tout en devant s'adapter dynamiquement à leur environnement, les systèmes informatiques deviennent de plus en plus complexe et leurs échelles de plus en plus larges, ce qui rend leur administration plus ardue. Horn souhaite alors faciliter leur utilisation tout en soulageant les professionnels pour leur permettre de se concentrer sur des considérations théoriques et conceptuelles. Quoi qu'il en soit, considérant la

complexité de ces adaptations, leur omniprésence et leur fréquence, par soucis de réactivité, de réduction de coûts et de baisse de la probabilité d'erreur, elles n'auraient pu raisonnablement pas être conduites par des opérateurs humains.

Il s'inspire alors d'une métaphore reliant systèmes informatiques et corps humains : ces derniers sont régulés par le système nerveux autonome, permettant à leurs utilisateurs (communément appelés êtres humains) de se concentrer sur des activités noétiques ou physiques sans se soucier de la régulation de leur température, de leur rythme cardiaque... C'est ainsi que nait l'informatique autonome, dont l'objectif est de rendre les systèmes informatiques autonomes, c'est à dire auto-gérés.

Pour Horn, un système autonome doit adopter huit comportements :

1. *Se connaître soi-même* et être constitué de composant se connaissant. Un système autonome doit posséder une connaissance détaillée de son champ d'action et de son statut actuel, mais également des ressources disponibles et de son voisinage, c'est à dire des systèmes avec lesquels il peut interagir. La connaissance de son statut passe par une description précise des ressources utilisées et, par dessus tout, des ses propres composants. Ces derniers possèdent également une identité ; plus particulièrement, ils doivent donc maîtriser leurs interactions avec d'autres systèmes (et donc, d'autres composants). Finalement, un tel système doit être capable de fusionner avec d'autres systèmes et de se séparer en sous-systèmes au besoin, ne serait-ce que temporairement.
2. *Se configurer et se reconfigurer en réponse à des changements.* Il s'agit, pour un système, de gérer son propre déploiement et de l'adapter à des changements de contexte. Cela signifie arranger et réarranger l'utilisation de ressources, mais également effectuer des sauvegardes de logiciels et données afin de surmonter de potentielles erreurs et corruptions futures.
3. *Toujours chercher à s'optimiser*. Un système autonome doit constamment chercher à améliorer son fonctionnement. Pour cela, il doit observer attentivement chacun de ses composants, connaître ses objectifs et réagir à l'aide d'une boucle de contrôle avancée.
4. *Se réparer* ou se "soigner", c'est à dire surmonter les dysfonctionnements dus à l'exécution de routines ou l'occurrence d'évènements extraordinaires en se reposant sur des ressources redondantes ou sous-utilisées. Afin de corriger le problème, il doit en particulier être capable d'en identifier la ou les source(s).
5. Détecter, identifier et *se protéger* contre virus et intrusions afin d'assurer la sécurité ainsi que l'intégrité du système et des données.
6. *Connaître son environnement et agir en conséquence*, plus précisément adapter la transmission d'informations au contexte. Il s'agit de comprendre les intentions des utilisateurs afin de fournir des informations pertinentes et de les présenter de manière adéquate, en s'adaptant par exemple à l'appareil depuis lequel le contenu est consulté.
7. *Être ouvert.* Considérant la multiplicité des acteurs impliqués dans un système et l'hétérogénéité de son environnement ces systèmes doivent se reposer sur des standards ouverts. Ces derniers facilitent la collaboration : humaine lors de leur conception et réalisation, inter-systèmes lors de leur exécution. Une solution pour l'informatique autonome ne saurait donc être propriétaire.
8. *Anticiper les besoins.* Sans rien imposer, un système autonome doit anticiper les besoins d'un utilisateur et s'apprêter à y répondre en (ré-)organisant ses ressources. Cela peut aller du déplacement de contenu afin d'en faciliter l'accès jusqu'à la réservation de ressource en prévision d'un pic de charge.

2.1.2 Les propriétés auto-*

Deux ans plus tard, Kephart et Chess fournirons à l'informatique autonome une réelle maturité en publiant un article considéré aujourd'hui comme fondateur du domaine [8]. Ils réaffirment l'objectif de l'informatique autonome, l'auto-gestion [1] des systèmes suivant des politiques haut niveau. Utilisateurs et administrateurs peuvent ainsi se concentrer sur les objectifs à atteindre, sans se soucier des détails de leur mise en place. Cette auto-gestion est atteinte au travers de l'implémentation de quatre propriétés fondamentales [8, 9] raffinant les comportements précédemment décrits :

1. L'*auto-configuration* [2] permet à un système de s'installer et de se paramétrer, lui-même et chacun de ses composants, selon ses objectifs ou pour répondre à des besoins clients. Il ne s'agit pas simplement de gestion du déploiement initial, mais également de reconfiguration et mise à jour du paramétrage suite à l'évolution de l'environnement (e.g., re-allocation de bande passante) et à l'ajout ou la suppression de composants (effectués par le système ou par des humains). Des composants peuvent ainsi intégrer (ou disparaitre du) système fluidement. Ce dernier comprends les implications de ces ajouts/suppression et peut s'adapter en cas de besoin.
2. L'*auto-guérison* [3] décrit la capacité d'un système à détecter, diagnostiquer et surmonter les problèmes pouvant survenir dans son domaine. Elle s'applique en particulier à la gestion des bugs et pannes tant logicielles que matérielles. Cette activité peut être pro-active, anticipant et évitant les dysfonctionnements.
3. L'*auto-protection* [4] permet à un système d'assurer son intégrité et sa survie. Pour ce faire, il lutte, de manière réactive ou pro-active, contre des problèmes à large échelle provenant en particulier d'erreurs en cascade ou d'attaques externes. Ces problèmes sont plus précisément spécifiés comme ceux n'étant pas corrigés par des mesures d'auto-guérison.
4. L'*auto-optimisation* [5] réfère à l'utilisation efficace de ressources. Il ne s'agit pas nécessairement d'effectivement atteindre un optimal, mais au moins de continuellement chercher à s'en approcher. Cet optimal et cette notion d'adéquation de l'utilisation des ressources est liée à une évaluation du système à travers différentes métriques reflétant son efficacité en terme de performance et de coût.

Différentes réflexions portant sur les propriétés d'auto-gestion ont été depuis proposés. De nombreuses nouvelles propriétés ont été ainsi introduites (e.g., self-scaling, self-stabilizing, self- immunity...) [10].
Néanmoins, ces quatre propriétés fondamentales sont les seules à faire l'objet d'un large consensus et forment toujours le socle commun de l'auto-gestion.

2.1.3 La boucle de contrôle MAPE-K

IBM a suggéré un modèle de référence [8] pour la définition de gestionnaires autonomes, des modules fournissant aux composants du système un comportement autonome. Il se base, comme le suggérait Horn [7], sur une boucle de contrôle particulière nommée boucle de contrôle autonome ou boucle MAPE-K. La figure 2.1 présente la version re-actualisée [11] par IBM.

1. en anglais : *self-management*
2. en anglais : *self-configuration*
3. en anglais : *self-healing*
4. en anglais : *self-protection*
5. en anglais : *self-optimization*

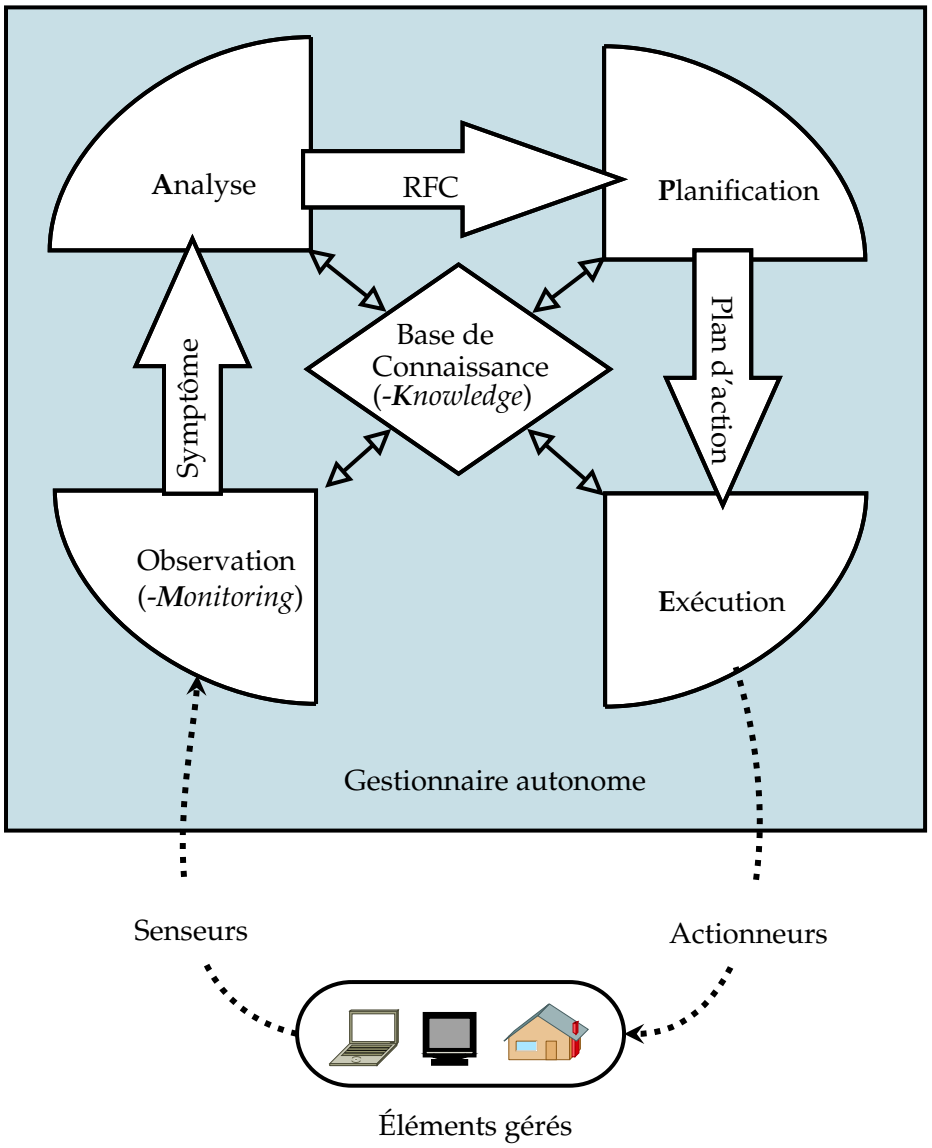


FIGURE 2.1 – La boucle de contrôle autonome MAPE-K

Un gestionnaire autonome est un module logiciel typiquement paramétré initialement par des opérateurs humains qui lui fournissent des objectifs et politiques de haut-niveau lui permettant d'assurer la gestion du système. Il est à l'écoute du système et de ses composants grâce à une collection de capteurs ou senseurs lui fournissant tout type d'informations pertinentes à la gestion du système. Grâce à sa connaissance du système, il pourra analyser ces informations et planifier des adaptations en concordance avec les politiques haut-niveau qu'il doit mettre en œuvre. Finalement, il va veiller à l'exécution de ces adaptations, agissant sur le système à l'aide d'un ensemble d'actionneurs.

Par la suite, nous introduisons chacune des étapes de la boucle MAPE-K. Afin de les illustrer, considérons un exemple et un scénario très simple : une pièce équipée, entre autre, d'une lampe et d'un store. Un gestionnaire doit fournir un éclairage adéquat, et se base pour cela sur les valeurs fournies par un capteur de luminosité et de deux actionneurs (interrupteurs) dirigeant l'état de la lampe et du store.

2.1.3.1 Observation - "Monitoring"

La tâche d'observation est dédiée à la surveillance du système, de ses propriétés et de son environnement à travers la collecte de toute sorte d'informations. Les informations pertinentes sont ensuite formatées et transmises à la tâche suivante sous la forme de symptômes.

Bien que l'observation se repose lourdement sur des sondes, elle ne se contente pas de simplement transmettre les données qu'elles envoient. Afin de ne propager que les informations pertinentes, cette tâche peut inclure des mécanismes d'agrégation et de filtrage, voir même d'analyse légère. Par exemple, l'absence de remontée d'information peut présager de la défaillance d'un senseur et doit donc être traitée comme une information en tant que telle.

Dans notre exemple, le module d'observation va récupérer la valeur transmise par le capteur de luminosité. Lorsque cette dernière est inférieure à un certain seuil, il va produire et transmettre un symptôme "faible luminosité".

2.1.3.2 Analyse

La tâche d'analyse traite les informations contenues dans les symptômes reçus en fonction de politiques et stratégies de gestion haut niveau. Elle repère des écarts entre fonctionnement actuel et fonctionnement souhaité, et génère des requêtes de changement (RFC [6]) transmises à au module suivant (Planification).

Reprenons l'illustration. Le module d'analyse compare le manque de luminosité à la politique du système relative à la luminosité. Celle-ci spécifie que, excepté week-end, la pièce doit être éclairée à partir d'une certaine heure déjà atteinte. L'analyse génère donc une RFC exigeant une augmentation de la luminosité ambiante.

2.1.3.3 Planification

La tâche de planification élabore des plans d'actions décrivant la manière dont les changements requis vont être mis en œuvre. Ces plans décrivent chacune des actions élémentaires à effectuer. Ils comportent également une composante temporelle et établissent ainsi un ordre ou des priorités au sein de ces actions.

Dans l'exemple, la planification doit établir une suite d'actions permettant d'augmenter la luminosité de la pièce. L'heure étant trop matinale pour espérer de la lumière naturelle et la lampe étant éteinte, la planification va demander son allumage via son interrupteur.

2.1.3.4 Exécution

Le module d'exécution est l'interface finale entre la boucle de contrôle et le système réel. Elle se charge de mettre en œuvre les plans prévus en exécutant les actions appropriées sur les éléments gérés par l'intermédiaire des actionneurs à sa disposition.

Finalement, la tâche d'exécution actionne l'interrupteur, allumant ainsi la lampe.

6. de l'anglais "*request for change*"

2.1.3.5 Base de connaissance - "Knowledge Base"

La base de connaissance est l'épine dorsale de la boucle MAPE-K. Elle contient tout la connaissance pertinente à la gestion du système : sa représentation, les politiques haut niveau à suivre, les métriques d'évaluation, les règles d'inférences propre à chaque tâche, l'historique de fonctionnement, les sondes et actionneurs disponibles...

Chacune des tâches peut y piocher des informations ou contribuer à cette base, dans le cadre de la constitution d'un historique, par exemple.

Dans l'exemple illustratif,
- outre la manière de traiter des mesures de luminosité, l'observation y consulte la valeur du seuil bas.
- l'analyse s'y informe de la politique à suivre concernant la luminosité, de la date et de l'horaire courants.
- la planification récupère dans la base de connaissance la liste des actionneurs, l'heure courante et l'état actuel de la lampe.
- l'exécution y trouve les informations nécessaires à la prise de contact avec l'actionneur, comme son adresse IP.

Cette liste souligne l'aspect central de la base de connaissance dans la boucle MAPE-K.

2.1.4 Positionnement vis à vis de l'informatique autonome

L'objectif de cette thèse n'est pas nécessairement la définition de politiques de gestion autonome, ni même l'implémentation de gestionnaire(s), mais plutôt d'améliorer les moyens permettant de le faire. Incidemment, la validation et l'illustration de ces améliorations va tout de même nous mener à concevoir des gestionnaires.

Rappelons que d'après Horn, la toute première propriété d'un système à vérifier pour permettre son adaptabilité autonome et dynamique est la connaissance de soi [7]. Cette connaissance passe bien entendu par l'observation du système, de ses paramètres et de son environnement, mais aussi par sa représentation. Nous nous situons donc au niveau de la base de connaissance et nous intéressons à ce second aspect : la représentation, description ou modélisation d'un système.

Cette représentation doit permettre l'implémentation de chacune des quatre propriétés d'autogestion et doit permettre de modéliser :
- le système
- ses composants
- leurs interactions
- les différents états ou configurations qu'ils peuvent adopter
- leurs adaptations ; c'est à dire des transformations permettant au système de changer d'état.

2.2 Représentation de systèmes dynamiques : contexte, méthodes et enjeux

La conception et l'étude d'architecture s'est fortement développée pour faire face à la complexité grandissante des systèmes [12]. D'après le dernier standard ISO/IEC/IEEE-42010 [12] en vigueur,

une architecture est définie comme les *"concepts et propriétés d'un système [...] relatifs à ses composants, leurs relations, et les principes guidant sa conception et son évolution"*.

Ces architectures peuvent être plus finement classifiées selon deux principaux critères [2] :

— D'après le point de vue qu'elles adoptent. Dans le domaine des architectures logicielles, un point de vue structurel ou comportemental est le plus souvent adopté [2, 13], certains travaux considérant les deux [2].

— D'après leur capacité à évoluer lors de l'exécution du système [14] ou non. Ce critère divise les architectures en deux catégories respectivement qualifiées de dynamiques et statiques.

Les architectures dynamiques étant les seules à permettre l'adaptabilité d'une application, elles seront naturellement l'objet de notre étude.

2.2.1 Architectures dynamiques : enjeux

De part leur nature profonde, la description d'architectures dynamiques ne peut se limiter à une unique topologie statique, mais doit inclure la spécification de l'ensemble des états (aussi appelés configurations) admissibles. Cet ensemble est généralement caractérisé par un style architectural, défini par Garlan et Shaw [15] comme *"une famille de systèmes en terme d'organisation structurelle"*. Plus spécifiquement un tel style *"détermine les composants et connecteurs pouvant être utilisés dans ses instances, ainsi qu'un ensemble de contraintes spécifiant leurs combinaisons acceptables"*.

Naturellement, un modèle pour la spécification d'architectures dynamiques doit également proposer une représentation pour les transformations elles-mêmes. Ces dernières incluent typiquement deux facettes : conditions d'applicabilités et effets.

La considération de transformations soulève assez immédiatement la question de la correction du système et, plus spécifiquement, de leur impact sur cette dernière. Il s'agit en particulier d'éviter l'introduction d'incorrections lors de transformations. En général, on distingue deux types de comportements acceptables : soit la préservation de la cohérence, soit son (ré-)établissement.

Outre sa correction assurant son bon fonctionnement, le système possèdent des exigences non-fonctionnelles, intimement liées à son adéquation ou efficacité. Par exemple, des configurations peuvent être évaluées vis à vis de leur qualité de service, leur consommation énergétique et/ou leur robustesse face à des pannes logicielles ou matérielles. Ces objectifs sont potentiellement concurrents. Dans les faits, utiliser des ressources redondantes peut améliorer le critère de robustesse tout en augmentant la consommation énergétique.

La satisfaction d'un objectif dépend des propriétés de chaque composant logiciel, tel que la machine sur laquelle il est déployé et les entités qu'il peut contacter, par exemple. Ces caractéristiques peuvent elles-mêmes être dynamiques et interdépendantes. Par exemple, l'ensemble des entités accessibles à partir d'une autre est récursivement dépendant des entités accessibles à partir de celles atteignables directement. Cet ensemble est sujet à évolution avec l'arrivée et le départ de composants ou connexions au sein du système.

Ainsi, modéliser un système afin d'en faciliter la gestion comporte au moins deux aspects (ou *concerns* [12]) généralement considérés séparément [1] : sa correction et sa performance (i.e., son adéquation vis à vis d'exigences non fonctionnelles). Ces considérations ont motivés la définition de langages de description appropriés et de formalismes évitant tout ambiguïté pour la conception, la gestion et l'analyse rigoureuse d'architectures.

2.2.2 Transformations et vérification de la cohérence

2.2.2.1 Types de transformations

En premier lieu, nous proposons un rapide tour d'horizon des différents types de transformations. Elles peuvent être classifiées selon :

- L'impact sur le modèle de représentation utilisé
  - Les transformations *exogènes* (e.g., [16–19]), qualifiées aussi de transformations de modèle, impliquent un changement de représentation. Elles peuvent par exemple être utilisées pour générer du code automatiquement ou pour transformer un modèle graphique en un modèle plus forme (e.g., [20]).
  - Au contraire, les transformations *endogènes* (e.g., [21–24]) ne modifient pas le modèle de représentation du système. Elles peuvent par exemple changer son état (sa structure, son comportement...) ou la précision de sa description.
- L'impact sur le niveau d'abstraction considéré
  - Les transformations *verticales* (e.g, [17–19, 24]) symbolisent des changements de granularité et de niveau de précision, qualifiées d'abstractions ou de raffinements. Abstraire un système permet par exemple de faciliter l'élaboration des preuves, tandis que le raffiner peut permettre de se rapprocher d'une implémentation.
  - Les transformations *horizontales* (e.g, [21–23]) ne modifient pas la précision de la vue architecturale. Elles sont usuellement liées à la transformation de modèle ou à des évolutions au sein de systèmes dynamiques.

Les transformations horizontales et endogènes, appelées reconfigurations, représentent typiquement les évolutions d'un système dynamique, sujets des présents travaux.

2.2.2.2 Les approches de vérification de la correction

De multiples approches de vérification ont été développées par le passé. Lukman a récemment proposé une étude comparative de ces approches et des différentes notions de correction considérées dans le cadre de la transformation de modèle [16]. Il s'intéresse donc à des transformations exogènes, qu'elles soient verticales ou horizontales. Les classifications proposées ainsi que les tendances identifiées sont transposables à la vérification de reconfigurations. Ainsi, dans le cadre de techniques formelles, les catégories suivantes se distinguent.

"*Model-checking on the fly*". L'approche probablement la plus immédiate consiste à vérifier la satisfaction d'un un ensemble de propriétés et contraintes lors de l'exécution du système. Si celui-ci est dynamique, ce processus est typiquement conduit après chaque transformation. Assimilable à du "*model checking on the fly*", elle peut entraîner des explosions combinatoires et nécessiter des procédures d'annulation ("*roll-back*") lors de la découverte de l'incorrection d'un état [25].

Ce type de techniques peut être affiné lorsque l'on s'intéresse en particulier à la conservation de la correction d'une application dynamique. Lorsqu'une transformation est appliquée à un état correct, il s'agit, plutôt que de confronter l'intégralité du système et de ses propriétés à vérification, de vérifier uniquement soit le sous-système impacté, soit les contraintes que cette transformation est susceptible d'invalider.

L'étude de Lukman [16] propose de multiples exemples dans le contexte de transformations exogènes. Dans le cadre de reconfigurations, on peut citer en particulier le travail d'Hirsch [22] basé sur le modèle tilde et la réécriture de graphe. Il propose une méthode et un modèle de haut niveau pour vérifier la préservation d'un style architectural quelconque lors d'une reconfiguration. Les transformations considérées sont spécifiées par un graphe, symbolisant une instance d'un style, et une règle de réécriture de graphe. Lorsque cette seconde est appliquée au premier, elle préserve le style si elle entraîne l'apparition (ou la non-apparition) d'un ensemble d'évènements.

Pour l'informatique autonome plus spécifiquement, Sharrock [26] adopte une telle approche lors du déploiement de serveurs supplémentaires afin d'absorber un pic de charge dans un répartiteur de tâches hiérarchique. Sachant qu'un composant ne peut gérer qu'un nombre limité de serveurs, les politiques proposées vérifient que ce dernier n'est pas dépassé. Si c'est le cas, la procédure est annulée et l'état initial du système est restauré.

Ce type d'approches réduit la complexité temporelle de la vérification, mais n'exempte pas de potentielles procédures d'annulation.

"*Model checking*" lors de la conception. Dans le cadre de systèmes dynamiques soumis uniquement à des reconfigurations, il est possible de générer et vérifier un graphe d'état/transition [27, 28] pour éviter les procédures d'annulation et mitiger la complexité de la vérification. Ceci n'est néanmoins possible que dans le cas de systèmes à états finis, ou exhibant une forme de périodicité ou de récurrence.

Pour l'informatique autonome en particulier, une approche similaire est proposée par la méthodologie FracToy [29] basée sur des descriptions formelles en Alloy. Dans un premier temps, il s'agit de définir une architecture ainsi que ses contraintes statiques et dynamiques. Un système autoconfigurable s'y conformant peut alors être décrit et vérifié. Cette vérification englobe en particulier la conformité de ses politiques de reconfigurations. Elle se base toutefois sur la recherche de contre-exemples et n'est donc envisageable que dans un domaine fini.

Preuves par théorèmes et correction par construction. La dernière méthode consiste à prouver formellement la satisfaction de certaines propriétés sous la forme de théorèmes. Ce type de preuves s'effectue typiquement en phase de conception et peuvent s'appuyer sur des assistants de preuves (e.g., COQ [30], Isabelle [23]). Les propriétés garanties sont diverses, reflétant la variabilité des domaines d'intérêts impliquant des transformations. Certains travaux prouvent ainsi la conservation : de la sémantique d'un système [31] vis à vis d'une transformation de modèle ; de l'information dans le cadre de transformations bidirectionnelles [32] ; de la conservation de la connexion ou de la séparation d'un nœud lors d'une reconfiguration [23]...

La correction par construction désigne l'application et la mise en œuvre de spécifications haut niveau [33], un système correct par construction ayant été construit de sorte à respecter sa spécification. Les méthodes de correction par construction s'appuient en général sur des preuves formelles pour générer de tels systèmes. Ce type d'approches est particulièrement utilisé dans la transformation de modèle et la génération automatique de code (e.g., [17–19, 34–36]).

Basées sur des preuves formelles et sur un raisonnement en phase de conception, ces méthodologies garantissent efficacement la consistance d'un système en ne requérant que peu ou pas de vérifications dans la phase d'exécution. Une stratégie permettant d'élaborer des méthodes correctes par construction consiste à investiguer les propriétés des transformations vis à vis de la conservation

de spécifications. Dans le cas de reconfigurations, il s'agit de prouver qu'elles préservent la correction du système. Lors de son exécution, l'application d'une telle transformation à un état correct produira alors nécessairement un (autre) état correct.

Positionnement et objectifs vis à vis de la vérification de correction. Les preuves formelles et les méthodes de correction par construction permettent d'efficacement assurer la validité d'un système en ne requérant que peu ou pas de vérifications en phase d'exécution. Toutefois, ces méthodes se positionnent principalement dans le cadre de la transformation de modèles. En particulier, elles visent souvent à générer automatiquement du code correct pour une spécification donnée.

La transposition de telles méthodes à des systèmes adaptables consiste à caractériser des reconfigurations préservant nécessairement la correction du système. Actuellement, la plupart de ces approches s'intéressent à la conservation de certaines propriétés (e.g., la connexion ou la séparation d'un composant [23]) plutôt que d'un style architectural. En particulier et à notre connaissance, aucune approche ne vise la préservation d'un style architectural quelconque (i.e., dont les propriétés et contraintes ne sont pas connues).

Ce problème, que nous tenterons de résoudre, n'est pas trivial : sans connaître les propriétés et contraintes à satisfaire, comment garantir leur conservation ?

2.2.3 Les approches de modélisation des systèmes dynamiques

2.2.3.1 Approches basées sur des langages

Un nombre considérable de langages de description d'architectures (ADLs[7]) ont été élaborés par le passé [37]. Ces langages sont encore largement utilisés (e.g., [38–40]), bien que leur conception remonte parfois aux années 1990. Ils proposent une syntaxe et une sémantique rigoureuse pour la description de composants et connecteurs constituants un système, ainsi que pour l'expression de ses propriétés structurelles et comportementales.

Les ADLs se concentrent toutefois en général sur des architectures statiques et ne fournissent pas un support satisfaisant pour des aspects dynamiques [2, 3, 6]. Parmi les ADLs généralistes classiques permettant la spécification d'évolutions, les plus notables sont Dynamic-ACME, Darwin, Dynamic-Wright, Rapide et Fractal ADL.

Dynamic-ACME [41] constitue un effort d'extension d'ACME [42] se restreignant cependant à la spécification de composants et connecteurs optionnels.

Darwin [43] implémente deux mécanismes dynamiques. Le premier, l'instanciation paresseuse, permet de retarder l'instanciation d'un composant jusqu'à sa première utilisation. Le second est la réplication de composants. La suppression de composants n'est par exemple pas possible.

Dynamic-Wright [44] enrichit Wright [45] pour lui permettre de considérer des architectures dynamiques. Il se limite toutefois à des évolutions prédéfinies, de sorte que le système doit avoir un nombre fini de configurations et de politiques de reconfigurations connues *a priori*.

Rapide [46] est un ADL adoptant une représentation basée évènements. Ses aspects dynamiques sont assez limités. Ils incluent principalement des modifications dans le routage des évènements et de la topologie de communication, ainsi que la création et la suppression d'interfaces.

7. de l'anglais *Architecture Description Langage*

Fractal ADL [47] et son extension, Fractal Aspect Component [48], intègrent des possibilités de reconfigurations assez complètes. Le modèle de composants Fractal [49] permet de plus l'exhibition d'interfaces génériques grâce à l'encapsulation de composants. Il a par conséquent connu un certain succès dans le cadre de l'informatique autonome en particulier. Les outils Jade [50, 51] et Tune [52, 53], par exemple, l'adoptent pour représenter les entités du système. Sa proximité avec l'implémentation mitige néanmoins ses capacités en terme de modélisation et de raisonnement sur les politiques de reconfiguration, ce qui a motivé son intégration au sein d'approches hybrides discutées dans la section dédiée.

En outre, la tendance générale est à l'utilisation d'approches semi-formelles pour la description d'architectures [54]. Sans formalisme, il ne peut néanmoins pas y avoir de vérification de cohérence [55].

Le π-ADL [56], un ADL de seconde génération, constitue une exception remarquable au sein de la nébuleuse des ADLs. Sa gestion du dynamisme et des transformations horizontales est très complète : il permet l'addition et la suppression de n'importe quel constituant d'un système. Ces modifications peuvent en particulier être spécifiées en fonction des entrées du système lors de son exécution. Les raffinements et transformations verticales sont spécifiés en π-ARL [24].

Le π-ADL est basé sur le π-calcul et couplé avec π-AAL [57], un langage expressif basé sur le μ-calcul permettant de spécifier les propriétés structurelles et comportementales d'une architecture.

L'approche π-ADL répond donc au besoin de formalisme permettant une vérification rigoureuse, et possède d'ailleurs plusieurs outils logiciels le supportant. Deux approches sont ainsi envisageables [57]. La première consiste à vérifier la satisfaction de propriétés et contraintes du modèle lors de l'exécution. Nous avons vu précédemment que cette solution peut être lourde et/ou ne détecter une incorrection que lors de son apparition.

La seconde réside dans l'élaboration de preuves par théorèmes en phase de conception. Ceci pourrait typiquement permettre de garantir une certaine forme de correction (i.e., la satisfaction d'un ensemble de propriétés et contraintes) et sa préservation, sans nécessiter plus de vérification lors de l'exécution du système. Néanmoins, ces preuves seraient difficilement transposables et généralisables à une architecture quelconque dont les contraintes sont *a priori* inconnues.

2.2.3.2 Approches basées sur des modèles génériques et approches combinées

Des techniques de représentation reposants sur des modèles génériques haut niveau fournissent des descriptions formelles ou semi-formelles des propriétés structurelles d'un système [4], leur côté visuel facilitant une compréhension intuitive de situations complexes. L'adoption de modèles largement répandus en facilite l'usage et la compréhension.

Concevoir et décrire des logiciels à l'aide d'UML, par exemple, est une pratique courante, tant dans les milieux académiques qu'industriels. UML fournit une définition et une terminologie standardisée pour la description d'un système, facilitant ainsi une compréhension consistante et profonde des architectures logicielles [58]. En dépit de son adoption massive, les descriptions basées sur UML en particulier manquent généralement d'outils formels pour garantir efficacement la correction d'un système, dû à l'inhérente semi-formalité d'UML.

Après UML, les graphes constituent probablement le modèle le plus largement adopté pour la spécification d'architectures logicielles. Ils bénéficient des mêmes avantages qu'UML couplés à un aspect formel leur permettant d'être au cœur de nombreuses techniques de vérification [16].

Néanmoins, la généricité de ces modèles peut les éloigner du système géré et les limiter parfois dans l'expression d'intérêts plus spécifiques liés par exemple à des propriétés comportementales [25].

2.2.3.3 Approches hybrides

Pour faire face à ces restrictions, de nombreux travaux favorisent l'adoption d'approches hybrides, en liant systématiquement des concepts architecturant décrits par une ADL (ou autre langage de description [59]) et représentation d'un modèle générique. Ce faisant, ils proposent des approches hybrides combinant des aspects tirés de ces deux types d'approches.

Medvidovic [60] et Kandé [61], par exemple, permettent d'enrichir UML avec des concepts de C2, Wright ou Rapide. D'autres propositions plus récentes visent l'intégration de concepts de l'ADL ACME au sein d'UML [40].

Dans le cadre de la gestion autonome, on retrouve par exemple des approches [21, 26, 53] basées sur la combinaison de Fractal ADL et d'UML. UML y est utilisé pour la spécification de styles architecturaux, de configurations et de règles de reconfigurations pour des systèmes dont le comportement sous-jacent est géré via une description en Fractal au sein de Tune.

2.2.3.4 Approches combinées par transformation de modèle

Plutôt que de modifier les représentations pour inclure des éléments exogènes, il est également possible de bénéficier des avantages liés à la fois aux modèles génériques et aux ADLs en passant d'une représentation à une autre grâce à des techniques de transformation de modèle.

Typiquement, ce type d'approches exploite les caractères visuel et familiers d'UML ou des graphes en permettant de définir des spécifications à l'aide de ces représentations. Ces spécifications sont ensuite traduites vers un ADL, un autre modèle, ou un quelconque formalisme. Elles bénéficient alors de la facilité d'utilisation et d'adoption de ces premiers et des avantages de ces seconds.

Il est par exemple possible de générer des descriptions en π-ADL à partir de profils UML [20].

2.2.3.5 Positionnement au sein des approches de modélisation pour les systèmes dynamiques

À une ou deux exceptions près, les ADLs souffrent de restrictions critiques quant au dynamisme et à la vérification.

Les graphes et les concepts leur étant liés offrent un certain nombre de solutions intéressantes vis à vis de ces problématiques. Celles-ci sont détaillées plus avant dans la section suivante.

Afin de se rapprocher du système géré, l'inclusion d'une modélisation basée sur les graphes au sein d'une approche hybride n'est pas exclue. Dans ce manuscrit, nous présentons une telle représentation couplant les graphes avec un modèle ad hoc visant un contexte applicatif spécifique (les systèmes Machine-à-Machine).

2.3 Approches de modélisation basées sur les graphes pour la représentation de systèmes dynamiques

Les graphes peuvent être, et ont été, utilisés conformément au standard ISO/IEC/IEEE-42010 [12] pour la spécification de vues architecturales. À un instant donné, l'état d'un système peut être modélisé par un graphe abstrait décrivant sa configuration actuelle. Leurs nœuds symbolisent alors n'importe quels constituants du système. La nature de ces constituants peut varier en fonction du système étudié et du paradigme adopté : services pour des architectures logicielles orientées service [62] ; composants logiciels pour des architectures logicielles orientées composants [63] ; machines pour la description d'architectures physiques [63] ; ou même êtres humains [64].

Les arcs du graphe représentent les interactions, connections, interdépendances et interrelations (utilisation, contrôle...) des éléments du système. Ces liens n'étant pas nécessairement symétriques et bidirectionnels, les graphes considérés se doivent d'être dirigés.

Un des points critiques pour la représentation de systèmes dynamiques est la spécification de leurs évolutions. Les graphes possèdent leurs propres formalismes pour la description de ces transformations. Ces derniers sont décrits dans la section 2.3.1.

Comme signalé précédemment, la spécification d'un système dynamique se doit de préciser l'ensemble des états qu'il peut adopter, généralement sous la forme d'un style architectural. Les méthodes de définition de styles architecturaux au sein des approches basées sur les graphes sont présentées dans la section 2.3.2.

Il est d'usage pour ces graphes d'être en outre attribués. En effet, les attributs (aussi appelés étiquettes ou labels) augmentent grandement l'expressivité du modèle en permettant de représenter toutes propriétés et informations pertinentes relatives à ses éléments. L'attribution des graphes est discutée plus avant dans la section 2.3.3.

2.3.1 Réécriture de graphe

La réécriture de graphe est une technique ayant fait l'objet de nombreux travaux [65] et ayant de multiples applications [5, 66–68]. Une règle de réécriture spécifie à la fois une transformation de graphe et les circonstances dans lesquelles elle peut être appliquée. À ce titre, ces règles constituent l'outil basique pour la représentation et la gestion du dynamisme dans les représentations basées graphe.

Il s'agit, en règle général, de remplacer un certain motif M, dit graphe mère, par un graphe fille D [8]. Pour cela, une occurrence de M (une image isomorphe) est recherchée au sein d'un graphe H, dit hôte, avant d'être supprimée, produisant le graphe H^-. Une image isomorphe de D est alors ajoutée au sein du graphe H^-. Plutôt qu'une simple juxtaposition, cet ajout est effectué selon une certaine notion d'intégration définissant les relations entre l'image de D et le reste du graphe.

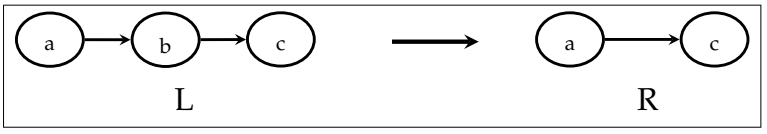
2.3.1.1 Les bases : approches SPO et DPO

Deux approches algébriques basées sur la théorie des catégories [69] se sont imposées de sorte à servir, dans la majorité des cas, de base à la définition formelle de ces règles de réécriture. Ces

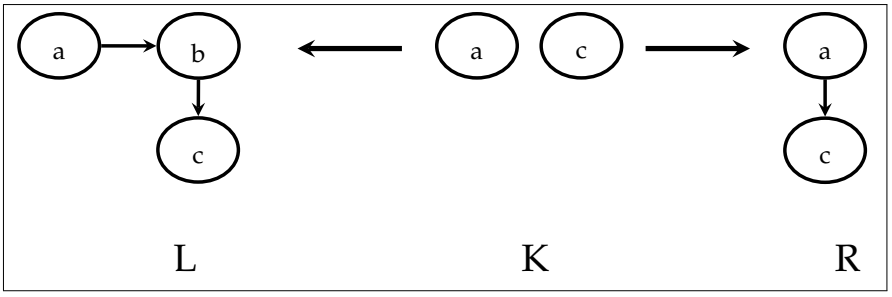
8. de l'anglais : *daughter*

approches, nommées Double PushOut (DPO)[70] et Single PushOut (SPO)[71, 72], présentent de nombreuses similitudes.

Dans les deux cas, M et D sont notés L et R et spécifient respectivement les parties gauches (Left) et droite (Right) d'une règle. L'approche d'intégration de R au sein de H^- est appelée *gluing*, ce qui peux se traduire par "collage" ou "encollage". Ce collage est rendu possible par le fait que les motifs initiaux ne sont pas complètement disjoints : il existe une relation entre les parties gauche et droite d'une règle. Lors de l'identification du motif décrit par L dans H dans la première étape de réécriture, seule l'image du sous-motif (de L) n'ayant pas de relation avec R est supprimée. Ainsi, certains éléments de H^- sont identifiables à des éléments de L, et donc à des éléments de R. De ce fait, H^- comprend une image d'un sous-graphe de R. Lors de la deuxième phase de réécriture, ce sous motif est étendu afin d'obtenir une l'image de la totalité de R. Sans simplification excessive, on peut considérer que ces approches diffèrent des deux manières suivantes [72].



(A) Approche SPO



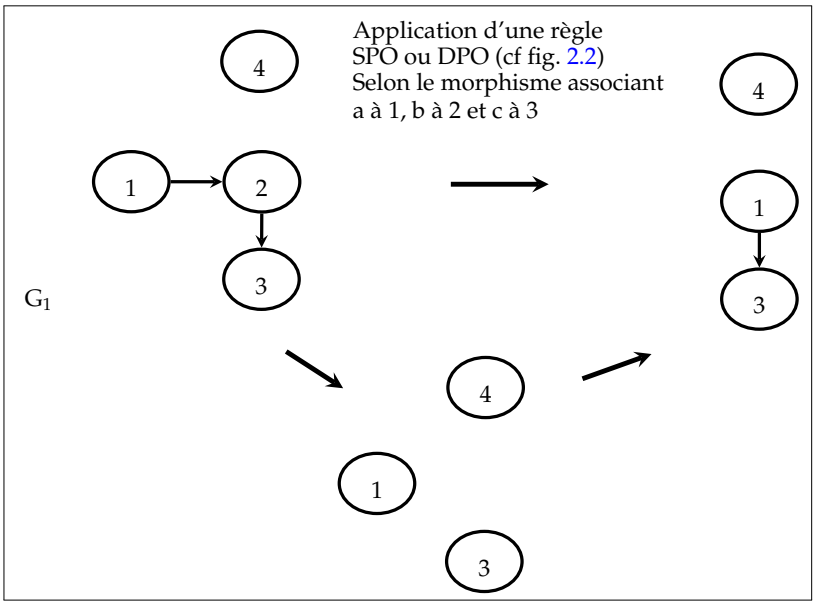
(B) Approche DPO

FIGURE 2.2 – Règle de réécriture : un exemple

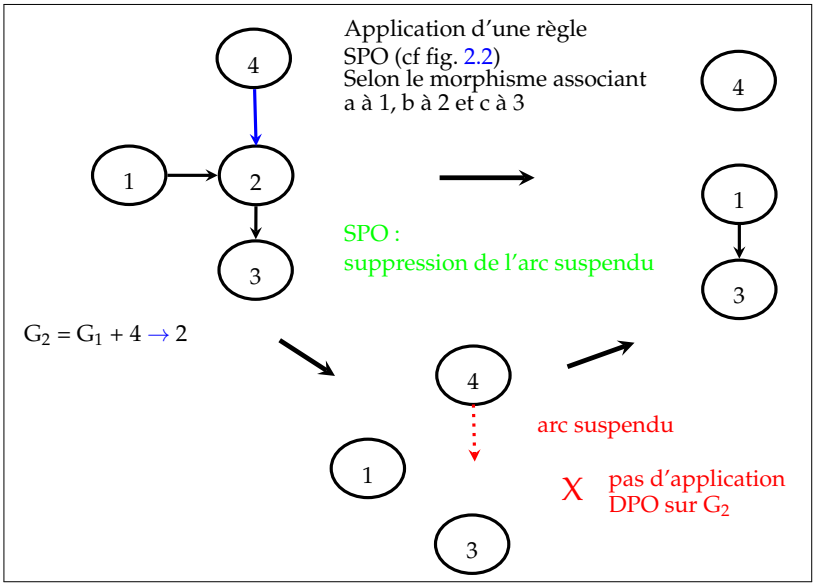
Spécification. Une règle SPO, notée $L \rightarrow R$ est caractérisée uniquement par L, R et un morphisme partiel entre ces derniers. Dans le cadre de l'approche DPO, un graphe interface K est explicitement spécifié. Une règle est alors notée $L \leftarrow K \rightarrow R$. Le graphe K (Ker ou Kernel) spécifie la partie invariante vis à vis de l'application de la règle.

La figure 2.2 présente un exemple d'une règle de réécriture exprimée selon l'approche SPO 2.2a et DPO 2.2b, les morphismes étant implicites et sont associés à la fonction identité (le nœud noté a de la partie gauche est associé au nœud noté a de la partie droite, etc). Les similitudes y sont flagrantes, la règle exprimée selon les deux approches ayant notamment les mêmes parties L et R. En l'occurrence, cette règle remplace une chaîne de trois nœuds en une chaîne de deux nœuds, et ce en supprimant le maillon central puis en introduisant un arc entre le premier et le troisième maillon.

Gestion des arcs suspendus. Supprimer un ou des nœuds d'un graphe peut conduire à l'apparition d'arcs dont une ou deux des extrémités a/ont disparu. On parle alors d'arcs suspendus. Dans l'approche DPO, une règle ne peut pas être appliquée si son exécution entraîne l'apparition de tels arcs. Cette condition supplémentaire est appelée **condition de suspension**. Dans le cadre SPO, ces arcs sont simplement effacés.



(A) Application d'une règle de réécriture SPO et DPO



(B) Gestion des arcs suspendus

FIGURE 2.3 – Application de règles de réécriture SPO et DPO : similitudes et différences

L'application des deux règles introduites précédemment est présentée dans la figure 2.3a. Initialement, deux graphes sont considérés : G_1, composé des nœuds notés 1, 2, 3 et 4, ainsi que deux arcs de 1 vers 2 et de 2 vers 3. Dans les deux approches, la règle précédente est applicable à G_1 selon le morphisme associant a à 1, b à 2 et c à 3. Son application, identique dans les deux approches, consiste à supprimer le nœud 2 ainsi que les arcs $1 \rightarrow 2$ et $2 \rightarrow 3$, et à ajouter un arc entre 1 et 3.

La figure 2.3b illustre le comportement des mêmes règles en présence d'un arc suspendu. Le graphe considéré est à présent G_2, construit à partir de G_1 en lui ajoutant l'arc de 4 vers 2 (représenté en bleu). La règle exprimée selon l'approche DPO n'est pas applicable à G_2 : son application entraînerait la suspension de l'arc de 4 vers 2. C'est ici que les approches diffèrent : la règle SPO est applicable à G_2, son application étant équivalente à celle intervenant sur G_1, si ce n'est que l'arc $4 \rightarrow 2$ est, de plus, supprimé.

Ainsi, les règles de réécritures de graphes peuvent être, dans le cadre des approches SPO et DPO, graphiquement spécifiées à l'aide de deux ou trois graphes. Cet aspect visuel et intuitif cachant la complexité intrinsèque du formalisme sur lesquelles elles reposent constitue un des facteurs clés du succès de ces approches.

2.3.1.2 Les extensions

Un certain nombre d'extensions ont été proposées à ces fondations. Celles-ci visent soit à restreindre l'applicabilité d'une règle soit à étendre les effets de son application. Les plus significatives sont détaillées ci-après.

Restriction d'applicabilité. Concernant la restriction de l'applicabilité d'une règle, on peut en particulier citer les conditions négatives d'application (NACs) [9] [73]. Les NACs définissent des motifs étendant L et dont la présence dans un graphe interdit l'application d'une règle.

Une règle considérant des NACs est donc applicable à un graphe G selon un morphisme m si pour tout $nac \in NAC$, il n'existe pas de morphisme m_{nac} de nac vers G étendant m (c'est à dire associant à tout élément de L l'image de cet élément par m).

Les NACs ont plus tard été étendues aux conditions d'application imbriquées (ACs) [10] [74]. Plus expressives que les NACs, les ACs permettent d'exprimer n'importe quelle formule logique du premier ordre sur un graphe. Toutefois, les NACs, contrairement aux ACs, peuvent être spécifiés visuellement sous la forme de graphes.

Extension des effets. En terme d'extension des effets d'une règle, les instructions de connexion forment une des adjonctions les plus notables, et certainement des plus étudiées et étendues [75].

Il existe deux principales approches basiques relatifs à ces instructions : les mécanismes NLC [11] et NCE [12]. Ceux-ci s'appliquent tous deux à des graphes non dirigés aux nœuds labellisés. Ils permettent de décrire l'établissement de connexions (arêtes ou arcs) entre l'image de D et H^-. Les instructions NLC et NCE sont de la forme (δ, μ) et (δ,ν) respectivement, où δ et μ désignent des labels de nœuds et ν désigne un nœud du graphe fille. Elles consistent à ajouter une arête entre tout arc libellé δ de H^- et soit tout arc libellé μ de D, soit l'arc désigné par ν.

Ces approches ont été par la suite étendues afin de prendre en compte des graphes dirigés (dNCE ou dNLC, d signifiant *directed*) ou des labels sur les arcs (eNCE ou eNLC, e signifiant edge), les deux pouvant être combinées en edNCE et edNLC.

Dans le cadre des approches SPO et DPO, des arcs ne peuvent être ajoutés qu'entre des nœuds apparaissant dans R. Pour ajouter des arcs entre des nœuds pré-existant dans H et de nouveaux nœuds, il faut donc que ces premiers apparaissent dans la partie invariante de la règle. Les instructions de connexion apportent ainsi une plus grande flexibilité. Ceci est particulièrement pertinent lorsqu'inclure tous les nœuds au sein de la partie invariante est fastidieux : quand le nombre d'arcs à ajouter est inconnu ou dépend du contexte, par exemple.

9. de l'anglais : Negative Application Conditions
10. de l'anglais : nested Aplication Conditions
11. *Node Label Controlled*
12. *Neighbourhood Controlled Embedding*

Il est intéressant de noter que ces instructions de connexion sont initialement des relations d'intégrations au sein d'approches algorithmiques pour la réécriture de graphe. Ces approches peuvent être considérées à la base comme concurrentes des approches algébriques (et au *gluing*) présentées précédemment. Elles ne sont néanmoins pas mutuellement exclusives et peuvent être associées afin de spécifier des règles plus expressives [6, 76].

2.3.1.3 Positionnement vis à vis des approches théoriques de réécriture de graphe

L'approche SPO est non seulement une des deux approches les plus utilisées, mais également plus expressive que la seconde [72], l'approche DPO. Il est d'ailleurs évident que l'adjonction de NACs permet de rétablir la condition de suspension et d'exprimer toute règle DPO sous la forme SPO.

Le principal avantage lié à l'adoption d'un point de vue se conformant à la théorie des catégories est son caractère abstrait et haut niveau facilitant l'élaboration de preuves [65, 69, 71]. Cette vision s'éloigne néanmoins de l'implémentation, ce qui a justifié des tentatives de traduction de l'approche SPO d'un point de vue ensembliste et opérationnel plus classique [6].

Nous nous inscrivons dans la continuité de cette tentative de traduction en améliorant les bases du modèle proposé dans [6].

2.3.2 Caractérisation de styles architecturaux

Pour la modélisation basée sur des graphe, deux méthodes de caractérisation de styles architecturaux dominent : les graphes types [62, 77] et les grammaires de graphes [22, 78, 79].

2.3.2.1 Graphe type

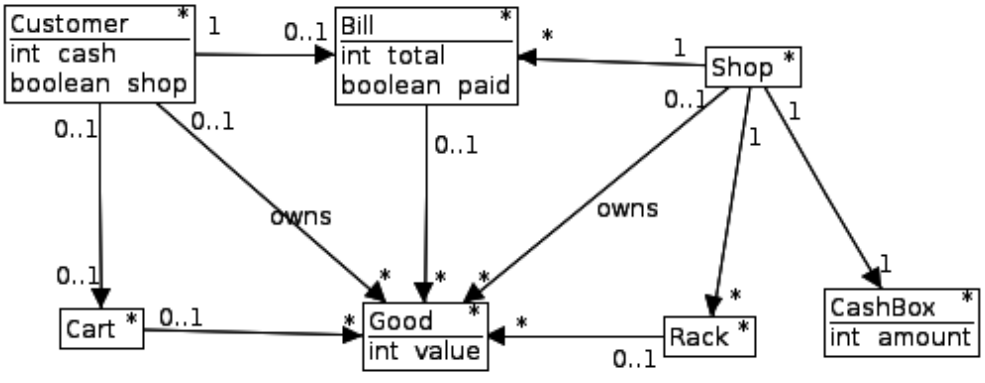


FIGURE 2.4 – Un graphe type.

Les graphes types [80] sont similaires aux diagrammes de classe UML [62]. La figure 2.4 illustre un graphe type décrivant une activité de shopping tiré d'un exemple de base d'AGG [27]. La ressemblance est effectivement flagrante.

On y retrouve des objets, ici des clients, caddies, biens, factures, magasins, rayons et caisses. Ceux-ci possèdent des attributs typés, chaque facture possédant par exemple un attribut entier total et un

booléen indiquant si elle est payée ou non. Des arcs attribués symbolisent les relations entre objets, un magasin pouvant par exemple posséder un bien.

Chaque objet et chaque relation est associé à des cardinalités minimum et maximum. Ainsi, l'activité peut être composée de n'importe quel nombre (*) de chaque objet. Sur une relation, source et destination possèdent des cardinalités. Par exemple, chaque client peut être lié à 0 ou 1 (0..1) facture, chaque facture étant à son tour liée à exactement 1 client (1).

2.3.2.2 Grammaire de graphes

Les grammaires génératives [81] constituent une des fondations majeures de la théorie des langages. Elles sont très largement utilisées pour spécifier formellement divers langages, qu'ils soient naturels ou informatiques. Cette caractérisation est à la base de la majorité des compilateurs actuels, ainsi que de nombreuses méthodes d'analyse de langages.

De telles grammaires sont en général spécifiées par un axiome, des termes terminaux et non-terminaux et un ensemble de règles de réécriture (de chaînes de caractères) appelées productions de la grammaire. L'application successive de règle de productions à partir de l'axiome permet de générer une chaîne de caractères. Si la dite chaîne est exempte de terme non terminal, c'est alors une phrase du langage.

Grâce à la réécriture de graphes, c'est tout naturellement que ces grammaires peuvent être, et ont été [65, 82], étendues des chaînes de caractères aux graphes. Le résultat, les grammaires de graphes, constitue un formalisme expressif ayant une longue tradition dans le domaine de la spécification de logiciels en général [66, 83, 84] et de styles architecturaux en particulier [22, 78, 79, 85].

Ces grammaires spécifient non plus un ensemble de phrases acceptables, c'est à dire un langage, mais un ensemble de graphes. Dans le cadre de la description de styles architecturaux, l'ensemble spécifié correspond typiquement aux configurations acceptables du système. Les règles de productions formalisent typiquement la procédure d'ajout d'un composant au système.

Remarques. Notons que les grammaires de graphe ont rapidement été abstraites pour donner les systèmes de remplacements de haut niveau [13] [86]. Leur étude s'est ainsi détournée de la théorie des langages dans laquelle elles prenaient leurs sources.

Il convient également de faire la distinction entre systèmes de réécritures et grammaires. Les deux termes désignent le même concept, mais reflètent des axes d'études divergents. Une grammaire est en général considérée simplement comme le moyen de spécifier un ensemble (de phrase, de graphe), l'attention portant alors sur ce dernier. L'utilisation du terme "système de réécriture" indique qu'il est l'objet principal d'intérêt, dans le cadre de l'étude de ses propriétés telles sa finition, sa confluence...

2.3.2.3 Positionnement et objectifs vis à vis de la caractérisation de styles architecturaux

Les graphes types sont faciles à utiliser et à interpréter, en particulier grâce à leur similitude avec UML, un modèle familier et répandu. Ils sont toutefois moins expressifs que les grammaires de graphes, qui permettent par exemple de spécifier simplement des relations de précédence. Nous

13. en anglais : *high level replacement systems*

adoptons, pour la représentation de style, ce second formalisme. Cette décision est motivée en particulier par l'une des particularités les plus notables des grammaires de graphes : leur aspect génératif.

En effet, un ensemble de règles de transformation (les productions) rentre dans la caractérisation même d'un style architectural. Ces règles peuvent être considérées comme *correctes par définition*, l'application d'une production p à un état acceptable (i.e., un état pouvant être atteint à partir de l'axiome en lui appliquant une suite (p_i) de règles de productions) résultant nécessairement en un état acceptable (car pouvant être atteint à partir de l'axiome en lui appliquant $(p_i) + p$, une suite de règles de production).

Par conséquent, la spécification d'un style à l'aide d'une grammaire fournit un ensemble de règles correctes par définition. Adopter cette représentation fournit donc une base de travail des plus intéressantes pour l'étude de la conservation de la consistance dans des systèmes adaptables.

2.3.3 Attribution

Une des éléments clés des représentations basées sur les graphes est la définition de leurs attributs, ces derniers représentant les propriétés basiques d'un élément du système.

2.3.3.1 Représentation, intégration et modification

La solution la plus complexe consiste à adopter une vue "éclatée" dans laquelle les attributs sont intégrés comme des nœuds particuliers du graphe [28, 87]. En particulier, leur domaine de définition et leurs opérations peuvent être définis sous la forme de signatures algébriques à parités multiples [88] SIG, de sorte que les attributs eux-même soient des éléments d'une SIG-algèbre [87].

Les attributs peuvent ainsi être naturellement manipulés à l'aide d'opérateurs prédéfinis, et ajoutés ou supprimés comme n'importe quel nœud du graphe [28]. Cette modularité n'est pas sans conséquence. Augmenter de la sorte la taille du graphe augmente la taille de l'entrée du problème de recherche de morphisme, un problème qui est, rappelons le, NP.

Dans une solution plus simple et plus immédiate, une liste de couples représentant des attributs et leurs domaines de définition est assignée à chaque élément du graphe (i.e., chaque nœud et chaque arc) [6, 27, 64]. Les modifications d'attributs peuvent alors soit être directement spécifiés au sein d'une règle [27], soit au travers d'instructions de modification [6].

Dans les deux approches, une règle ne peut modifier la valeur d'un attribut que si celui-ci est dans son champ d'application, c'est à dire s'il apparaît dans la règle. Ceci peut entraîner de multiples applications de règles. En particulier, cela peut créer un effet domino lorsque changer la valeur d'un attribut impacte récursivement une succession d'attributs interdépendants.

2.3.3.2 Variabilité

Considérer des attributs variables améliore grandement la flexibilité des outils de manipulation de graphes. Dès 1979 [89], plusieurs travaux [6, 90–92] ont traité de leur intégration dans le contexte de la recherche de motifs et de la transformation de graphes. En particulier, [92] introduit de la variabilité dans les attributs des motifs, mais également dans ses nœuds et sous-graphes. Ces travaux

sont parfois intuitifs mais informels [6, 92], et se restreignent quasi-exclusivement à la variabilité au sein des motifs [6, 89–92]. Or, dans des systèmes réels, la valeur d'un attribut peut être inconnue, du fait par exemple d'une décision de placement repoussée, de l'absence ou de la défaillance d'une sonde...

2.3.3.3 Positionnement et objectifs vis à vis de l'attribution

Nous nous fixons en particulier deux objectifs vis à vis de l'attribution.

Premièrement, il s'agit d'enrichir le modèle proposé dans [6] qui ne considère des attributs variables qu'au sein des motifs afin d'étendre la variabilité au graphe modèle. Ceci permet de refléter des valeurs inconnues ou encore de proposer des approches d'instanciation paresseuse similaires à ce que propose Darwin.

Deuxièmement, nous avons vu que les approches classiques peuvent présenter certaines restrictions d'applicabilité ou d'efficacité vis à vis de la modification et de l'évolution des attributs. Nous les expliciterons plus avant et les lèveront dans ce manuscrit.

2.4 Outils et environnements de développement

De nombreux outils visent ou se reposent sur la manipulation de graphes, implémentant en particulier des règles de réécritures [66].

Certains de ces outils ont été développés en visant spécifiquement un ou des cas d'utilisations particuliers. Ces derniers incluent majoritairement des tâches d'ingénierie informatique, avec par exemple Fujaba [14] [93, 94] et AToM^3 [15] [95, 96], deux outils supportant la transformation de modèle pour la conception multi-paradigme d'applications informatiques.

Nous nous intéressons ici à la seconde catégorie d'outils et environnements de développement, ceux génériques et ne se restreignant pas à un domaine applicatif spécifique. Cette catégorie comporte de nombreux outils (GrGen [16] [97], GP [17] [98],...), trois des plus éminents, AGG, Groove et GMTE, sont détaillés ci-après.

2.4.1 AGG

AGG [18] [27] est un environnement de développement de systèmes de transformation de graphes attribués, permettant de spécifier des grammaires de graphe. Développé en Java au sein de l'Université Technique de Berlin [19] depuis 1997, c'est l'un des outils (si ce n'est l'outil) de transformation de graphe les plus aboutis, les plus cités et les plus utilisés actuellement [99].

Deux facteurs ont largement contribué au succès d'AGG : son efficacité temporelle et son interface utilisateur graphique masquant la complexité du formalisme sous-jacent.

14. *From UML to Java And Back Again*, homepage : fujaba.de
15. *A Tool for Multi-formalism and Meta-Modelling*, homepage : atom3.cs.mcgill.ca
16. Graph Rewrite Generator, homepage : www.info.uni-karlsruhe.de/software/grgen/
17. Graph Programs
18. *The Attributed Graph Grammar System*, homepage : user.cs.tu-berlin.de/ gragra/agg/index.html
19. *Technische Universität Berlin*

2.4.1.1 Base théorique

AGG permet de manipuler des multi-graphes attribués par des classes Java. Les règles de réécriture y sont définies suivant l'approche SPO et peuvent inclure des NACs. Elles peuvent également modifier la valeur d'attributs apparaissant dans sa partie droite, les attributs pouvant être exprimés sous la forme d'une expression analytique (ré-)évaluée à la fin de chaque transformation. AGG accepte des relations non-injectives dans la recherche de motif, mais non au sein des règles de réécritures elles-mêmes (i.e., entre L et R).

En plus de règles de réécriture, AGG permet également de spécifier des graphes types et des contraintes de cohérence, auxquels sont confrontés automatiquement les graphes initiaux et les règles définies par l'utilisateur. De plus, il est capable de générer, à partir d'une grammaire, un graphe d'état correspondant aux différents état accessibles, chaque transition correspondant à l'application d'une règle de production.

On peut donc considérer AGG, entre autre choses, comme un outil de spécification et de vérification de grammaires de graphes acceptant comme base axiomatique, pour sa notion de correction, un graphe type et des contraintes.

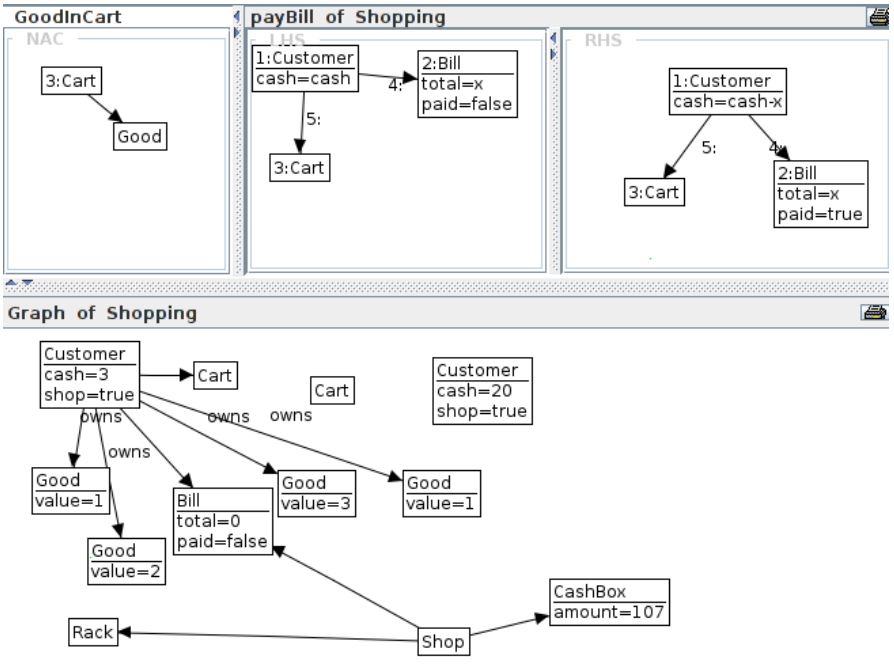


FIGURE 2.5 – Réécriture de graphe et modification d'attributs avec AGG

La figure 2.5 est une capture d'une partie de l'interface utilisateur d'AGG. Les nœuds des graphes et leur attributs y sont représentés d'une manière analogue à UML. La partie inférieure représente le graphe de départ, tandis que la partie supérieure détaille les trois composants d'une règle de réécriture, de gauche à droite son NAC, sa partie gauche et sa partie droite. La règle illustrée ici représente le paiement d'une facture par un client. Elle met à jour la valeur de l'attribut "cash" du nœud "customer" en lui soustrayant "x", l'attribut "total" de sa facture, et passe l'attribut "paid" de la dite facture de "false" à "true". Ces changements sont simplement stipulés dans l'expression de la valeur de l'attribut dans la partie droite de la règle.

2.4.1.2 Efficacité

AGG intègre deux principaux mécanismes d'optimisation liés à l'étape la plus lourde de la transformation de graphe : l'identification d'un motif au sein d'un graphe cible (ou hôte). Cette identification est formalisée sous la forme d'un problème de recherche d'homomorphisme entre deux graphes, un problème NP-complet pour des graphes quelconques (il peut être simplifié dans certains cas, comme par exemple en présence de graphes planaires).

AGG représente et résout cette recherche sous la forme d'un problème de satisfaction de contrainte, réduisant significativement la complexité algorithmique du processus [100].

Il comporte également des mécanismes de retours en arrière intelligents [20] : la recherche de motif est effectuée de manière incrémentale, permettant de reprendre la recherche à un sous motif valide après avoir éliminé une extension possible.

2.4.2 GROOVE

GROOVE [21] [28] est un ensemble d'outils pour la transformation de graphe et la génération de graphes d'états. Il est développé depuis 2004 à l'Université de Twente [22] et implémenté en Java.

GROOVE est constitué de trois composants. Le simulateur, un outil comportant une interface graphique permettant la définition, l'édition et l'utilisation de graphes et de règles de réécriture. Il intègre en particulier les fonctionnalités des deux autres composants.
Le générateur, un outil permettant de construire le graphe d'état d'une grammaire. Des stratégies et des mécanismes de priorité permettent d'y modifier l'ordre d'application des productions.
Finalement, le vérificateur propose de confronter des propriétés de logique temporelle à un graphe d'état.

2.4.2.1 Base théorique

GROOVE permet de manipuler des multi-graphes à l'aide de règles de réécriture décrites suivant le formalisme SPO. Graphes et règles peuvent découler d'un graphe type. Tout comme AGG, des relations non-injective dans la recherche de motif sont acceptables, mais pas au sein des règles de réécritures elles-mêmes.

Ce environnement possède de nombreuses particularités intéressantes, visant en particulier la manipulation des attributs, pour lesquels il adopte une vue explosée (représentant donc les attributs comme des nœuds particuliers). Il permet l'utilisation de nœud non attribué (et intègre donc des mécanismes d'attribution partielle), jouant entre autre le rôle de *wildcard* ou "joker" dans la recherche de motif. Des formules logiques relativement avancées sur les attributs peuvent également être spécifiées, formules devant alors être satisfaites lors de l'identification de motif. En plus des attributs, des informations sur les nœuds peuvent être représentés sous la forme de `flags`.

D'autres fonctionnalités pourraient encore être citées, comme la spécification de règles imbriquées, la paramétrisation de règle de réécriture, les mécanismes de contrôle pour la génération d'un

20. *smart-backtracking*
21. *GRaphs for Object-Oriented VErification*, hompeage : groove.cs.utwente.nl/
22. *Universiteit Twente*

graphe d'état, l'intégration de concepts de logique munie de quantificateurs... Une des plus remarquables est la possibilité de définir, dans la partie gauche d'une règle de réécriture, un arc sous la forme d'une expression régulière. Cet arc spécifie alors des chemins, ce qui permet non seulement de ne pas définir explicitement l'intégralité d'un motif, mais également de gérer et d'anticiper, avec une seule règle, plusieurs cas réels.

Cette grande versatilité et ce confort dans la manipulation des attributs est probablement l'un des arguments majeurs en faveur de GROOVE [23], mitigeant ses performances inférieures à celles d'AGG.

2.4.2.2 Attribution éclatée et manipulation d'attributs

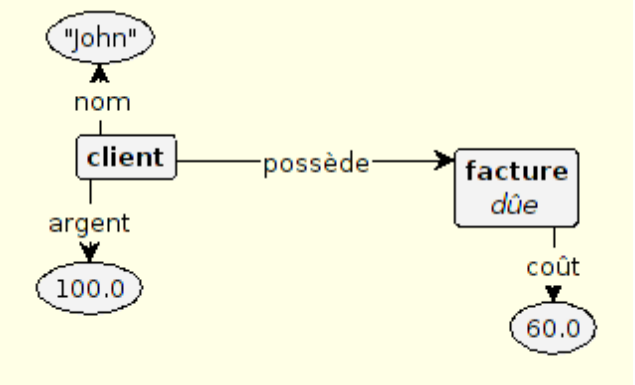


FIGURE 2.6 – Graphe attribuée selon une vue éclatée

Le moteur adopte une vue éclatée comme illustré dans la figure 2.6. Les attributs sont considérés comme des nœuds particuliers. Le graphe illustré ici est composé de deux nœuds de type client et facture, liés par un arc reflétant une relation de possession. Chaque attribut est relatif à un champ particulier, le client, par exemple, possède deux champs nommés argent et nom dont les valeurs sont respectivement 100 et John. Ils sont liés au nœud qu'ils qualifient via un arc attribué par le nom de leur champ (e.g., argent ou nom).

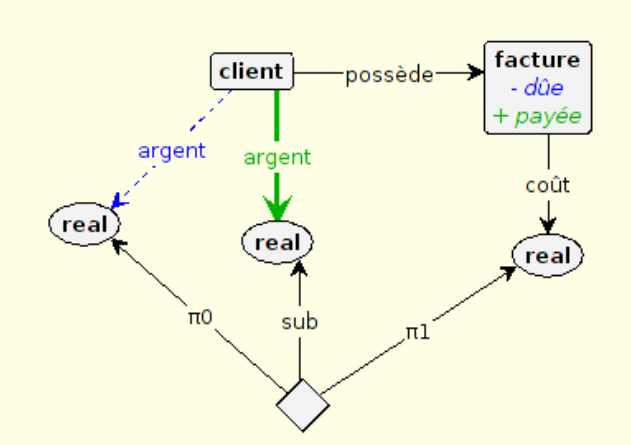
Cette vue éclatée permet d'ajouter, supprimer et manipuler les attributs par le biais de règles de réécriture. Notons que les attributs doivent encore une fois apparaître explicitement au sein de la règle afin d'être modifiables.

Une règle modifiant les attributs d'un graphe est présentée dans la figure 2.7a, le résultat de son application au graphe précédent étant illustrée dans la figure 2.7b. Cette règle décrit le paiement d'une facture par un client, son coût étant déduit du montant qu'il possède.

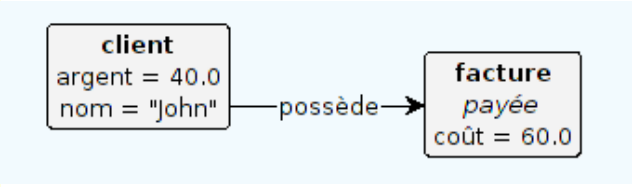
Dans GROOVE, les éléments ajoutés et supprimés sont illustrés en vert gras et bleu pointillés, respectivement. Le flag "due" est ainsi supprimé du nœud "facture", un flag "payée" y étant ajouté. De même, un attribut est supprimé du nœud client et un autre lui est ajouté, les deux qualifiant le même champ, "argent".

Les opérateurs sont des nœuds représentés sous la forme de losange et liés à leurs entrées et sorties par des arcs. L'arc de la $\imath$-ème entrée est attribué par $\pi\imath$, tandis que l'arc de la sortie qualifie l'opérateur. Ici, sub désigne l'opérateur comme étant une soustraction. Ainsi, l'attribut "argent" ajouté au client est le résultat de la soustraction du coût de la facture à son solde précédent.

23. Le second argument majeur étant la vérification de propriétés temporelles au sein du graphe d'état.



(A) Règle de paiement



(B) Résultat

FIGURE 2.7 – Manipulation d'attributs avec GROOVE

2.4.3 GMTE

GMTE [24] [6] est un moteur de recherche d'homomorphisme et de transformation de graphes attribués développé au sein du LAAS-CNRS depuis une dizaine d'années. Implémenté en C++, il ne possède pas d'interface graphique mais est utilisable en tant que librairie C++ ou via une API java.

2.4.3.1 Base théorique

Les graphes considérés ne sont pas simples, mais ne sont pas non plus des multi-graphes : il peut exister un arc d'un composant vers lui-même, mais il ne peut exister plusieurs arcs ayant la même source et la même destination.

GMTE considère des règles de transformations définies d'après le formalisme SPO mais explicitant une partie invariante d'une manière similaire à l'approche DPO. Une règle peut de plus modifier la valeur d'attributs apparaissant dans sa partie droite grâce à l'adjonction d'instructions de modification.

Il est intéressant de noter que, malgré l'adoption d'un paradigme de réécriture tiré de la théorie des catégories, la représentation interne du moteur s'appuie sur une vue ensembliste et non catégorielle [6]. En outre, GMTE n'admet que des relations injectives.

GMTE implémente un certain nombre d'extensions : instructions de connexion edNCE, NAC et conditions d'application sous la forme de prédicats sur les attributs. Une autre de ses particularités est le support d'associations inexactes [25] [101], une technique rendue populaire par son utilité dans les bases de données. Associations inexactes et instructions de connexion constituent deux des principales originalités et intérêts majeurs de GMTE.

24. *Graph Matching and Transformation Engine*, homepage : homepages.laas.fr/khalil/GMTE/index.php ?n=GMTE.HomePage
25. *inexact pattern matching*

2.4.3.2 Efficacité

GMTE permet en particulier d'exécuter simultanément une règle à un graphe selon tous les homomorphismes acceptables. Ce mécanisme induit un net gain d'efficacité dans le cadre de transformation de modèle ou de transformation de vue architecturale [102, 103], par exemple.

Outre ces domaines, GMTE se fixe comme objectif et domaine d'efficacité la recherche de petits ou moyens motif au sein de grand graphe. Les évaluations expérimentales présentées par Guennoun dans [6] suggèrent que cet outil est significativement plus efficace qu'AGG dans ces conditions. Ces comparaisons ont été néanmoins effectuées en utilisant l'interface graphique d'AGG. Guennoun défend ce gain temporel en avançant deux explications : le temps nécessaire à AGG pour la traduction en problème de satisfaction de contraintes, ainsi qu'une meilleure optimisation de l'implémentation générale du moteur.

2.4.4 Utilisation au sein de ce manuscrit

Dans les travaux présentés ici, nous serons amenés à utiliser AGG et GMTE afin d'implémenter et tester certaines des contributions élaborées.
En particulier, l'implémentation de surcouches à ces deux outils a permis la comparaison expérimentale de leurs méthodes natives pour la modification d'attributs avec une nouvelle méthodologie que nous proposons.

GROOVE présente encore quelques instabilités. Plus spécifiquement, son support des formats standards pour la définition de graphe est plus qu'erratique, ce qui complexifie le benchmarking sous la forme de tests et comparaisons systématiques. Pour cette raison, il a dû être temporairement écarté.

2.5 Synthèse

Ce manuscrit propose une réflexion autour de la représentation de systèmes dynamiques pour leur gestion autonome. Dans ce cadre, nous nous situons donc au niveau de la base de connaissance.

Nous adoptons une représentation basée graphe dans laquelle l'état du système à un instant donné, ses évolutions et l'ensemble de ses états acceptables sont caractérisés par un graphe, des règles de réécritures et une grammaire de graphe.

Pour la réécriture, nous nous inscrivons dans la continuité des tentatives de traduction de l'approche SPO en une vue opérationnelle et ensembliste, proche de l'implémentation.

Chapitre 3

Modélisation de systèmes dynamiques

Application au traitement d'évènements complexes et à l'auto-optimisation

3.1 Introduction

Ce premier chapitre est dédié à l'introduction et l'illustration du modèle utilisé dans ces travaux.

Dans un premier temps, ses fondements sont présentés et illustrés au travers d'exemples dans la section 3.2. Ce modèle est basé sur les graphes, ceux-ci lui conférant un aspect visuel facilitant une compréhension instinctive de situation et concepts complexes. Du fait de sa formalité, il ne souffre pas d'ambiguïté et permet de concevoir, étudier et gérer rigoureusement un système. Les composants du système, leurs relations et leurs propriétés sont représentés sous la forme d'un graphe conceptuel dirigé et attribué.

De part leur nature profonde, la description de systèmes dynamiques ne peut se restreindre à des topologies statiques ; elle doit intégrer ses adaptations et transformations. C'est le rôle des règles de réécriture de graphe, qui spécifient à la fois les effets d'une transformation et les circonstances dans lesquelles elle est applicable.

Nous nous inscrivons dans la continuité de tentatives de traduction d'approches catégorielles en une vue ensembliste et opérationnelle, concrète et plus proche de l'implémentation. Nous adoptons ainsi les bases du modèle proposé dans [6].

Cette première partie théorique est suivie d'une section applicative illustrative dans laquelle nous nous intéressons à un moteur de traitement d'évènements complexes [1] (CEP) en tant que service. Le CEP est un des exemples les plus éminents des technologies permettant le traitement de flux continus, rapides et massifs de données générés par des capteurs et technologies mobiles. Le CEP consiste à appliquer des règles, ou requêtes, à ces flux de données afin de les manipuler. Le moteur CEPaaS s'inscrit dans un environnement multi-clouds impliquant de multiples clients et intégrant des problématiques d'auto-optimisation liées à la qualité de service et à la consommation énergétique du système.

1. en anglais : *Complex Event Processing*

La propriété d'auto-optimisation[2] permet à un système de s'adapter de manière autonome afin d'accomplir au mieux sa fonction. La notion d'efficacité à optimiser permet de répondre à différents critères. En particulier, il peut s'agir de trouver un compromis entre performance (rapidité, qualité de service, qualité d'expérience...) et consommation de ressources (matérielles, énergétiques, pécuniaires...) (e.g., [104]).

Plus précisément, nous concevons et décrivons dans la section 3.3 le module responsable de l'analyse et la gestion de bout en bout des requêtes au sein du moteur CEPaaS. Utilisant une représentation haut niveau abstrayant opérateurs et requêtes, il s'affranchit en particulier de leurs langages de définition. Cette représentation abstraite exhibe uniquement les informations nécessaires à la gestion du système en s'appuyant sur une classification innovante des opérateurs CEP. Ce module intègre diverses politiques d'auto-optimisation dont un échantillon est finalement proposé.

3.2 Caractérisation formelle d'un système et de ses évolutions : les graphes et leurs transformations

3.2.1 Graphes partiellement attribués

Les graphes constituent un formalisme décrivant l'état d'un système dont les propriétés et caractéristiques des éléments sont représentées par des attributs. Chaque attribut peut avoir une valeur définie (i.e., fixée à l'instant donné) ou non, sa valeur étant représentée par une constante ou une variable, respectivement.

Contrairement à modèle présenté dans [6], il n'est pas imposé dans ce manuscrit que chacun des éléments d'un graphe soit attribué. Celui-ci est donc partiellement attribué.

Par la suite, les conventions suivantes sont adoptées :

Conventions et notations.
La notation canonique où, pour tout ensemble E, $|E|$ désigne sa cardinalité est utilisée ici.
Par défaut, les intervalles considérés sont des intervalles d'entiers.

La définition ci-après formalise la notion de graphe dirigé partiellement attribué, et introduit les notations associées.

Définition 3.1. (Graphe dirigé et partiellement attribué) Un graphe dirigé et partiellement attribué G est défini comme un système (V, E, Att) où :

1. V et $E \subseteq V^2$ correspondent respectivement aux ensembles des nœuds et arcs du graphe[3], de sorte que (v_1, v_2) désigne un arc de v_1 vers v_2.

Soit EL^{att} l'ensemble des éléments (arc ou nœud, $EL^{att} \subseteq V \cup E$) attribués de G. Pour tout $el \in EL^{att}$, avec $N \in \mathbb{N}$ le nombre d'attributs lui étant liés :

— $Att_i^{el} = (A_i^{el}, D_i^{el})$ avec $i \in [1, N]$, est le i-ème attribut de l'élément el. A_i^{el} est sa valeur et D_i^{el} sa nature ou domaine de définition tel que $A_i^{el} \in D_i^{el}$.

— $Att^{el} = \bigcup_{i \in [1,N]} (A_i^{el}, D_i^{el})$ est l'ensemble des attributs de l'élément attribué el. Sa cardinalité $|Att^{el}|$ est le nombre d'attributs de el, soit N.

Finalement,

2. en anglais : *self-optimization*
3. V et E sont des notations canoniques provenant des termes anglais *Vertexes* et *Edges*

2. $Att = \bigcup_{el \in EL^{att}} Att^{el}$ est la famille des ensembles d'attributs Att^{el} pour tout élément attribué el, de sorte que $|Att| = |EL_{att}| < |V| + |E|$.

Afin d'éviter toute ambiguïté du fait de l'introduction d'attributs, un **sous-graphe** $G^S = (V^S, E^S, Att^S)$ du graphe $G = (V, E, Att)$ ayant EL^{att} pour ensemble d'éléments attribués est un graphe vérifiant $V^S \subseteq V$, $E^S \subseteq (E \cap (V^S)^2)$ et $Att^S = \{Att^{el} | el \in EL^{att} \cap (V^S \cup E^S)\}$. Ainsi, les attributs d'un élément sont conservés s'il appartient au sous-graphe.

Pour faciliter la manipulation des graphes et de leurs attributs, les conventions et notations suivantes sont adoptées :

Conventions et notations.
Pour tout attribut $Att = (A, D)$, $Const(Att)$ ou $Const(A)$ est un booléen égal à vrai si et seulement si Att a une valeur fixée, c'est-à-dire si A est une constante. De plus, la valeur d'un attribut fixé sera notée entre guillemets dans les exemples.
Un nœud v_1 et un arc (v_1, v_2) attribués peuvent être notés respectivement $v(Att^{v_1})$ et $v_1 \xrightarrow{Att^{(v_1,v_2)}} v_2$.
Un graphe attribué peut être désigné par (V, E) où V et E sont les éléments du graphe notés comme ci-dessus.
Par défaut et à moins d'indication contraire explicite, les ensembles de nœuds de deux graphes sont considérés disjoints.

L'exemple suivant et la figure 3.1 illustrent la notion de graphe partiellement attribué utilisée dans ce manuscrit.

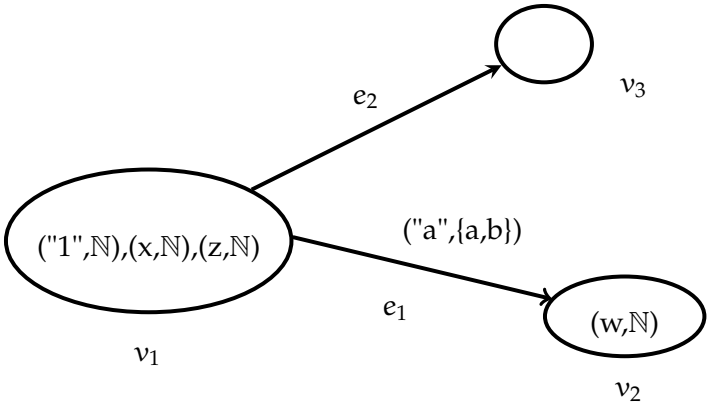


FIGURE 3.1 – Un graphe dirigé et partiellement attribué

Exemple 3.1. *Soit un graphe* $G = (V, E, Att)$ *tel qu'illustré dans la figure 3.1. D'après cette dernière, le graphe est constitué de trois nœuds* $V = \{v_1, v_2, v_3\}$ *et deux arcs* $E = \{e_1 = (v_1, v_2), e_2 = (v_1, v_3)\}$.

Trois de ces éléments, v_1, v_2 *et* e_1, *sont attribués. En posant* $EL^{att} = \{v_1, v_2, e_1\}$, *on obtient donc* $Att = \bigcup_{el \in EL^{att}} Att^{el}$. *En particulier,* v_1 *possède trois attributs (notés à l'intérieur du nœud), d'où* $Att^{v_1} = \bigcup_{i \in [1,3]} Att_i^{v_1}$. *Plus précisément,* $Att^{v_1} = \{("1", \mathbb{N}), (x, \mathbb{N}), (z, \mathbb{N})\}$. *Cela signifie que le premier attribut, noté* $Att_1^{v_1}$, *a pour valeur fixée* $A_1^{v_1} = 1$ *et comme domaine de définition* $D_1^{v_1} = \mathbb{N}$. *Ses deux autres attributs n'ont pas de valeur fixée pour le moment, celles-ci étant respectivement désignées par les deux variables* x *et* z.
Les autres attributs sont comme suit : $Att^{v_2} = \{(w, \mathbb{N})\}$ *et* $Att^{e_1} = \{("a", \{a, b\})\}$. *En particulier, le nœud* v_3 *et l'arc* e_2 *ne possèdent pas d'attribut, ce que permet le caractère partiel de l'attribution de* G.

La figure 3.1 ne présente pas d'ambiguïté et définit graphiquement le graphe G *dont l'expression textuelle est établie ci-dessus. D'après la dernière convention introduite,* G *peut également être noté* (V_{att}, E_{att}) *où* $V_{att} = \{v_1(("1", \mathbb{N}), (x, \mathbb{N}), (z, \mathbb{N})), v_2((w, \mathbb{N})), v_3\}$ *et* $E_{att} = \{e_1 = v_1 \xrightarrow{("a",\{a,b\})} v_2, e_2 = v_1 \to v_3\}$.

Notons que les noms des nœuds et des arcs sont purement arbitraires et conventionnels : ils ne portent pas de valeur intrinsèque.

3.2.2 Relations entre graphes

Les transformations de graphes forment une des clés de la modélisation de système dynamiques par des graphes. Elles nécessitent la présence de certaines relations entre graphes, liées par exemple à l'existence d'un motif dans un graphe.

Les concepts introduits dans [6] sont parfois informels et ne permettent pas de prendre en compte toutes les subtilités de la réécriture de graphe.

Par conséquent, nous formalisons dans cette sous-section les concepts d'unification, d'affectation et homomorphismes définis dans [6]. Ces définitions sont également étendues afin de permettre l'attribution partielle, la considération d'attributs variables dans le graphe cible représentant le système (et non seulement dans les motifs) et de relations liées à des associations non injectives.

3.2.2.1 Associations d'éléments

Lors de la recherche d'un motif dans un graphe, le motif lui-même est décrit sous la forme d'un graphe. Cette recherche fait donc intervenir des correspondances entre les éléments de ces deux graphes. À son tour, la recherche de correspondances entre éléments entraîne des associations entre leurs attributs. Le résultat de telles associations est décrit par un ensemble d'identifications.

Définition 3.2. (Ensemble d'identifications) Un ensemble d'identifications I est un ensemble de couple de valeurs d'attributs.

Le terme "identification" provient du fait que l'occurrence d'un couple $(A, \tilde{A})$ dans I signifie que A est identifié à $\tilde{A}$, de sorte que A et $\tilde{A}$ sont considérés équivalents.

Un ensemble d'identifications I peut être assimilé à une relation sur un ensemble de valeurs dont la clôture transitive [4] définit une relation d'équivalence. Pour n'importe quelle valeur d'attribut A, notons I_A la classe d'équivalence à laquelle appartient A d'après la relation d'équivalence induite par I. I_A est l'ensemble des valeurs $\tilde{A}$ équivalent à A selon la clôture transitive de la relation induite par I. Son expression formelle, détaillée et expliquée dans le tableau 3.1, est :
$I_A = \{\tilde{A} | \exists N \in [1, |I|], \exists((A^1_k, A^2_k))_{k\in[1,N]} \in I^N, A^1_1 = A \wedge \forall j \in [1, N-1], A^2_j = A^1_{j+1} \wedge A^2_N = \tilde{A}\}$.

En particulier et par définition, $\tilde{A} \in I_A \implies I_{\tilde{A}} = I_A$.

Un ensemble d'identifications est le résultat de l'association d'éléments d'un graphe ou de leurs attributs formalisée par la notion d'unification.

Définition 3.3. (Unification d'attributs) Deux attributs $Att_1 = (A_1, D_1)$ et $Att_2 = (A_2, D_2)$ sont unifiables si et seulement si :

4. On appelle clôture transitive d'une relation binaire R la plus petite relation binaire transitive contenant R.

Expression formelle	Signification
$I_A = \{\tilde{A}\|$	I_A est l'ensemble des valeurs $\tilde{A}$ telles que
$\exists N \in [1, \|I\|]$, $\exists (A_k^1, A_k^2))_{k\in[1,N]} \in I^N$,	il existe une suite d'éléments de I
$A_1^1 = A \wedge$	partant de A,
$\forall j \in [1, N-1], A_j^2 = A_{j+1}^1$	telle que le second membre de tout élément est égal au premier membre de l'élément suivant
$\wedge A_N^2 = \tilde{A}\}$	et arrivant à $\tilde{A}$

TABLE 3.1 – Classe d'équivalence induite par un ensemble d'identifications

1. Ils ont le même type : $D_1 = D_2$
2. Si les deux ont des valeurs définies, elles sont égales : $\mathtt{Const}(A_1) \wedge \mathtt{Const}(A_2) \implies A_1 = A_2$.

Si ces attributs sont unifiables, le résultat de leur unification, noté $\mathtt{Unif}(A_1, A_2)$ ou $\mathtt{Unif}(\mathtt{Att}_1, \mathtt{Att}_2)$, est l'ensemble d'identifications :

— $\emptyset$ si et seulement si $A_1 = A_2$;
— $\{(A_1, A_2), (A_2, A_1)\}$ sinon.

Par définition, la relation induite par un ensemble d'identifications résultant d'une unification est symétrique.

Exemple 3.2. *Les attributs* $\mathtt{Att}_1 = (x, \mathbb{N})$ *et* $\mathtt{Att}_2 = (y, \mathbb{N})$ *sont unifiables et le résultat de leur unification,* $\mathtt{Unif}(\mathtt{Att}_1, \mathtt{Att}_2)$*, est égal à* $\{(x, y), (y, x)\}$*, signifiant que* x *est équivalent à* y*.*

Un ensemble d'identifications est consistant si toute classe d'équivalence de la clôture transitive de la relation qu'il définit contient au plus une constante. En d'autres termes, aucune constante n'est équivalente à une constante différente.

Définition 3.4. (Ensemble d'identifications consistant) Un ensemble d'identifications I est consistant si et seulement si pour toute valeur A apparaissant dans I, $\forall \tilde{A} \in I_A, \mathtt{Const}(A) \wedge \mathtt{Const}(\tilde{A}) \implies A = \tilde{A}$.

Exemple 3.3. *L'ensemble d'identifications* $I = \{(y, "1"), ("1", y), (x, y), (y, x), (z, y), (y, z)\}$ *ne contient qu'une seule valeur constante ; il est donc nécessairement consistant.* $\tilde{I} = I \cap \{(y, "2"), ("2", y)\}$*, par exemple, n'est pas consistant. Il impliquerait l'équivalence de* y *à la fois à 1 et 2, ce qui n'est pas possible.*

Finalement, deux éléments sont unifiables si chacun de leurs attributs sont unifiables deux à deux, et si leur unification produit un ensemble d'identifications consistant.

Définition 3.5. (Unification d'éléments) Deux éléments el_1 et el_2 sont unifiables si au moins l'un des deux n'est pas attribué, ou si les trois conditions suivantes sont remplies :

1. $|\mathtt{Att}^{el_1}| = |\mathtt{Att}^{el_2}|$,

2. $\forall i \in [1, |Att^{el_1}|], Att_i^{el_1}$ et $Att_i^{el_2}$ sont unifiables et
3. $\bigcup_{i\in[1,|Att^{el_1}|]} Unif(A_i^{el_1}, A_i^{el_2})$ est consistant.

Le résultat de cette unification, notée $Unif(el_1, el_2)$ est l'ensemble d'identifications :
— $\emptyset$ si au moins l'un des deux éléments n'est pas attribués ;
— $\bigcup_{i\in[1,|Att^{el_1}|]} Uniff(A_i^{el_1}, A_i^{el_2})$ dans le cas contraire.

Un élément non attribué peut par conséquent être unifié à n'importe quel élément de même nature. En plus de leur intérêt premier, ils peuvent donc être également utilisés comme motifs génériques.

Exemple 3.4. *Soient les nœuds* $v_1(("1", \mathbb{N}), (x, \mathbb{N}), (z, \mathbb{N}))$ *de l'exemple 3.1 et* $v_4((y, \mathbb{N}), (y, \mathbb{N}), (y, \mathbb{N}))$. *L'union des ensembles d'identifications produits par l'unification de leurs attributs, notée* I, *est égale à* $\{(y, "1"), ("1", y), (x, y), (y, x), (z, y), (y, z)\}$. *D'après l'exemple 3.3,* I *est consistant et les nœuds* v_1 *et* v_4 *sont donc unifiables.*

Comme vu précédemment, identifier les éléments de graphes requiert l'association d'attributs potentiellement variables. Une affectation est la fonction impactant ces identifications en substituant à chaque variable une valeur appropriée. Chaque attribut équivalent à un autre dont la valeur est définie voit sa valeur remplacée par la constante correspondante. Les valeurs d'attributs dont la classe d'équivalence ne contient pas de constante sont remplacées par un représentant choisi arbitrairement. Ce choix est purement syntaxique. Notons $I^r_{\tilde{A}}$ le représentant de la classe d'équivalence $I_{\tilde{A}}$ de sorte que pour tout $\tilde{A} \in I_A$, (rappelons que $I_{\tilde{A}}$ est alors égal à I_A) $I^r_{\tilde{A}} = I^r_A$. Cette discussion est formalisée dans la définition ci-après.

Définition 3.6. (Affectation induite par un ensemble d'identifications consistant) Pour tout ensemble d'identifications consistant I, l'affectation induite par I, notée Aff_I, est une application sur l'ensemble des graphes, de leurs attributs et des valeurs de ces derniers, telle que pour tout graphe $G = (V, E, Att)$ dont l'ensemble d'éléments attribués est S,
$Aff_I(G) = (V, E, Aff_I(Att))$ où $\tilde{Att} = \bigcup_{el\in S, i\in[1,|Att^{el}|]}(Aff_I(A_i^{el}), D_i^{el})$ avec :
— $Aff_I(A_i^{el}) = A_i^{el}$ si $I_{Att_i^{el}} = \emptyset$,
— $Aff(A_i^{el}) = A \in I_{A_i^{el}}$ tel que $Const(A)$ si une telle valeur existe,
— $Aff_I(A_i^{el}) = I^r_{A_i^{el}}$ sinon.

Exemple 3.5. *Avec le graphe* G *et l'ensemble d'identification* I *définis dans les exemples 3.1 et 3.3 respectivement,*
$Aff_I(G) = (\{v_1(("1", \mathbb{N}), ("1", \mathbb{N}), ("1", \mathbb{N})), v_2((w, \mathbb{N})), v_3\}, E)$.

3.2.2.2 Recherche de motifs : morphismes de graphes

Trouver un motif décrit par un graphe à l'intérieur d'un autre, appelé graphe hôte, est habituellement lié à la notion de morphisme. Un morphisme de graphe non attribué de $G = (V, E)$ vers $H = (V_H, E_H)$ est caractérisé par une fonction $f : V \to V_H$ conservant la structure de G. Cela signifie que s'il existe un arc entre deux nœuds de G, il doit exister un arc dans H entre leurs images par f. En présence d'attributs variables, les éléments de graphes peuvent être assimilés via le concept d'unification, les associations d'attributs ainsi générées devant être cohérentes. En adoptant les conventions suivantes, ces concepts sont intégrés à la définition de morphisme comme détaillé dans la définition 3.7.

Conventions et notations.
Toute fonction introduite dans la suite est implicitement totale.
Par abus de notation, afin de les alléger et comme cela ne crée pas d'ambiguïté, toute fonction $f : A \to B$ est assimilée à son extension canonique : $A \cup A^2 \to B \cup B^2$ telle que $\forall (a_1, a_2) \in A^2, f((a_1, a_2)) = (f(a_1), f(a_2))$.
De plus, nous adoptons les notations canoniques où $Dom(f)$ désigne son domaine, $Codom(f)$ son codomaine et $Im(f)$ l'image de $Dom(f)$ par f. Ces notations ne prennent pas en compte l'assimilation proposée ci-dessus, de sorte que $Dom(f) = A$, $Codom(f) = B$ et $Im(f) = f(A) \subseteq Codom(f)$.

Définition 3.7. (Morphisme cohérent de graphes partiellement attribués) Il existe un morphisme m entre deux graphes $G = (V, E, Att)$ et $H = (V_H, E_H, Att_H)$, noté $G \xrightarrow{m} H$ si et seulement s'il existe une fonction $f : V \to V_H$ telle que :

1. $\forall (v_a, v_b) \in E, f((v_a, v_b)) \in E_H$,

Ce morphisme est cohérent s'il existe un ensemble d'identifications consistant I tel que :

2. $\forall el \in V \cup E, v$ et $f(v)$ sont unifiables,
3. $\forall (el_a, el_b) \in (V \cup E)^2, f(el_a) = f(el_b) \implies el_a$ et el_b sont unifiables et
4. $I \supseteq I_2 \cup I_3$ avec
   — $I_2 = \bigcup_{el \in V \cup E} Unif(el, f(el))$
   — $I_3 = \bigcup_{(el_a, el_b) \in EL_3} Unif(el_a, el_b)$, où EL_3 est l'ensemble des couples unifiés par 3., soit $\{(el_a, el_b) \in (V \cup E)^2 | f(el_a) = f(el_b)\}$.

Si f est injective, m est un **homomorphisme** h et G est homomorphe à H.
Si f est bijective et si (f^{-1}, Aff) est également un (homo-)morphisme, m est un **isomorphisme**. G et H sont alors isomorphes.

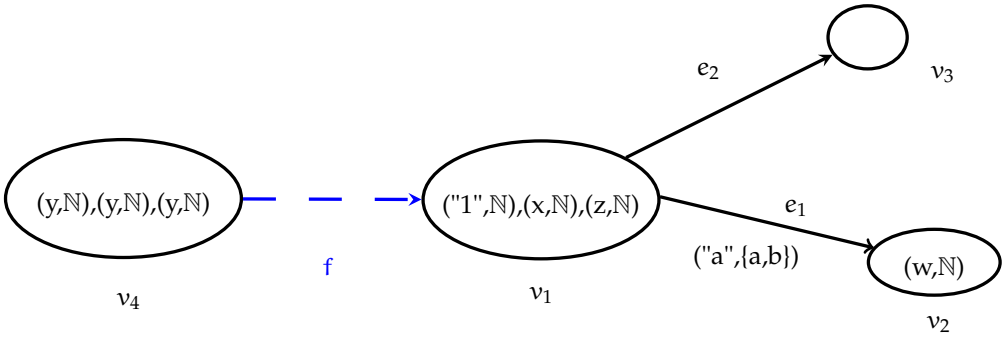


FIGURE 3.2 – Un (homo)morphisme cohérent de graphes attribués

Une illustration de morphisme cohérent, présenté dans la figure 3.2, est détaillée dans l'exemple suivant.

Exemple 3.6. *Soient le motif défini par le graphe* $M = (\{v_4((y, \mathbb{N}), (y, \mathbb{N}), (y, \mathbb{N}))\}, \emptyset)$ *et* G *le graphe défini dans l'exemple 3.1. Soit* f *la fonction, représentée en bleu dans la figure 3.2,* $f : V_M \to V_G$ *telle que* $f(v_4) = v_1$. *Comme vu dans l'exemple 3.4, ces nœuds sont unifiables et le résultat de leur unification et l'ensemble d'identifications consistant* I *introduit dans l'exemple 3.3.* $m = (f, I)$ *est donc un morphisme cohérent de* M *vers* G. *De plus,* f *étant injective,* m *est un homomorphisme.*

Remarque 3.1. Contrairement aux morphismes classiques, la relation définie par les morphismes cohérents n'est pas transitive, du fait d'unifications potentiellement inconsistantes. En d'autres termes, l'existence de deux morphismes cohérents $m_1^2 = (f_1^2, I_1^2)$ et $m_2^3 = (f_2^3, I_2^3)$ tels que $G_1 \xrightarrow{m_1^2} G_2$ et $G_2 \xrightarrow{m_2^3} G_3$ implique nécessairement l'existence mais pas la cohérence d'un troisième morphisme m_1^3 tel que $G_1 \xrightarrow{m_1^3} G_3$. Cependant, en posant Aff_i^j l'affectation induite par I_i^j, la cohérence d'un morphisme $m_2^2 = (f_2^2, I_2^2)$ [5] de $Aff_1^2(G_2)$ vers $Aff_2^3(G_2)$ implique la cohérence de $m_1^3 = (f_2^3 \circ f_2^2 \circ f_1^2, I_2^3 \cup I_2^2 \cup I_1^2)$. Ces considérations sont schématisées dans la figure 3.3.

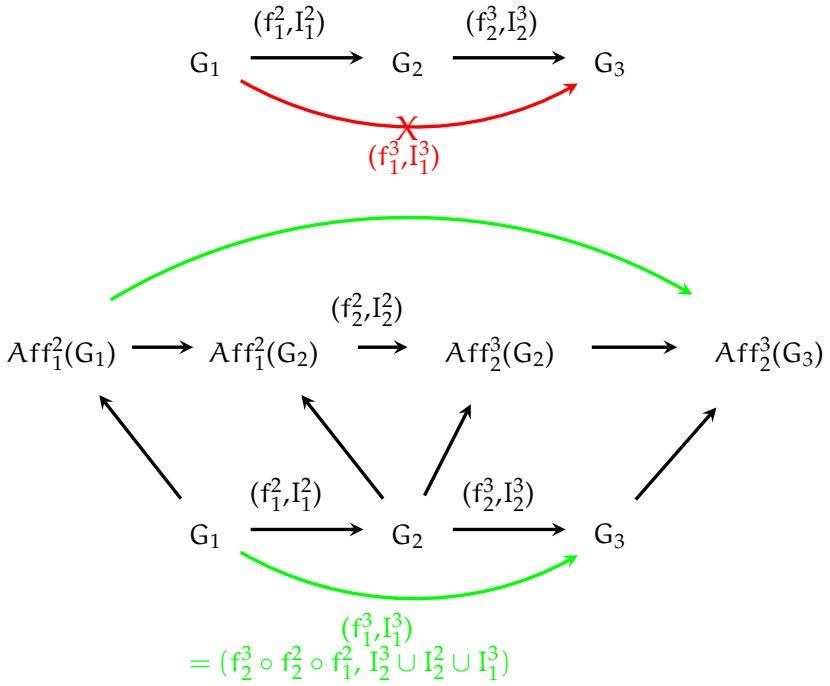


FIGURE 3.3 – Morphismes cohérents et transitivité

Par la suite, tout morphisme introduit sera implicitement cohérent.

3.2.2.3 Compatibilité

Le collage [6] de deux graphes est une opération centrale dans les approches de transformation considérées. Afin de le formaliser opérationnellement, il est nécessaire d'introduire une autre relation n'apparaissant pas dans [6] : la compatibilité.

La compatibilité entre deux graphes G_1 et G_2 revient à considérer deux de leurs sous-graphes induits G_1^S et G_2^S et d'associer leurs nœuds via une fonction f. Cette association doit alors vérifier une relation moins contrainte qu'un isomorphisme : si un arc existe entre deux nœuds de G_1^S, il n'est pas nécessaire qu'il y en ait un entre leurs images par f dans G_2^S. Néanmoins, si un tel arc existe, les deux doivent être unifiables.

Définition 3.8. (Graphes compatibles) Deux graphes $G_1 = (V_1, E_1, Att_1)$ et $G_2 = (V_2, E_2, Att_2)$ sont (f, I)-compatibles si et seulement s'il existe un ensemble d'identifications consistant I et une fonction $f : V_1^S \subseteq V_1 \to V_2^S \subseteq V_2$, tels que :

5. f_2^2 pouvant être la fonction identité
6. en anglais : gluing

1. $\forall v \in V_1^S$, v et $f(v)$ sont unifiables,
2. $\forall e \in E_1 \cap (V_1^S)^2, f(e) \in E_2 \implies e$ est unifiable avec $f(e)$,
3. $\forall (el_a, el_b) \in (V_1^S)^2 \cup (E_1 \cap (V_1^S)^2)^2, f(el_a) = f(el_b) \implies el_a$ et el_b sont unifiables et
4. $I \supseteq I_a \cup I_b$ avec
   — $I_a = \bigcup_{el \in EL_{1,2}} Unif(el, f(el))$, où $EL_{1,2}$ est l'ensemble des éléments concernés par 1. et 2., soit $V_1^S \cup \{e \in E_1 \cap (V_1^S)^2 | f(e) \in E_2\}$.
   — $I_b = \bigcup_{(el_a, el_b) \in EL_3} Unif(el_a, el_b)$ où EL_3 est l'ensemble des couples unifiés par 3., soit $\{(el_a, el_b) \in (V_1^S)^2 \cup (E_1 \cap (V_1^S)^2)^2 | f(el_a) = f(el_b)\}$.

En particulier, si G_1 et G_2 sont (f, I)-compatibles et si f est bijective, alors G_2 et G_1 sont (f^{-1}, I)-compatibles. Notons qu'il existe nécessairement, pour tout couple de graphes, une compatibilité C tels que ces derniers sont C-compatibles, quitte à ce que $Dom(f) = I = \emptyset$.

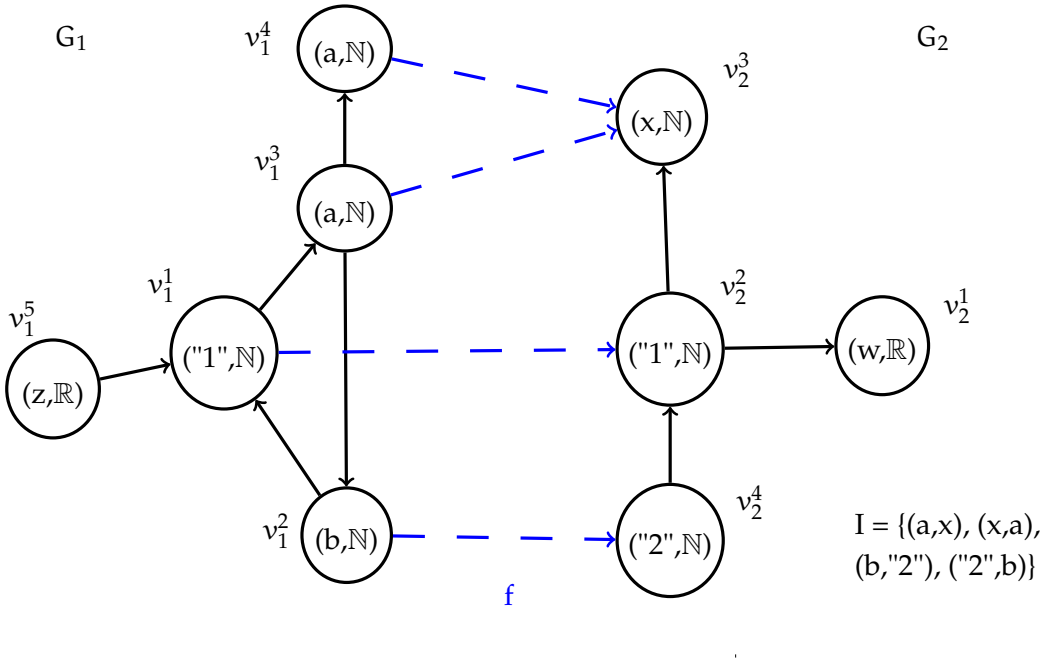


FIGURE 3.4 – Deux graphes compatibles

La figure 3.4 offre une illustration de graphes compatibles détaillée dans l'exemple suivant. Par soucis de lisibilité, les attributs des arcs n'ont pas été représentés. Ils seront considérés égaux et constants (n'impactant ainsi pas le processus d'identification).

Exemple 3.7. *Soient G_1 et G_2 les deux graphes définis dans la figure 3.4. Soient V_1^S et V_2^S les ensembles de nœuds nommés v_1^1, v_1^2, v_1^3, et v_1^4 et v_2^2, v_2^3, et v_2^4, respectivement.*
La fonction f, représentée en bleu dans la figure, a pour domaine V_1^S et pour image V_2^S. Elle associe les nœuds v_1^1 à v_2^2, v_1^2 à v_2^4, v_1^3 à v_2^3 et v_1^4 à v_2^3. On notera que f n'est pas injective, v_2^3 étant l'image par f à la fois de v_1^3 et v_1^4.
Les éléments associés via f sont unifiables, et l'ensemble d'identification résultant de ces associations, $I = \{(a, x), (x, a), (b, "2"), ("2", b)\}$, est consistant. G_1 et G_2 sont donc (f, I)-compatibles.

Néanmoins, f ne peut pas caractériser un morphisme entre le sous graphe de G_1 induit par V_1^S et G_2. En effet, malgré l'existence de deux arcs (v_1^3, v_1^2) et (v_1^3, v_1^4) dans $E_{G_1} \cap (V_1^S)^2$, il n'existe pas d'arcs correspondants (v_2^3, v_2^4) et (v_2^3, v_2^3) dans G_2.

3.2.3 Transformations de graphes

Une configuration d'un système pouvant être modélisée par un graphe, ses évolutions peuvent l'être par des transformations de graphes, généralement symbolisées par des règles de réécriture de graphe.

L'expression opérationnelle de ces règles n'est que très informelle dans la proposition considérée [6]. Pour pouvoir fournir une définition rigoureuse, nous introduisons et formalisons dans un premier temps deux opérations sur lesquelles elles se basent.

3.2.3.1 Expansion et restriction

Les deux premières transformations présentées ici sont deux opérations binaires sur l'ensemble des graphes : l'expansion et la restriction. La première correspond au collage ou à la fusion de deux graphes, qui est similaire à l'union dans la théorie des ensembles classiques. La seconde est son pendant, à rapprocher de l'intersection. Ces deux opérations reposent sur une similarité entre éléments plutôt qu'une stricte égalité. L'identification des éléments et sous graphes analogues s'effectue grâce à la notion de compatibilité précédemment définie.

L'expansion est par conséquent un opérateur binaire sur l'ensemble des graphes. Cet opérateur dépend d'une compatibilité, tel que défini ci-après.

Définition 3.9. (Expansion en fonction d'une compatibilité C**,** $\uparrow_C$**)** Pour tous graphes C-compatibles G_1 et G_2 avec $C = (f, I)$, pour n'importe lesquels de leurs sous-graphes, $G_1^S = (V_1^S, E_1^S, Att_1^S)$ et $G_2^S = (V_2^S, E_2^S, Att_2^S)$ dont les éléments attribués sont respectivement EL_1^{att} et EL_2^{att},
$G_1^S \uparrow_C G_2^S = (V, E, Att)$ où :

- $V = V_2^S \cup \{v \in V_1^S | v \notin Dom(f) \vee f(v) \notin V_2^S\}$.
  V est l'union de l'ensemble des nœuds de G_2^S avec les nœuds de G_1^S n'ayant pas d'image par f ou dont l'image par f n'est pas un nœud de G_2^S. Ceci permet d'éviter les doublons et de gérer le fait que f puisse ne pas être injective.
- $E = \{(v_a, v_b) \in V^2 | (v_a, v_b) \in E_1^S \cup E_2^S \vee$
  $(\exists v_c \in V_1^S \cap Dom(f), P_1^1(v_a, v_b, v_c) \vee P_1^2(v_a, v_b, v_c)) \vee$
  $(\exists (v_c, v_d) \in E_1^S \cap Dom(f)^2, P_2(v_a, v_b, v_c, v_d))\}$ où
  - $\forall (v_a, v_b) \in V^2, \forall v_c \in V_1^S \cap Dom(f), P_1^1(v_a, v_b, v_c) = (f(v_c) = v_a \wedge (v_c, v_b) \in E_1^S)$.
    $P_1^1(v_a, v_b, v_c)$ signifie qu'il existe un arc dans G_1^S dont la destination est v_b et la source v_c, un nœud dont l'image par f est v_a.
  - $\forall (v_a, v_b) \in V^2, \forall v_c \in V_1^S \cap Dom(f), P_1^2(v_a, v_b, v_c) = (f(v_c) = v_b \wedge (v_a, v_b) \in E_1^S)$.
    $P_1^2(v_a, v_b, v_c)$ signifie qu'il existe un arc dans G_1^S dont les extrémités sont v_a et v_c, un nœud dont l'image par f est v_b.
  - $\forall (v_a, v_b) \in V^2, \forall (v_c, v_d) \in E_1^S \cap Dom(f)^2, P_2(v_a, v_b, v_c, v_d) = (f((v_c, v_d)) = (v_a, v_b))$.
    $P_2(v_a, v_b, v_c, v_d)$ signifie qu'il existe un arc (v_c, v_d) dans G_1^S dont les extrémités ont pour images par f les nœuds v_a et v_b.

  E contient les arcs des deux graphes sur lesquels l'opération est appliquée, ainsi que tout arc dont une ou deux extrémités est l'image par f de l'extrémité d'un arc de G_1^S.
- $Att = \{Att^{el}, el \in V \cup E | P_3(Att^{el}) \vee P_4(Att^{el}) \vee P_5(Att^{el})\}$ où $\forall el \in V \cup E$:
  - $P_3(Att^{el}) = \exists i \in [1, 2], el \in EL_i^{att} \wedge Att^{el} = Aff_I((Att_i^S)^{el})$.
  - $P_4(Att^{el}) = el \in (V_2^S \cup E_2^S) \backslash EL_2^{att} \wedge$
    $\exists el_a \in ((V_1^S \cap Dom(f)) \cup (E_1^S \cap Dom(f)^2)) \cap EL_1^{att}, f(el_a) = el \wedge$
    $Att^{el} = Aff_I((Att_1^S)^{el_a}))$.

Tout élément non attribué étant l'image d'un élément attribué "hérite" ainsi des attributs de ce dernier.

— $P_5(el) = (el = (v_a, v_b) \in E \backslash (E_1^S \cup E_2^S) \wedge$
$(\exists v_c \in V_1^S \cap Dom(f), P_1^1(v_a, v_b, v_c) \wedge (v_c, v_b) \in EL_1^{att} \wedge Att^{el} = Aff_I((Att_1^S)^{(v_c, v_b)})) \vee$
$(\exists v_c \in V_1^S \cap Dom(f), P_1^2(v_a, v_b, v_c) \wedge (v_a, v_c) \in EL_1^{att} \wedge Att^{el} = Aff_I((Att_1^S)^{(v_a, v_c)})) \vee$
$(\exists (v_c, v_d) \in E_1^S \cap Dom(f)^2 \cap EL_1^{att}, P_2(v_a, v_b, v_c, v_d) \wedge Att^{el} = Aff_I((Att_1^S)^{(v_c, v_d)}))).$

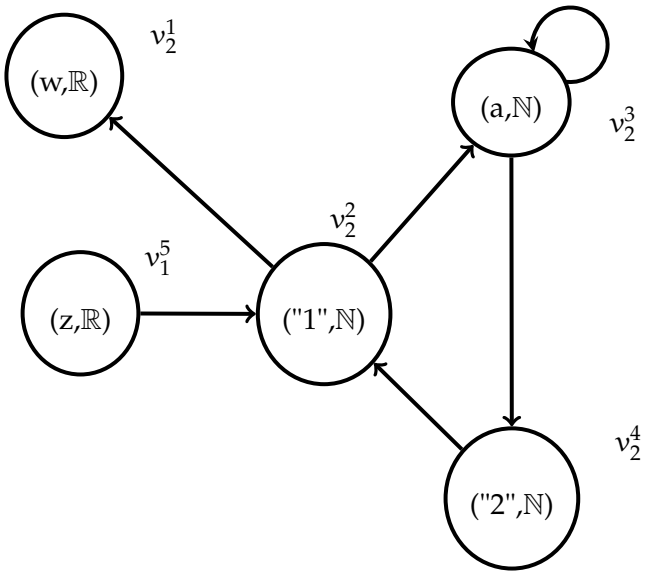


FIGURE 3.5 – $G_1 \uparrow_C G_2$, une opération d'expansion entre les deux graphes compatibles de la figure 3.4

L'exemple suivant détaille une opération d'expansion illustrée dans la figure 3.5. Les deux graphes compatibles sur lesquels l'opération est appliquée sont ceux représentés dans la figure 3.4. Ces deux figures mettent en exergue l'équivalence entre une expansion et la fusion de deux graphes.

Exemple 3.8. *Soit* G_1, G_2, f, I, V_1^S *et* V_2^S *tels que définis et illustrés dans l'exemple 3.7 et la figure 3.4. En posant* $C = (f, I)$, $G_1 \uparrow_C G_2$ *est le graphe représenté dans la figure 3.5.*
Les nœuds du graphe résultant sont ceux de G_2 *(*$v_2^1, v_2^2, v_2^3, v_2^4$*) adjoint de* v_1^5*, l'unique nœud de* G_1 *n'ayant pas d'image par* f *dans* G_2*.*
Le nœud v_2^2*, est l'image par* f *de* v_1^1*, qui est lui même une extrémité de l'arc* (v_1^5, v_1^1)*. L'arc* (v_1^5, v_2^2) *remplit donc la condition* P_1^2 *de la définition d'expansion et est par conséquent présent dans le graphe résultant. De même,* (v_2^3, v_2^3) *et* (v_2^3, v_2^4) *sont les images par* f *de deux arcs existants dans* G_1*,* (v_1^3, v_1^4) *et* (v_1^3, v_1^2)*. Ils remplissent donc la condition* P_2^2 *de la définition.*

La remarque suivante décrit quelques propriétés immédiates de l'expansion.

Remarque 3.2. Pour tous graphes G_1 et G_2, pour toute compatibilité $C = (f, I)$ telle que G_1 et G_2 sont C-compatibles, et pour tous sous graphes G_1^S de G_1 et G_2^S de G_2,

1. $G_1^S \uparrow_C G_2^S$ est invariant par Aff_I.[7]
2. il existe un morphisme, caractérisé par une certaine fonction[8] et par I, tel que $G_1^S \to G_1^S \uparrow_C G_2^S$.
3. il existe un homomorphisme, caractérisé par l'identité et par I, tel que $G_2^S \to G_1^S \uparrow_C G_2^S$.

7. D'après la définition de l'expansion, $G_1^S \uparrow_C G_2^S$ intègre l'affectation Aff_I.
8. Cette fonction coïncide avec f sur $\{v \in Dom(f) | f(v) \in V_2^S\}$ et avec l'identité sur le reste de son domaine de définition.

La restriction est un opérateur binaire sur l'ensemble des graphes. Cette opération, comparable à une intersection de graphes, est formalisée dans la définition ci-dessous.

Définition 3.10. (Restriction en fonction d'une compatibilité $C, \downarrow_C$**)** Pour tous graphes C-compatibles G_1 et G_2 avec $C = (f, I)$, pour n'importe lesquels de leurs sous-graphes, $G_1^S = (V_1^S, E_1^S, Att_1^S)$ et $G_2^S = (V_2^S, E_2^S, Att_2^S)$ dont les éléments attribués sont respectivement EL_1^{att} et EL_2^{att},
$G_1^S \downarrow_C G_2^S = (V, E, Att)$ où :

— $V = V_2^S \cap f(V_1^S \cap Dom(f))$.
V est l'ensemble des images de nœuds de G_1^S par f étant des nœuds de G_2^S.

— $E = E_2^S \cap f(E_1^S \cap Dom(f)^2)$. E est l'ensemble des arcs de G_2^S étant l'image d'au moins un arc de G_1^S par f.

— $Att = \{Att^{el}, el \in V \cup E|$
$(el \in EL_2^{att} \wedge Att^{el} = Aff_I((Att_2^S)^{el})) \vee$
$(el \notin EL_2^{att} \wedge (\exists el_a \in ((V_1^S \cap Dom(f)) \cup (E_1^S \cap Dom(f)^2)) \cap EL_1^{att}, f(el_a) = el \wedge Att^{el} = Aff_I((Att_1^S)el_a))\}$.

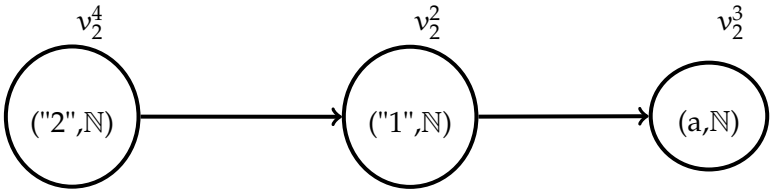


FIGURE 3.6 – $G_1 \downarrow_C G_2$, une opération de restriction entre les deux graphes compatibles de la figure 3.4

La figure 3.6 illustre le résultat d'une opération de restriction sur les deux graphes présentés dans la figure 3.4. Elle est décrite dans l'exemple suivant.

Exemple 3.9. *Soit* G_1, G_2, f, I V_1^S *et* V_2^S *tels que définis et illustrés dans l'exemple 3.7 et la figure 3.4. En posant* $C = (f, Aff)$, $G \downarrow_C \tilde{G}$ *est représenté dans la figure 3.6.*
Les éléments du graphe résultant de cette restriction sont exactement les éléments du second graphe, G_2*, étant l'image par* f *d'au moins un élément du premier,* G_1*. Ici, l'ensemble des nœuds d'intérêt est donc* $Im(f)$*, l'image par* f *du domaine de* f*, soit* v_2^2*,* v_2^3 *et* v_2^4*. De même, le graphe obtenu contient tous les arcs de* G_2 *étant l'image par* f *d'au moins un arc de* G_1*, soit* (v_2^4, v_2^2) *et* (v_2^2, v_2^3)*.*

La remarque ci-après décrit quelques propriétés immédiates de la restriction.

Remarque 3.3. Pour tous graphes G_1 et G_2, pour toute compatibilité $C = (f, I)$ telle que G_1 et G_2 sont C-compatibles, et pour tous sous-graphes G_1^S et G_2^S :

1. $G_1^S \downarrow_C G_2^S$ invariant par Aff_I.
2. Il existe un homomorphisme tel que $G_1^S \downarrow_C G_2^S \rightarrow G_2^S$.
3. Si f est injective, alors il existe un homomorphisme tel que $G_1^S \downarrow_C G_2^S \rightarrow G_2^S$.

3.2.3.2 Réécriture de graphes

Une règle de réécriture de graphe spécifie à la fois une transformation de graphe et les circonstances dans lesquelles elle peut être appliquée. À ce titre, ces règles constituent l'outil basique pour la représentation et la gestion du dynamisme dans les modèles basés sur les graphes.

Dans ce manuscrit, nous nous appliquons à traduire l'approche SPO d'un point de vue ensembliste et opérationnel. Les définitions et formalisations précédentes (notamment la restriction et l'expansion) permettent de prendre en compte l'attribution partielle et variable des graphes manipulés ainsi que des morphismes non injectifs.

Au modèle utilisé comme fondation [6], il faut donc :

— ajouter la notion de morphisme partiel et non injectif entre les différentes parties d'une règle, et étudier les conséquences de cet ajout. Un morphisme partiel de A vers B est défini comme un morphisme d'une sous algèbre de A vers B [71]. Pour les graphes, cela peut être traduit par l'existence d'un morphisme entre un sous-graphe de A et B.

— étudier les implications de la considération de graphes n'étant pas des multi-graphes, l'approche SPO étant définies pour ces derniers.

Définition 3.11. (Règle de réécriture de graphe) Une règle de réécriture de graphe r, notée $L \xrightarrow{m_r} R$, est caractérisée par un couple de graphes $(L = (V_L, E_L Att_L), R = (V_R, E_R, Att_R))$ et un morphisme $m_r = (f_r, \emptyset)$ d'un sous graphe $L^I = (V_L^I = Dom(f_r), E_L^I, Att_L^I)$ de L vers R.

Une règle r est **applicable** à un graphe $G = (V, E, Att)$ si les conditions suivantes sont remplies.

1. Il existe une image de L dans G. En fonction des différentes approches, la présence de cette image peut être formalisée par un morphisme, un homomorphisme ou un isomorphisme de sous graphe induit [9] de L vers G. Ce morphisme sera noté $m_L = (f_L, I_L)$.
2. Le morphisme m_L vérifie la condition $\forall (v_1, v_2) \in (V_L^I)^2, ((v_1, v_2) \notin E_L \wedge f_r((v_1, v_2)) \in E_R) \implies f_L((v_1, v_2)) \notin E_G$. Cette condition traduit l'impossibilité d'ajouter un arc entre deux nœuds s'il en existe déjà un [10].
3. Dans le cas général où m_r et m_L sont des morphismes non injectifs, il faut que $\forall (v_1, v_2) \in V_L^2$, $f_L(v_1) = f_L(v_2) \implies ((v_1 \notin Dom(f_r) \wedge v_2 \notin Dom(f_r)) \vee ((v_1, v_2) \in Dom(f_r)^2 \wedge f_r(v_1) = f_r(v_2)))$. Deux nœuds de L ayant la même image par m_L n'ont soit pas d'image par m_r, soit la même.

Son **application** consiste à :

1. Effacer dans G l'image par f_L de la partie de L n'appartenant pas à L^I et supprimer les potentiels arcs suspendus. Formellement, le résultat de cette étape est un graphe intermédiaire G_{tmp}, l'unique sous graphe de G ayant pour arcs et nœuds $V_{tmp} = V \backslash f_L(V_L \backslash V_L^I)$ et $E_{tmp} = (E \cap (V_{tmp})^2) \backslash f_L(E_L \backslash E_L^I)$.
2. Ajouter une copie isomorphe de la partie de R n'appartenant pas à l'image de L^I par f_r et intégrant les affectations obtenues via m_r, donnant le graphe
   $G_{tmp} \uparrow_{(f, I_r)} R$ où $f : f_L(Dom(f_r)) \to Im(f_r)$, tel que $\forall v \in f_L(Dom(f_r)), f(v) = f_r(v_L)$ avec n'importe quel $v_L \in Dom(f_r), v = f_L(v_L)$.
   L'unicité de v_L est dépendante de l'injectivité de f_L. D'après les conditions d'application, si v_L n'est pas unique, chaque v_L potentiel a la même image par f_r. La fonction f est donc bien définie.

Un élément est dit invariant vis à vis de l'application d'une règle si cette dernière ne le supprime ni ne l'ajoute. Ces éléments sont ceux de L^I et de la partie de R étant l'image par f_r de L^I. Par conséquent, ces graphes décrivent la partie du graphe hôte invariante pour l'application de r (aux fusions de nœuds et affectations d'attributs près).

9. C'est à dire que L est isomorphe à un sous graphe induit de G
10. nb : considérer des multi-graphes permet de lever cette restriction.

Conventions et notations.
Dans la suite, nous supposons que toute variable désignant la valeur d'un attribut présent dans la règle est muette, c'est à dire qu'elle n'apparaît pas dans le graphe hôte.

Les figures 3.7 et 3.8 offrent un exemple de règle de réécriture de graphe et illustre les étapes de son application. Celles-ci sont détaillées dans l'exemple suivant. Par soucis de lisibilité, les attributs des arcs n'ont pas été représentés. Ils seront considérés égaux et constants.

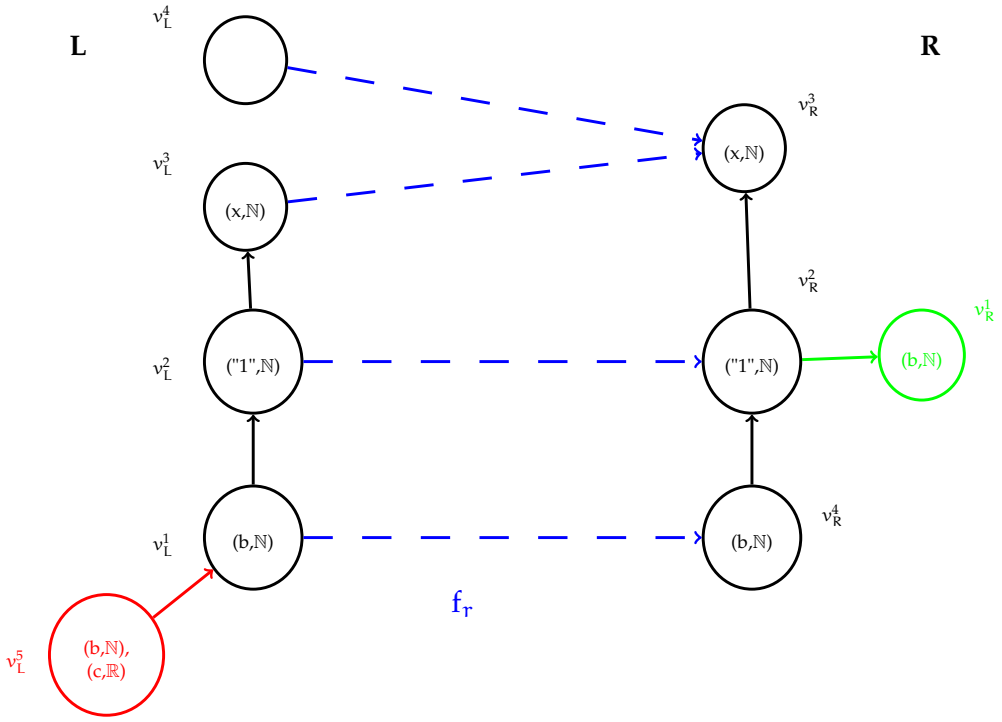


FIGURE 3.7 – Une règle de réécriture de graphe

Exemple 3.10. *La figure 3.7 décrit une règle de réécriture* $r = L \xrightarrow{m_r} R$. f_r *est la fonction associée au morphisme partiel* m_r *et représentée en bleu dans la figure et* L^I *est le sous-graphe de* L *induit par* $Dom(f_r)$. *On notera que* f_r *n'est pas injective, et qu'un nœud de* L, *nommé* v_L^4 *n'est pas attribué. Il pourra donc être associé à n'importe quel nœud.*
Un nœud de L, *noté* v_L^5 *et représenté en rouge, ne fait pas partie du domaine de* f_r. *Lors de l'application de* r, *son image sera donc supprimée, de même que l'image de* (v_L^5, v_L^1). *Au contraire, un nœud de* R, *nommé* v_R^1 *et illustré en vert, ne fait pas partie de l'image de* f_r. *Il sera donc ajouté lors de l'application de* r.

Un exemple d'application de r *à un graphe* G *est présenté dans la figure 3.8. La règle* r *est appliquée d'après le morphisme* m *(qui est d'ailleurs un homomorphisme) associant les nœuds* v_R^5 *à* v_G^6, v_L^4 *à* v_G^4, v_L^3 *à* v_G^3, v_L^2 *à* v_G^1 *et* v_L^1 *à* v_G^2.
Dans un premier temps, les images par m *de la partie de* L *n'étant pas dans le domaine de* f_r *sont supprimées. Cette dernière est constituée du nœud* v_L^5 *et de l'arc* (v_L^5, v_L^1) *dont les images sont respectivement* v_G^6 *et* (v_G^6, v_G^2). *La suppression du nœud noté* v_G^6 *entraîne l'apparition d'un arc suspendu,* (v_G^5, v_G^6), *qui est également supprimé.*
Finalement, une copie isomorphe de la partie de R *ne faisant pas partie de l'image de* f_r, *soit le nœud* v_R^1 *et l'arc* (v_R^2, v_R^1), *est ajouté. À travers l'opération d'expansion, les nœuds ayant une même image par* f_r *sont également fusionnés. En l'occurrence les nœuds notés* v_G^3 *et* v_G^4 *sont fusionnés pour donner le nœud* v_R^3. *L'arc* (v_G^3, v_G^4) *devient par conséquent* (v_R^3, v_R^3).

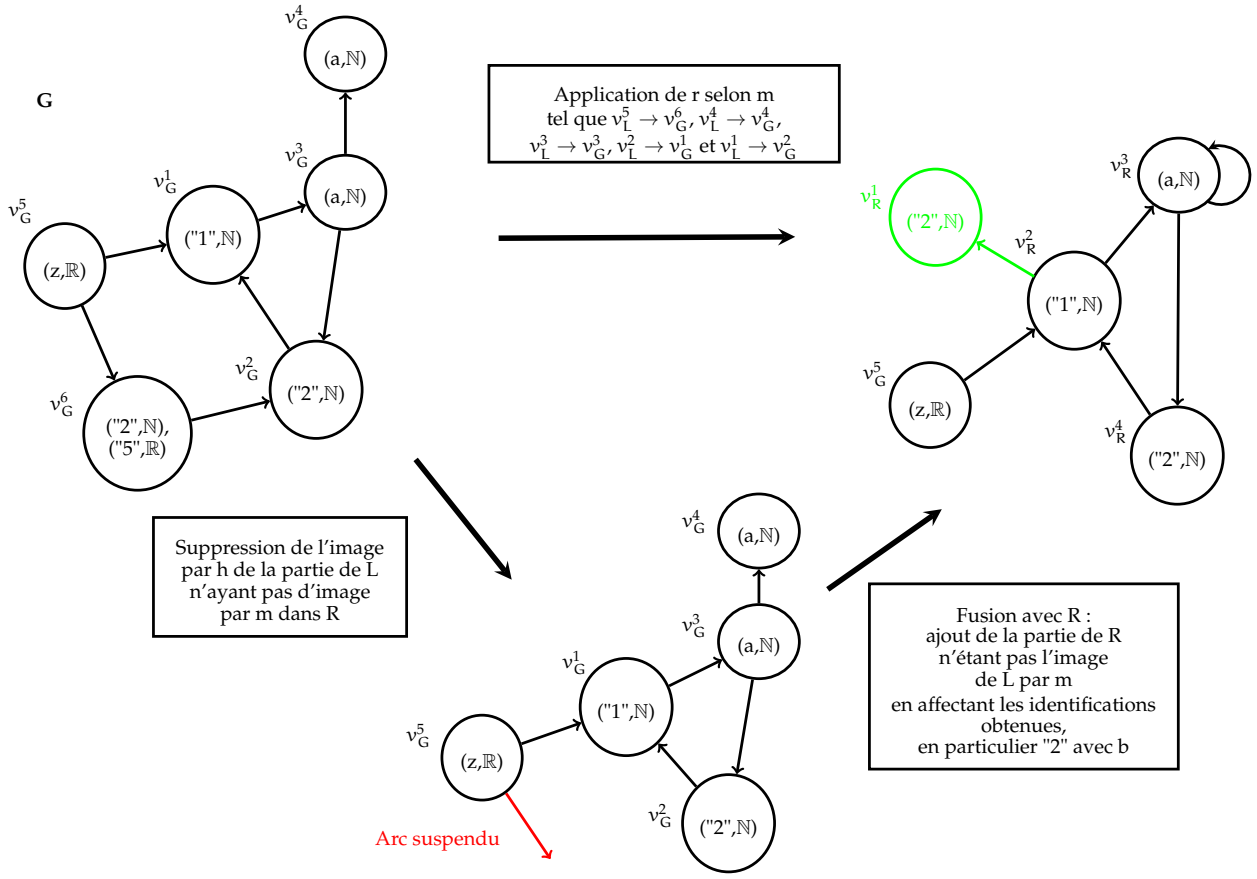


FIGURE 3.8 – Application d'une règle de réécriture de graphe

3.3 Analyse et gestion de requêtes pour un moteur de traitement d'évènements complexes en tant que service

Le traitement d'évènements complexes [11] (CEP) est un des exemples les plus éminents des technologies permettant le traitement des flux continus, rapides et massifs de données générés par capteurs et technologies mobiles. Il consiste à appliquer des règles, ou requêtes, à ces flux de données afin de les manipuler. Cette section exploite et illustre le formalisme présenté précédemment dans le cadre de la réalisation d'un moteur "CEPaaS" [12], une solution de CEP en tant que service dans des environnements multi-clouds. Ce contexte implique de multiples clients et intègre des problématiques liées à la qualité de service (QoS) et à la consommation énergétique du système. Nous concevons et décrivons ici le module d'analyse et de gestion de requêtes [13] (QAM).

L'objectif est de fournir à QAM une représentation haut niveau abstrayant opérateur et requête et n'exhibant que les propriétés nécessaires à leur gestion pour palier à la fragmentation des solutions techniques de CEP. Pour ce faire, nous examinons les étapes de vie d'une requête et identifions les caractéristiques pertinentes au contexte en élaborant une classification innovante des opérateurs de CEP. Cette conception modulaire et abstraite permet l'adoption de QAM par la majorité des systèmes CEP.

11. en anglais : *Complex Event Processing*
12. *Complex Event Processing As A Service*
13. en anglais : *Query Analyser and Manager*

Les travaux présentés ici ont été effectués en collaboration Wilson Akio Higashino et Miriam Capretz, dans le cadre d'une coopération avec la Western University, Ontario, Canada.

3.3.1 Contexte : Le traitement d'évènements complexes

3.3.1.1 L'approche CEP

Les capteurs et technologies mobiles au succès grandissant génèrent un flux continu, rapide et massif de données. Leur exploitation a motivé l'élaboration de solutions innovantes, dont le CEP est l'un des exemples les plus éminents.

Le CEP est défini comme "un ensemble d'outils et de techniques permettant d'analyser et de manipuler une suite complexe d'évènements corrélés intervenant dans les systèmes d'informations distribués modernes" [14] [105].

Les systèmes fondés sur l'approche CEP interprètent les données entrantes sous la forme d'un flux d'évènements. Ils lui appliquent des requêtes (ou règles) définies par leurs utilisateurs et composées de différents opérateurs interconnectés. Ces opérateurs et interconnections sont souvent représentés sous la forme de graphes d'exécution représentant ainsi une ou plusieurs règles. Ces dernières traitent continuellement le flux entrant et dérivent de ces évènements simples des évènements "complexes" sémantiquement enrichis. Ceux-ci peuvent alors être analysés afin d'établir un diagnostic rapide du système, de déclencher diverses actions ou de lancer des plans de réactions (e.g., l'adaptation dynamique de processus d'affaires [106]). Les systèmes de CEP peuvent être vus et utilisés pour la réalisation de modules d'observation avancés dans le cadre de la boucle autonomique MAPE-K. Ce modèle de fonctionnement a favorisé l'adoption du CEP dans de multiples situations exigeant des réactions promptes et autonomes, telles que la gestion de réseaux ou de maisons intelligentes.

Les systèmes CEP. Les bases du CEP ont été établies autour de systèmes et moteurs considérés comme des classiques du domaine, tels Aurora [107] et STREAM [108]. Leurs architectures centralisées les écartent toutefois du scénario de CEPaaS.
L'apparition et la prolifération des environnements et solutions de type cloud a poussé à leur utilisation en tant qu'environnement d'exécution dans des travaux plus récents, comme TimeStream [109] et StreamCloud [110]. Ces travaux se reposent néanmoins majoritairement sur des architectures agglomérées qui ne peuvent être transposées à un outil vastement dispersé [111]. Une réponse possible à cette problématique consiste à fédérer ces moteurs dans des environnements distribués, comme proposé par DiCEPE [15] [112]. À l'association de moteurs hétérogènes, nous préférons ici la conception d'une solution unifiée.

Les langages de définition de requêtes. Les requêtes de CEP sont en général définies dans des langages spécifiques et propriétaires, comme CQL [113]. En dépit d'efforts de standardisation [114], une multitude de langages de descriptions sont toujours utilisés à l'heure actuelle [111, 115].

Face à cette grande disparité et cette fragmentation des solutions techniques, une représentation haut niveau et abstraite s'impose. Par la suite, nous proposons donc une telle représentation, tout

14. "set of tools and techniques for analysing and controlling the complex series of interrelated events that drive modern distributed information systems"
15. *Distributed Complex Event Processing Engine*

en exhibant les propriétés nécessaires à la gestion des requêtes, que nous étudions en détail dans la section 3.3.3.

Gestion autonome. La plupart des systèmes de CEP n'intègrent pas de mécanismes de bout en bout pour la gestion de requêtes ; c'est à dire de la conception d'une requête par un utilisateur jusqu'à son exécution. Généralement, ils ne s'y intéressent que partiellement et se restreignent à certaines étapes du processus.

Kalyvianaki *et al*. [116], par exemple, décrivent un planificateur de requêtes optimisant l'exécution de multiples requêtes en mutualisant leurs opérateurs.
Abadi *et al*. [117] proposent quant à eux de conduire des actions de reconfigurations dynamiques lors de l'exécution d'un système Borealis [117], le successeur d'Aurora. L'optimisation *a priori*, lors de la phase de conception, n'a été que superficiellement traitée dans la littérature, en particulier dans le cas d'une requête isolée. En effet, la plupart des travaux existants ne s'intéressent à l'adaptation d'une requête qu'*a posteriori*, négligeant la possibilité qu'un client définisse une règle non-optimale. Il existe néanmoins quelques exemples de travaux et de techniques se plaçant dans le cadre de l'optimisation *a priori*, parmi lesquels la transformation d'une suite d'agrégateurs binaires en un agrégateur n-aire [118] et la permutation de filtre et d'agrégateurs [113].

3.3.1.2 CEPaas : Intérêt et challenges

Un système *CEPaaS* fourni des fonctionnalités de CEP à la demande en utilisant des technologies de service standardisées. Ce modèle de service, largement adopté aujourd'hui, apporte de nombreux avantages aux utilisateurs, tels que :

1. Pas d'investissement préalable dans des infrastructures matérielles et logicielles.
2. Faible coût de maintien grâce à peu ou pas de besoins de monitoring, de maintenance et de sauvegarde des infrastructures physiques
3. Mises à jour constantes, gratuites et sans interruption de service.

Fournir un tel service de CEP présente néanmoins de nombreux challenges, expliquant le manque d'offre similaire actuellement :

1. Faible latence et grande réactivité ; les évènements sont non seulement continuellement générés, mais tendent en plus à perdre très rapidement de leur intérêt. Dans le cas d'un moteur de CEPaaS, ce critère est entravé par l'absence de maîtrise vis à vis de l'emplacement des créateurs et consommateurs de ressources, ainsi que la multiplicité des clients.
2. Grande disponibilité, du fait des caractéristiques des cas d'utilisation du CEP.
3. Élasticité, la charge du système étant par essence imprévisible et hautement variable.

Face à ces contraintes, un environnement de déploiement multi-cloud semble être la réponse adéquate. En effet, une conception distribuée profite à l'extensibilité du système, la considération de clouds publiques améliorant également son accessibilité. Finalement, l'inhérente élasticité des environnements de cloud peut ainsi être exploitée via des mécanismes classiques d'allocation et désallocation de ressource en fonction de la charge du système.

3.3.2 Analyse et gestion de requêtes de bout en bout

Afin d'accompagner les requêtes depuis leur conception jusqu'à la fin de leur cycle de vie, leur gestion est découpée en différents stades introduits ici.

La figure 3.9 illustre informellement le cycle de vie typique d'une requête, et son passage au travers des différentes étapes de sa gestion détaillées ci-après. Tout d'abord, un utilisateur définit une requête qui est transformée en graphe et envoyée sous cette forme au module QAM.
Cette requête passe ensuite par trois étapes successives de gestion aboutissant à son déploiement. Son exécution est alors pilotée par QAM, réagissant de manière autonome à des changements de contexte.

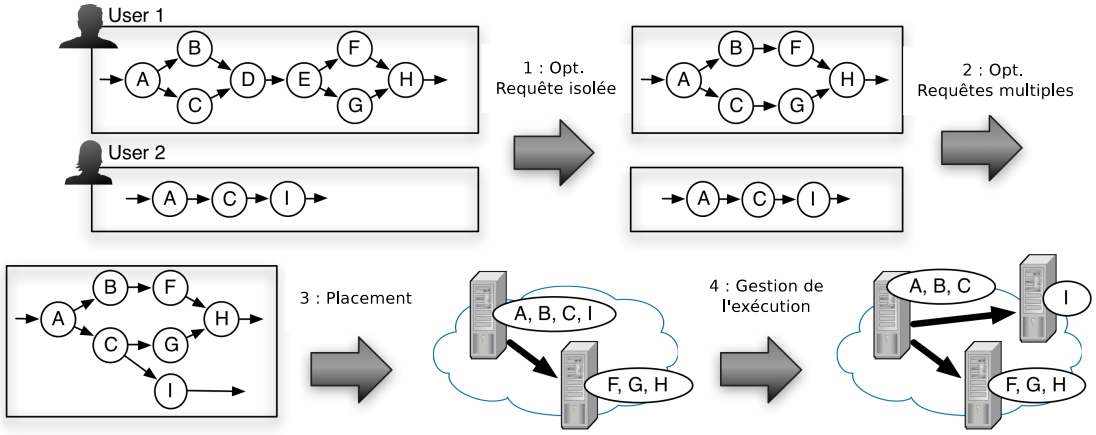


FIGURE 3.9 – Étapes pour l'analyse et la gestion de requêtes.

3.3.2.1 Optimisation de requêtes isolées

L'optimisation de requêtes isolées consiste à modifier un graphe d'exécution en conservant ses fonctionnalités mais en améliorant son efficacité (vis à vis de divers critères tels que la consommation CPU, réseau et énergétique, ou encore le temps de traitement).

Dans l'exemple de la figure 3.9, on suppose que D et E sont redondants, E annulant l'effet de D. Ce processus entraîne alors la suppression de ces deux opérateurs dans la requête de l'utilisateur 1. Cet exemple décrit l'application d'une des politiques détaillées dans la section 3.3.4.

Cette tâche a pour objectif premier de corriger la définition de requêtes non-optimales par un client. Elle s'effectue dans la phase de conception d'une requête (mais lors de l'exécution du moteur CE-PaaS) directement après la création et l'enregistrement d'une requête par un client. Elle ne peut se reposer sur la moindre information relative à l'état actuel du système et des ressources disponibles.

3.3.2.2 Optimisation de multiples requêtes

Différentes requêtes peuvent présenter des schémas opératoires communs, leurs graphes d'exécutions possédant alors des sous-graphes isomorphes. Si possible, ces schémas sont alors partagés en fusionnant les graphes d'exécution des requêtes identifiées. Cette tâche admet typiquement les

mêmes critères d'évaluation et d'optimisation que la précédente. Elle peut être menée à bien de manière indépendante à la réception d'une demande client ou être incluse dans le placement.

Dans l'illustration de la figure 3.9, les graphes d'exécutions des deux requêtes sont fusionnés selon le motif commun $A \rightarrow C$, les opérateurs A et C étant alors partagés.

3.3.2.3 Placement

Le placement désigne communément le mappage de chaque opérateur constituant une requête sur les ressources computationnelles (physiques ou virtuelles) disponibles. Dans le contexte d'un système de CEP multi-cloud, différents choix sont effectués à deux niveaux de granularité :

— Premièrement, un cloud est sélectionné pour l'exécution de chaque opérateur. Cette décision repose typiquement sur des métriques agrégées.
— Ensuite, le nombre et le type de serveur requis sont définis avant de répartir les opérateurs sur les différents processeurs disponibles.

Cette tâche est menée à bien lors du déploiement initial du système, lorsqu'une nouvelle requête est enregistrée et, en général, à chaque fois qu'un processus de reconfiguration le nécessite. Par exemple, un opérateur peut être dupliqué afin de paralléliser son exécution pour absorber une charge trop importante, requérant alors une décision de placement pour la nouvelle instance de l'opérateur.

3.3.2.4 Gestion de l'exécution

Cette étape réfère à la gestion autonome des requêtes durant leur exécution, au sens classique du terme. Système et requêtes sont reconfigurés en réponse à divers évènements et changements de contexte tels que des dépassement de seuil de QoS et/ou des pannes physiques ou logicielles.

Dans l'exemple de la figure 3.9, un nouveau serveur est provisionné, sur lequel l'opérateur I est migré.

Ces transformations sont effectuées dans la phase d'exécution et peuvent nécessiter des prises de décisions complexes.

3.3.3 Classification des opérateurs de CEP

Afin de faire face à la fragmentation des solutions techniques pour la définition d'opérateurs CEP, il nous faut les abstraire tout en représentant leurs propriétés pertinentes à leur manipulation. Dans le cadre d'une modélisation basée sur des graphes, ces propriétés seront reflétées par des attributs.

Par conséquent, nous décrivons dans cette section une classification innovante des opérateurs de CEP se focalisant sur leurs propriétés de reconfiguration. Cette classification identifie les propriétés d'un opérateur pertinentes pour sa gestion, permettant par la suite d'abstraire les objets manipulés en exhibant uniquement ces informations. Ainsi, il est possible de spécifier des transformations et politiques de reconfiguration de haut niveau, génériques et extensibles, théoriquement applicables à tout opérateur convenablement classifié.

L'abstraction par exhibition de propriétés pertinentes seulement est au cœur de la conception d'un modèle survolant les solutions techniques disparates. En l'occurrence, les caractéristiques nécessaires à la gestion d'opérateurs CEP sont leur modalités de partage et de duplication, ainsi que leur comportement vis à vis des flux traités.

3.3.3.1 Modalités de partage

Ce critère détaille la capacité des opérateurs à être partagés par plusieurs requêtes. Cette caractéristique est au cœur de l'étape de gestion dite d'optimisation de multiples requêtes, dans laquelle des schémas opératoires communs sont recherchés. Si deux sous graphes de graphes d'exécution sont isomorphes, ils peuvent être fusionnés (via une opération d'expansion) si les opérateurs les constituant sont partageables.

Ce critère dépend principalement de l'implémentation d'un opérateur. Il est bien entendu possible qu'un opérateur ne puisse pas du tout être partagé. Dans le cas contraire, ses propriétés vis à vis de deux sous-catégories doivent être identifiées :

- *Données :*
  - *Fonctionnel* : l'opérateur peut être partagé entre toute requête l'utilisant pour effectuer le même traitement (même paramètres), que les données traitées soient les mêmes ou non.
  - *Source* : l'opérateur ne peut être partagé qu'entre requête l'utilisant pour exécuter le même traitement sur les mêmes données (même source ou flux d'entrée).
- *Utilisateur :*
  - *Multi-utilisateurs* : l'opérateur peut être partagé par des requêtes de clients différents.
  - *Utilisateur unique* : l'opérateur ne peut être partagé qu'entre des requêtes provenant du même utilisateur.

3.3.3.2 Modalité de duplication

Ce critère décrit la capacité d'un opérateur à être cloné pour paralléliser son exécution et la distribuer sur plus de serveurs. Lorsqu'un opérateur peut être dupliqué, deux aspects doivent être considérés : comment dispatcher le flux entrant entre les n instances de l'opérateur et comment fusionner à nouveau chaque sous-flux après traitement.

Un opérateur ayant un flux d'entrée et n flux de sortie, permettant de séparer et dispatcher les flux est appelé diviseur. Quatre type de diviseurs sont considérés :

- *Aléatoire* : les entrées sont dispatchées de manière aléatoire ou ne dépendant pas de leurs propriétés (e.g., en fonction de l'état des différentes cibles potentielles).
- *Selon attribut* : les instances destinataires sont sélectionnées de manière déterministe en fonction de la valeur de certains de leurs attributs.
- *Selon requête* : les entrées sont séparées en fonction de leurs requêtes d'appartenance (e.g., après un traitement partagé).
- *Utilisateur* : l'utilisateur fournit une fonction de transmission sans en détailler les propriétés.

Notons qu'un séparateur aléatoire est plus contraignant que les autres : si un opérateur admet un séparateur aléatoire, il peut également admettre des séparateurs selon attribut et/ou requête.

Similairement, les flux peuvent être à nouveau agrégés après traitement de trois manières différentes :

— *Union* : les flux sont transmis sur le même canal de sortie dans leur ordre d'occurrence.
— *Union triée* : les flux sont transmis sur le même canal de sortie, dans un ordre spécifié par une fonction de tri sur certains de leurs attributs.
— *Utilisateur* : l'utilisateur fournit une fonction de transmission sans en détailler les propriétés.

3.3.3.3 Comportement

Cette sous-catégorie s'intéresse aux caractéristiques fonctionnelles d'un opérateur. Ces dernières sont limitées à deux facteurs influençant tout particulièrement les stratégies d'optimisation du système :

— *Complexité* : la complexité algorithmique de l'opérateur en fonction de la taille des flux entrants (traitement à complexité linéaire, polynomiale, logarithmique...).
— *Sélectivité* : réfère au ratio de la taille des sorties sur celles des entrées. Un filtre aura typiquement une sélectivité inférieure à 1, ce dernier transmettant partiellement son flux d'entrées.

3.3.3.4 Exemples d'opérateurs classifiés

Nous proposons ici d'introduire quelques opérateurs communs et de les classer à titre d'exemple. Les opérateurs considérés sont décrits dans le tableau 3.2, d'après un modèle de flux classique [113] consistant en un multi-ensemble d'éléments $\sigma = \langle s, t \rangle$, où s est une collection d'objets ayant une forme Γ particulière, et t est son horodatage.

Nom	Paramètres	Description
Filtre	p	Transmet les σ satisfaisant un prédicat p
Transformation	$f(\Gamma_1, \Gamma_2)$	Change la forme des s de Γ_1 à Γ_2 en usant de la fonction f
Tri	W, A	Tri les σ en fonction de leurs attributs A
Union		Retransmet tous les σ entrants vers un même canal
Fusion	W_1, W_2, A	Union de deux flux ignorant les occurrences multiples de σ ayant les mêmes attributs A sur les fenêtres W_1 et W_2
N-Fusion	$W_1, \ldots, W_n, A$	Fusion n-aire
Motif	M	Transmet les σ satisfaisant le motif évènementiel M
Utilisateur	?	Opérateur défini par l'utilisateur

TABLE 3.2 – Exemples d'opérateurs de CEP

Les opérateurs Tri, Fusion et N-Fusion appliquent un traitement à leurs entrées en les regroupant par fenêtres. Le paramètre W décrit les caractéristiques de ce partitionnement (taille glissante de la fenêtre...). Ces opérateurs reçoivent également un paramètre A spécifiant une liste d'attributs sur lesquels les opérations doivent se baser. Par exemple, l'opérateur de tri range les σ d'après une fonction d'ordre sur A.

Le tableau 3.3 caractérise ces opérateurs d'après la classification élaborée. Ces exemples permettent plusieurs constats :

Nom	**Partage**		**Duplication**			**Comportement**	
	Données	Utilisateur	Dupl.	Sep.	Agr.	Complexité	Sel.
Filtre	fonctionnel	multiple	vrai	aléatoire	union	linéaire	$\leqslant 1$
Transfo.	fonctionnel	multiple	vrai	aléatoire	union	linéaire	1
Tri	source	multiple	vrai	aléatoire	union triée	linéaire	1
Union	sources	multiple	faux			linéaire	1
Fusion	sources	multiple	vrai	attributs	union	linéaire	$\leqslant 1$
N-Fusion	sources	multiple	faux			sur-linéaire	$\leqslant 1$
Motif	source	multiple	?	utilisateur	utilisateur	?	?
Utilisateur	?	Non	?	utilisateur	utilisateur	?	?

TABLE 3.3 – Classification des opérateurs introduits

Partage. En règle générale, tous les opérateurs sans état (`Filtre`, `Transformation`...) peuvent être partagés entre requêtes pour traiter des flux de sources distinctes, contrairement aux opérateurs à états. En effet, il faudrait alors qu'un opérateur à état maintienne et gère plusieurs états en parallèle, compliquant son implémentation et sa gestion.
Les opérateurs basiques peuvent généralement être partagés entre multiples utilisateurs, tandis qu'un opérateur propriétaire ne le sera pas.

Comportement. La plupart des opérateurs de CEP ont une complexité linéaire au moins. Une complexité moindre est rare ; les opérateurs considèrent au moins une fois chaque élément qu'ils reçoivent.
Il est possible de comparer la sélectivité d'un opérateur à 1, mais souvent difficile d'obtenir un nombre exact. À la réception de n et m évènements, un opérateur de fusion, par exemple, peut en transmettre entre 1 et $n + m$.

3.3.4 Transformations et politiques d'auto-optimisation

Cette section fournit un échantillon des transformations et politiques d'auto-optimisation élaborée pour QAM. Ces exemples sont pris dans le cadre de l'optimisation d'une requête isolée à la fin de sa conception (mais dans la phase d'exécution du moteur CEPaaS), juste après sa définition par un utilisateur. De ce fait, très peu d'informations contextuelles (état du système, débit des flux..) sont accessibles. Cette phase d'optimisation et les politiques proposées ont pour objectif premier la correction de non-optimalités dans la définition d'une requête. Celles-ci peuvent par exemple apparaître du fait d'utilisateurs néophytes du CEP et des problématiques d'optimisation en général, lors de l'utilisation de principes de conception haut niveau, ou lors de l'application d'autres politiques de gestion autonome.

En particulier, nous introduisons tout d'abord le processus de duplication d'opérateurs avant de spécifier deux politiques d'optimisations : la suppression d'un motif séparation/union inutile et le traitement parallèle de sous-flux.

3.3.4.1 Formalisation de requêtes

Les requêtes définies par les utilisateurs sont transformées en graphes (acycliques) attribués et dirigés. Dans le graphe résultant, les nœuds représentent des opérateurs et les arcs des flux de données/évènements. Les attributs de ces éléments reflètent leurs propriétés pertinentes dont : le nombre et les identifiants des sources, requêtes et utilisateurs d'appartenance pour les arcs ; pour les opérateurs, leur caractérisation au sein de la classification proposée. Par soucis de lisibilité, les simplifications suivantes sont adoptées, celles-ci n'introduisant ni imprécision ni ambiguïté.

Premièrement, seuls les attributs intervenant dans les transformations décrites sont représentés, c'est à dire :

1. $\mathtt{Id}$, un identifiant unique.
2. $\mathtt{Nom}$ (chaîne de caractères), le nom de l'opérateur.
3. $\mathtt{Duplicable} \in$ {"Vrai", "Faux"}, un booléen reflétant le caractère duplicable de l'opérateur.
4. *Séparateur* et 5. *Agrégateur*, les noms des opérateurs (de type séparateur et agrégateur, respectivement) à déployer lors d'une duplication.

Les agrégateurs et séparateurs ne possèdent qu'un seul attribut : leur nom. De plus, les domaines de définition des attributs seront implicités.

Finalement, les attributs des arcs ne sont pas illustrés. Dans le cas de l'optimisation d'une requête isolée, les flux apparaissant proviennent nécessairement d'une même requête, et donc d'un même client.

3.3.4.2 Duplication d'opérateurs

La duplication d'opérateurs est une procédure d'adaptation classique basique permettant de partager la charge que représente l'exécution d'un opérateur sur plusieurs serveurs. Elle consiste à cloner un opérateur, diviser son flux d'entrée entre chacun des clones puis de les fusionner après traitement.
Ce processus peut être défini par deux *transformations basiques du systèmes*, c'est à dire deux actions accomplissant un objectif spécifique de bas niveau n'améliorant pas nécessairement le système.

Duplication initiale. Il s'agit de remplacer une unique instance d'un opérateur par deux clones, le séparateur et l'agrégateur appropriés étant alors déployés. Les extrémités des flux d'entrée et de sortie de l'opérateur initial doivent également être modifiées, le séparateur devenant la destination des entrées et l'agrégateur la source des sorties.

Ces modifications ne dépendent pas des attributs de l'autre extrémité ni de ceux de l'arc. Il serait alors fastidieux et peu extensible de définir une instruction de connexion de type edNCE [65] pour chaque combinaison d'attributs. Par conséquent, cette duplication initiale s'effectue en trois temps.

Dans un premier temps, il s'agit de déployer le motif objectif. L'opérateur initial est conservé pour permettre la redirection des flux dans la suite. La figure 3.10 illustre les parties droite et gauche de la règle de réécriture de graphe $\mathrm{r}^{\mathtt{init}}_{\mathtt{dupli}}(\mathtt{id})$ spécifiant cette première étape. Le paramètre $\mathtt{id}$ précise (l'identifiant de) l'opérateur à dupliquer. Le morphisme associé est caractérisé par la fonction associant le nœud v_1 de sa partie droite au nœud noté v_1 de sa partie gauche. Dans la suite, les morphismes caractérisant chaque règle de réécriture sont implicites et associent les nœuds de même

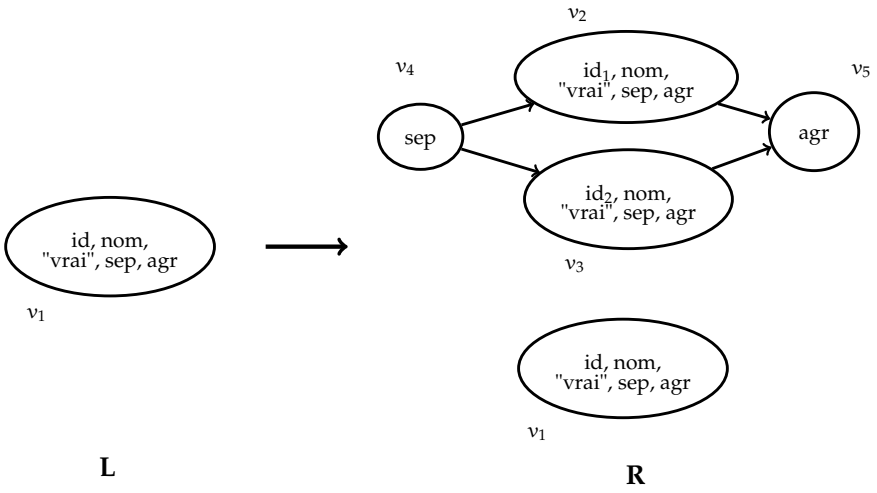
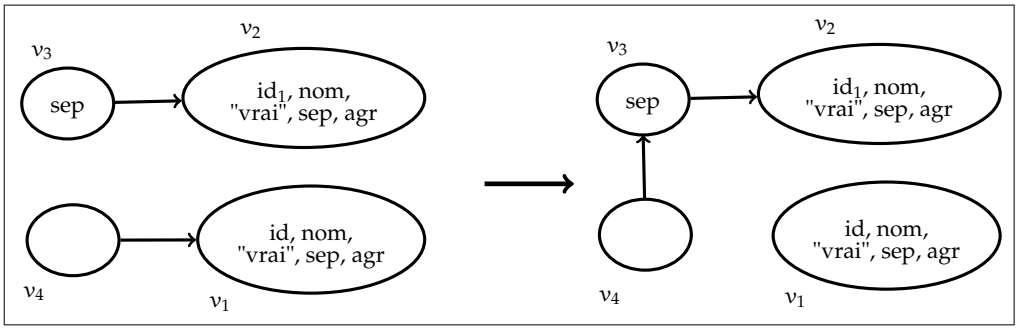


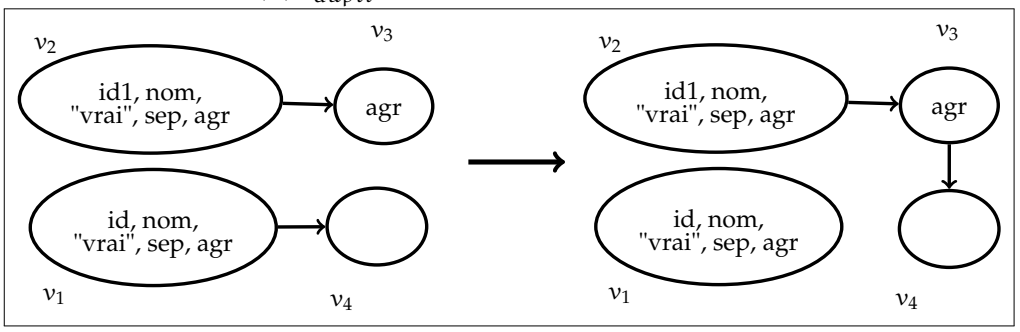
FIGURE 3.10 – $r^{init}_{dupli}(id)$, duplication initiale

nom. Notons que cette transformation est applicable à tout opérateur pouvant être dupliqué (son troisième attribut étant fixé à "$Vrai$").

Ensuite, chaque flux d'entrée est redirigé vers le séparateur en répétant la règle $r^{E}_{dupli}(id, id_1)$ de la figure 3.11a. L'agrégateur devient après la source de chacune des sorties par applications répétées de $r^{S}_{dupli}(id, id_1)$ définie dans la figure 3.11b.



(A) $r^{E}_{dupli}(id, id_1)$, redirection d'une entrée



(B) $r^{S}_{dupli}(id, id_1)$, modification de la source d'une sortie

FIGURE 3.11 – Modifications des flux

Finalement, l'opérateur initial est supprimé à l'aide de la règle $r^{fin}_{dupli}(id)$ représentée dans la figure 3.12.

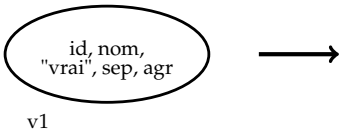


FIGURE 3.12 – $r^{fin}_{dupli}(id)$, fin de duplication

Ce processus est décrit dans l'algorithme 3.1 ci-dessous.

Fonction duplication
Données : G, le graphe représentant la requête considérée.
id, l'identifiant de l'opérateur à dupliquer.
début
  appliquer $r^{init}_{dupli}(id)$ à G
  $idClone \leftarrow id_1$
  pour chaque *homomorphisme* $h : L_{r^{E}_{dupli}(id,idClone)} \rightarrow G$ **faire**
    appliquer $r^{E}_{dupli}(id, idClone)$ à G selon h
  fin
  pour chaque *homomorphisme* $h : L_{r^{S}_{dupli}(id,idClone)} \rightarrow G$ **faire**
    appliquer $r^{S}_{dupli}(id, idClone)$ à G selon h
  fin
  appliquer $r^{fin}_{dupli}(id)$ à G
fin

ALGORITHME 3.1 - Duplication d'un opérateur

Addition d'une instance. Ce procédé consiste à incrémenter le nombre d'instances d'un opérateur dupliqué.

Cette transformation est caractérisée par la règle de réécriture $r^{add}_{dupli}(id)$, définie dans la figure 3.13. De même que dans la règle précédente, le paramètre id identifie l'opérateur ciblé.

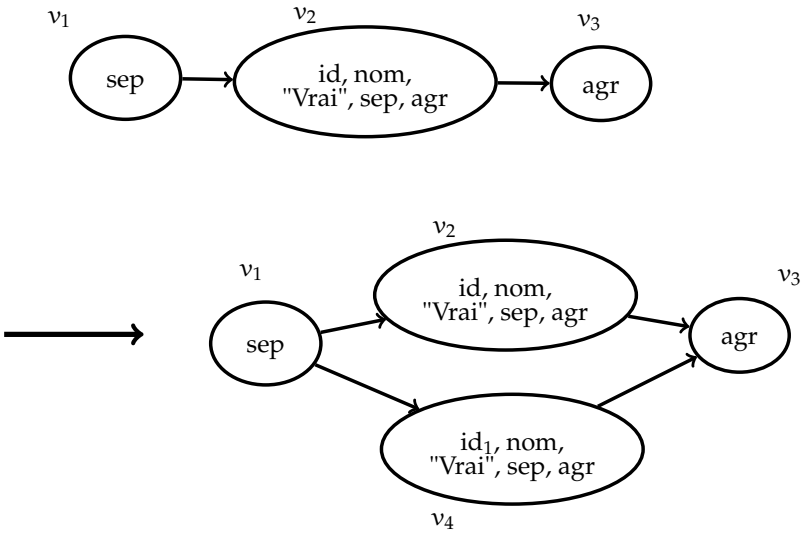


FIGURE 3.13 – $r^{add}_{dupli}(id)$, copie d'un opérateur dupliqué

3.3.4.3 Suppression d'un motif agrégateur/séparateur inutile

Cette politique d'optimisation décrit la suppression d'un motif composé d'un agrégateur suivi d'un séparateur, noté motif A/S, lorsque ce motif est inutile. Plus particulièrement, n flux d'entrée sont fusionnés puis directement séparés en m flux de sortie. Ce motif est superflu lorsque les deux conditions suivantes sont remplies :

1. L'agrégateur ne modifie pas les flux le traversant. D'après la classification, seul un opérateur `Union` satisfait cette condition.
2. Le séparateur ne doit pas discriminer les flux de manière à introduire des informations supplémentaires dans ses sorties. Conséquemment, un séparateur selon *requête* (respectivement, selon `utilisateur`) est acceptable si et seulement si toutes ses entrées appartiennent à la même requête (resp. au même utilisateur). Dans le cadre de l'optimisation de requête isolée, chaque flux appartient nécessairement à la même requête et au même utilisateur. Ainsi, les séparateurs selon *requête* ou `utilisateur` satisfont cette condition. Les séparateurs *aléatoire* sont quant à eux toujours acceptables.

Pour ne pas introduire de problématique d'équilibrage de charge, nous supposons dans la suite que $n = m$.

Fonction supprAS
Données : G, le graphe représentant la requête à optimiser.
AcceptSep, l'ensemble des séparateurs acceptables.
`/* Dans le cas de l'optimisation d'une requête isolée, AcceptSep = {"Aléatoire", "Selon utilisateur", "Selon requête"} */`
début
  `/* Étape 1 : contournement de tout motif A/S inutile */`
  pour chaque $sep \in AcceptSep$ **faire**
    pour chaque *homomorphisme* $h : L_{r^1_{A/S}(sep)} \to G$ **faire**
      appliquer $r^1_{A/S}(sep)$ à G selon h
    fin
  fin
  `/* Étape 2 : suppression des motifs A/S isolés */`
  pour chaque *homomorphisme* $h : L_{r^2_{A/S}} \to G$ **faire**
    si $r^2_{A/S}$ *est applicable à* G *selon* h **alors**
      appliquer $r^2_{A/S}$ à G selon h
    fin
  fin
fin

ALGORITHME 3.2 - Suppression des motifs A/S inutiles

L'algorithme 3.2 détaille la procédure supprimant tous les motifs A/S inutiles. Elle se déroule en deux étapes. Premièrement, chaque motif A/S inutile est contourné par des applications successives de la règle $r^1_{A/S}(nom)$ définie dans la figure 3.14.

Ensuite, chaque motif isolé est supprimé en appliquant la règle $r^2_{A/S}$ illustrée dans la figure 3.15. Cette dernière comporte deux conditions d'application négative représentées en pointillés et tirets

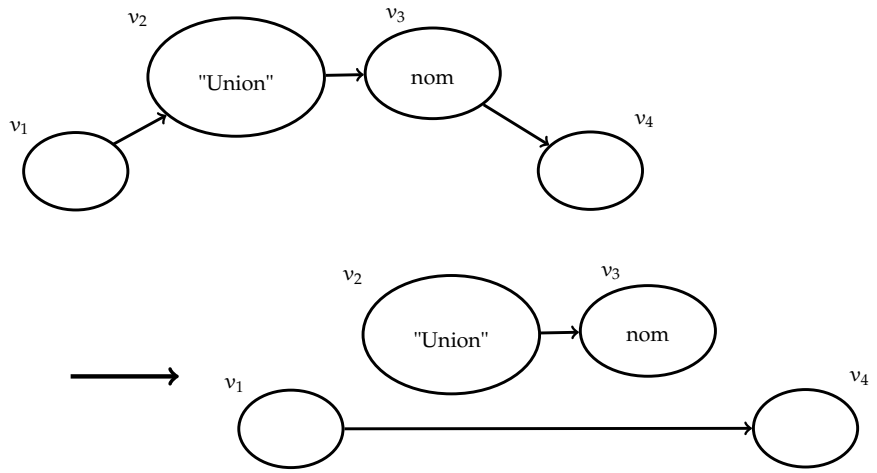


FIGURE 3.14 – $r^1_{A/S}(nom)$: contournement d'un motif A/S inutile

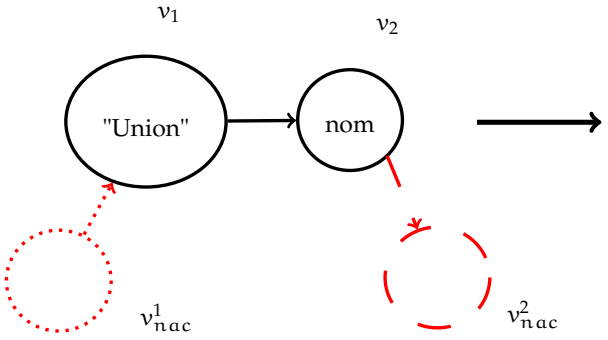


FIGURE 3.15 – $r^2_{A/S}$: suppression d'un motif A/S inutile

rouges. Elles spécifient que l'opérateur d'union n'a aucun flux d'entrée et que le séparateur (ou agrégateur) suivant n'a pas de sortie.

L'application de cette politique est informellement illustrée dans la figure 3.9. Supposons que, dans la requête de l'utilisateur 1, les opérateurs D et E représentent une union et un séparateur *aléatoire*, respectivement. Les sorties de B et C sont directement transmises à F et G, respectivement, et le motif $D \rightarrow E$ est supprimé.

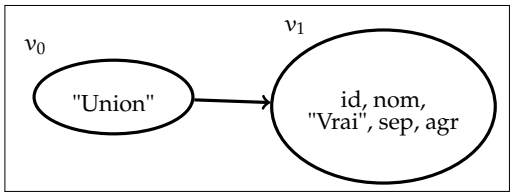
3.3.4.4 Traitement de sous-flux

Cette politique d'optimisation transpose au CEP une stratégie classique consistant à diviser un problème en plusieurs sous-problèmes pour traitement. Dans le contexte actuel, nous considérons un opérateur O traitant un flux résultant d'une agrégation de n flux. Cette situation est illustrée dans la figure 3.16a où O est représenté par v_1. Dans l'idéal, l'opération devrait être parallélisée et conduite sur chaque flux pré-agrégation : les places d'O et de l'agrégateur devraient être échangées afin d'atteindre une situation telle qu'illustrée dans la figure 3.16b.

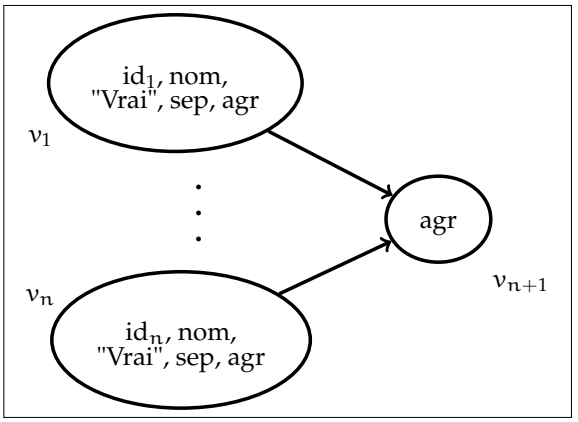
Dans les faits, cette transformation est équivalente à n duplications d'O (voir figure 3.16c), suivie de la suppression d'un motif A/S inutile. Elle est donc applicable si :

1. O peut être dupliqué.

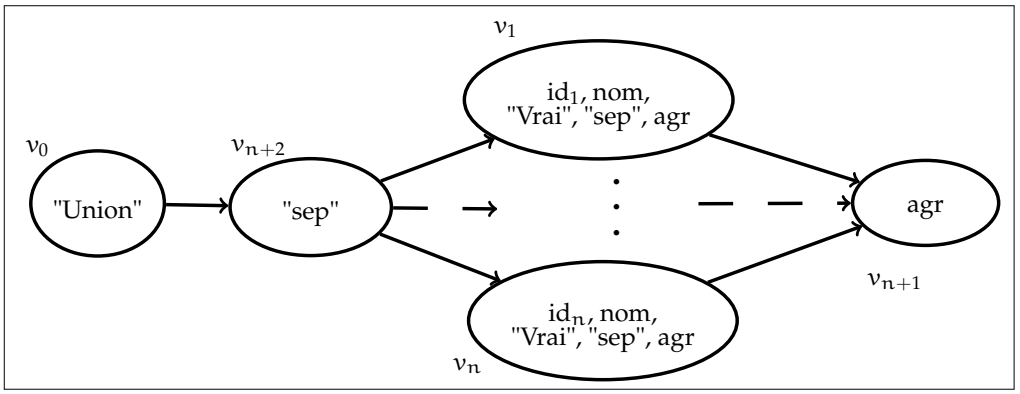
2. le motif A/S introduit lors de la duplication d'O est inutile. Nous avons vu précédemment qu'il faut alors :
   (a) que l'agrégateur initial soit une union.
   (b) que le séparateur introduit lors de la duplication d'O soit *"Aléatoire"*, ou, dans le cas de l'optimisation d'un requête isolée, *"Selon utilisateur"* ou *"Selon requête"*



(A) Situation initiale



(B) Situation finale souhaitée



(C) Situation intermédiaire

FIGURE 3.16 – Scénario de traitement de sous-flux

Cette politique peut ainsi être formalisée par l'algorithme 3.3, où $G_{init}(sep)$ est le graphe de la figure 3.16a, son paramètre fixant la valeur du nom du séparateur noté sep dans les figures.

Données : G, le graphe représentant la requête à optimiser.
$AcceptSep$, l'ensemble des séparateurs acceptables.
début
  /* Étape 1 : détection et duplication */
  pour chaque $sep \in AcceptSep$ **faire**
    pour chaque *homomorphisme* $h : G_{init}(sep) \rightarrow G$ **faire**
      /* Récupération de l'id du nœud à dupliquer */
      idOp $\leftarrow$ id
      /* Récupération du demi degré entrant de l'union initiale (i.e., du nombre d'arcs entrants dans le noeud représentant l'union initiale) */
      $n \leftarrow |\{e \in E_G | \exists v \in V_G, e = (v, h(v_0))\}|$
      duplication(G, idOp) **pour** $i \leftarrow 2$ **à** n **faire**
        appliquer r^{add}_{dupli}(idOp) à G
      fin
    fin
  fin
  /* Étape 2 : suppression des motifs inutiles ainsi créés */
  supprAS(G, $AcceptSep$)
fin

ALGORITHME 3.3 - Transformation pour traitement de sous-flux

3.3.5 Discussion

La première politique de gestion introduite décrit l'identification et la suppression d'opérateurs inutiles. Elle améliore donc nécessairement le système, tant d'un point de vue performance par la suppression d'opérations que de son coût par la suppression d'opérateurs.

La seconde politique transpose au CEP une stratégie d'optimisation classique, "diviser pour mieux régner". Plus précisément, il s'agit de diviser un problème en sous-problèmes, afin d'effectuer plusieurs traitements de tailles réduites. Cette stratégie comporte deux intérêts majeurs. Tout d'abord, elle permet de paralléliser le dit traitement. Deuxièmement, elle implique un fort gain d'efficacité dans le cas d'un traitement dont la complexité est sur-linéaire. Rappelons à ce sujet qu'il est extrêmement rare pour un opérateur de CEP de posséder une complexité sous-linéaire (i.e. logarithmique ou constante). Cette politique améliore ainsi la performance du système, bien que le déploiement d'opérateurs supplémentaires puissent entraîner un surcoût.

Ces deux politiques sont donc pertinentes dans le cadre de l'amélioration d'au moins un des aspects du système, et s'inscrivent ainsi dans le cadre de son auto-optimisation. La seule question en suspens est l'ampleur de l'amélioration induite. Une évaluation (circonstancielle car dépendante de la configuration, de la nature exacte des opérateurs impliqués...) n'est pas fournie ici ; elle est en court de réalisation dans le cadre de l'implémentation de QAM, la collaboration avec la Western University se poursuivant.

Il est également intéressant de souligner l'exemple fournit vis à vis de l'introduction, lors d'une transformation de haut niveau, d'un motif trivialement inefficace. En effet, nous avons vu comment

une duplication peut entraîner l'apparition d'un motif A/S inutile dans le cadre de la politique permettant le traitement de sous-problèmes (sa suppression étant immédiatement prévue dans cette politique).

3.4 Conclusion du chapitre "auto-optimisation"

Dans ce chapitre, **nous élaborons les bases du méta modèle utilisé** tout au long de ce manuscrit. Ce modèle est ensuite illustré et appliqué dans le cadre de la conception d'un moteur de CEPaaS incluant des problématiques d'auto-optimisation considérant les performances du système de même que sa consommation énergétique.

L'approche adoptée est basée sur les graphes, un formalisme universel couramment utilisé pour la spécification d'un vaste panel de systèmes. Dans ce cadre, les graphes représentent des états du système étudié, aussi appelés configurations. Les évolutions et transformations du système peuvent alors être caractérisées par des transformations de graphes, et en particulier des règles de réécriture de graphe. Celles-ci se basent sur l'existence de relations entre graphes, liées par exemple à l'existence de motifs communs.

Nous nous inscrivons dans la continuité de tentatives de traduction d'approches catégorielles en une vue ensembliste et opérationnelle, concrète et proche de l'implémentation. Nous adoptons ainsi les bases du modèle proposé dans [6] , vis à vis desquelles nous apportons un certain nombre de contributions récapitulées dans le tableau 3.4.

Concept	**Contribution**
Ensemble d'identifications	formalisation
Unification d'attributs	formalisation utilisant le concept précédent
Ensemble d'identifications consistant	formalisation
Unification d'éléments	1) formalisation utilisant les concepts précédents 2) extension permettant aux deux éléments de posséder des attributs variables
Affectation	formalisation utilisant les concepts précédents
Morphisme	1) formalisation 2) extension pour intégration d'attributs variables dans les deux graphes 3) extension pour prise en compte d'attribution partielle 4) extension à des relations non injectives 5) introduction de la notion de cohérence pour conserver la propriété de transitivité des morphismes
Compatibilité	introduction de la relation
Restriction et Expansion	introduction de l'opération
Règle de réécriture	formalisation utilisant les concepts précédents

TABLE 3.4 – Récapitulatif des contributions de formalisation

En particulier, le formalisme résultant s'étend à présent à l'**attribution partielle**, aux **relations non injectives** et aux **attributs variables**. Son aspect visuel facilite une compréhension intuitive de situations complexes. En effet, une proportion significative des termes introduits sont "internes". Bien que nécessaire à la formalisation du modèle, un architecte logiciel peut les ignorer. Le tableau 3.5 résume les principaux éléments visibles par un architecte/utilisateur et leur correspondance dans un système réel.

Concept	**Formalisation**
Composant/Constituant élémentaire	Nœud attribué
Connexion/Relation	Arc attribué
Propriété	Attribut
Configuration/État du système à un instant donné	Graphe partiellement attribué
Transformation/Évolution du système	Règle de réécriture de graphe

TABLE 3.5 – Récapitulatif de la formalisation des différents concepts liés aux systèmes dynamiques

Après l'avoir introduit, nous appliquons et illustrons ce modèle dans le cadre de la réalisation d'un moteur "CEPaaS", une solution innovante de traitement d'évènements complexes (CEP) en tant que service dans des environnements multi-clouds. Plus particulièrement, nous concevons et décrivons ici le module d'analyse et de gestion de requêtes (QAM), le gestionnaire autonome du moteur dont une première description a fait l'objet d'une publication [119]. Grâce à une représentation haut niveau abstrayant opérateurs et requêtes, QAM s'affranchit en particulier de leurs langages de définition. Sa conception modulaire permet son adoption par la majorité des systèmes CEP.

Dans un premier temps, **nous identifions les quatre différentes étapes du cycle de vie et de la gestion d'une requête**, à savoir : l'optimisation de requêtes isolées juste après leur définition ; l'optimisation de multiples requêtes par le partage de motifs opérationnels commun ; le placement des opérateurs sur les ressources physiques disponibles ; la gestion de l'exécution. Chaque étape correspond à un contexte, des problématiques et des méthodes de gestion différentes.

L'abstraction et l'exhibition d'uniquement des propriétés pertinentes est au cœur de la conception d'un modèle survolant les solutions techniques disparates. Pour les identifier, nous proposons une **classification novatrice des opérateurs de CEP** se focalisant sur leurs propriétés de reconfiguration. Cette classification **permet** ainsi **la spécification de transformations et politiques extensibles, indépendantes des langages de description et théoriquement applicables à tout opérateur convenablement classifié**.

Finalement, nous détaillons **trois exemples de transformations et politiques d'auto-optimisation élaborée pour QAM** dans le cadre de l'optimisation d'une requête isolée : la duplication d'un opérateur ; l'identification et la suppression d'opérations inutiles ; la parallélisation de traitement.

L'implémentation de l'intégralité du gestionnaire pour répondre à l'ensemble des phases identifiés est en cours dans le cadre d'une collaboration avec la Western University, Ontario, Canada. Nous visons en particulier une évaluation expérimentale du temps nécessaire à la conduite d'une politique de reconfiguration ainsi que des gains et compromis qu'elle induit en terme de qualité de service et de consommation d'énergie.

Chapitre 4

Préservation d'un style architectural par construction

Application à l'auto-protection d'architectures dynamiques

4.1 Introduction

La propriété d'auto-protection [1] permet à un système de conserver son intégrité en se protégeant à différents niveaux [120].
Le premier, qualifié d'auto-protection externe, consiste à se prémunir contre tout évènement exogène menaçant la survie du système, comme une attaque malicieuse. Un grand nombre de méthodes et techniques de sécurité vont en ce sens : détection et lutte contre les intrusions et attaques, protection et anonymisation de données...
Au contraire, le second type d'auto-protection, dit interne, est souvent passé sous silence [120]. Proche de la sûreté de fonctionnement, il consiste à protéger le système de lui-même, ou, plus précisément, des évolutions engagées dans le cadre de sa gestion autonome : il s'agit d'assurer que ces dernières maintiennent le système dans un état cohérent. Cette considération est sous-jacente à chacune des propriétés self-*.
Ainsi, l'auto protection interne doit être mise en œuvre par toute politique de gestion autonome, quelle que soit la propriété qu'elle implémente. Par conséquent, elle constitue l'objet principal des contributions exposées dans ce chapitre.

Nous élaborons dans la suite une méthodologie permettant la construction et la caractérisation générative de règles de transformations *correctes par construction* dont l'application préserve nécessairement la correction du système. L'auto-protection interne d'un système peut ainsi être efficacement assurée en utilisant, dans les politiques autonomes de reconfiguration, uniquement des transformations "correctes par construction".

Avant tout, il convient de fixer la notion de correction. Du fait de leur nature, la description de systèmes dynamiques ne peut se limiter à une seule configuration ; elle doit couvrir le champ des états acceptables. Ce champ peut être caractérisé par un style architectural, qui qualifie ce qui est correct et ce qui ne l'est pas. La section 4.2 introduit les grammaires de graphes, le formalisme utilisé

1. en anglais : *self-protection*

pour la spécification de styles architecturaux, ainsi que les divers types de corrections considérés ici.

Les grammaires de graphes offrent une définition générative de l'ensemble des configurations correctes sous la forme d'un système de réécriture intégrant des règles de réécriture. Ainsi, la caractérisation d'un style architectural fournit également un ensemble initial de transformations correctes. Il s'agit par la suite de générer, à partir de cet ensemble, de nouvelles règles correctes. Pour ce faire, nous introduisons dans la section 4.3 des opérations sur les règles de réécriture et montrons qu'elles préservent leur correction sous certaines conditions que nous étudions. Conformément au point de vue ensembliste et opérationnel adopté précédemment, nous proposons des méthodes concrètes pour l'implémentation et l'exploitation de ces opérations.

4.2 Systèmes dynamiques et correction

4.2.1 Styles architecturaux et correction

4.2.1.1 Caractérisation des styles architecturaux : les grammaires de graphes

Les grammaires de graphes, introduites dans la section 2.3.2, ont pour origine les grammaires génératives de Chomsky. De telles grammaires sont en général spécifiées par un axiome, des termes terminaux et non-terminaux et un ensemble de règles de réécriture (de chaînes de caractères) appelées productions de la grammaire.

Exemple 4.1. *Considérons une grammaire simple comportant un terme non terminal* B*, deux termes terminaux* a *et* b *ainsi que l'ensemble de productions* $p_1 : AX \to aB$, $p_2 : B \to bB$ *et* $p_3 : B \to b$.
Cette grammaire caractérise le langage contenant les mots décrits par l'expression rationnelle ab^+*, c'est-à-dire l'ensemble des mots composés de la lettre* a *suivie de* n *occurrences de* b *où* $n \in \mathbb{N}$. $abbb$*, par exemple, peut être obtenu à partir de l'axiome en appliquant* p_1*, deux fois* p_2*, puis* p_3.

Grâce à la réécriture de graphe, ces grammaires peuvent être transposées aux graphes comme suit.

Définition 4.1. (**Grammaire de graphe**) Une grammaire de graphe est définie par un système (AX, NT, T, P) où

1. AX est l'axiome. Il peut être représenté soit par un nœud fictif soit par un graphe symbolisant le motif minimal de l'application (un client et un serveur pour une application client/-serveur, par exemple).
2. NT est l'ensemble des types de nœuds non terminaux.
3. T est l'ensemble des types de nœuds terminaux.
4. P est un ensemble de règles de réécriture de graphes appelées productions de la grammaire. Ces règles sont constituées des termes de la grammaire, de sorte que tout nœud d'une production est unifiable avec au moins un terme de T ou NT.

Une des particularités les plus notables de ces grammaires est leur aspect génératif. Ainsi, un ensemble de règles de transformation rentre dans la caractérisation même d'un style architectural. Elles sont liées à la spécification des configurations correctes comme le précise la définition suivante.

Définition 4.2. (**Instance d'une grammaire de graphe**) Une instance d'une grammaire (AX, NT, T, P) est un graphe G_1 isomorphe selon (f_i, I_i) à un graphe G_2 pouvant être obtenu à partir de l'axiome AX en appliquant une suite de règles de production de P, tel que G_1 est invariant pour Aff_{I_i}.
Si elle est exempte de nœud unifiable avec un élément de NT, elle est dite **consistante**.

Afin de modéliser la génération des instances d'une grammaire de graphe, on peut s'appuyer sur un graphe d'état. AGG [27] et GROOVE [28], par exemple, proposent la génération automatique d'un tel graphe. Les nœuds de ce type de graphes représentent alors une instance de la grammaire et ses arcs des transformations résultantes de l'application d'une de ses règles de production.

Définition 4.3. (**Graphe de génération**) Le graphe de génération d'une grammaire de graphe $Gr = (AX, NT, T, P)$ est le graphe $G = (V, E, Att)$ où

1. V est l'ensemble des instances de Gr.
2. $E \subseteq V^2$ est un ensemble d'arcs attribués par Att tels que :
   (a) $\forall e \in E, (|Att^e| = 2) \wedge (D_1^e = P) \wedge (D_2^e$ est l'ensemble des morphismes [2]$) \wedge Const(Att_1^e) \wedge Const(Att_2^e)$.
   (b) $\forall (v_1, v_2) \in V^2, (v_1, v_2) \in E \Rightarrow A_1^{(v_1,v_2)} \_ A_2^{(v_1,v_2)}(v_1) = v_2$.

Ce graphe de génération peut être utilisé comme modèle de déploiement, considérant qu'il illustre toutes les séquences possibles de transformations permettant de générer chaque configuration. Sa construction peut être longue, mais elle se passe en phase de conception du système. En outre, ce graphe est invariable pour un style donné et est indépendant du contexte et des évènements se déroulant pendant la phase d'exécution.

4.2.1.2 Correction

Le standard ISO/IEC/IEEE-42010 [12] traitant de la description d'architectures leur prête 19 utilisations fondamentales. L'une d'entre elles est la définition d'un style architectural, plus précisément *"la spécification d'un groupe de systèmes* [de configurations] *partageant des propriétés communes (tel qu'un style architectural)"* [3]. Un tel style décrit les organisations acceptables des éléments d'un système. Ainsi, la spécification du style constitue la base axiomatique de la notion de correction ; sa validité n'est pas questionnée. À un instant donné, un système, ou une application, est donc dans un état correct s'il se conforme au style architectural le spécifiant. Une **configuration** est par conséquent **correcte** vis à vis d'un style architectural si c'est une instance de la grammaire le spécifiant.

Afin de maintenir la cohérence et la stabilité du système, ses évolutions (au moins celles gérées par le gestionnaire autonome) doivent préserver sa correction. Une règle de réécriture est dite correcte lorsque son application maintient *nécessairement* le système dans un état acceptable. En d'autres termes, si une transformation correcte vis à vis d'un style architectural est applicable à une configuration s'y conformant, le résultat de son application s'y conforme également. La définition suivante traduit ces considérations dans le contexte présent.

Définition 4.4. (**Transformation correcte**) Une règle de réécriture r est dite correcte vis à vis d'une grammaire GG (ou du style caractérisé par GG) si l'ensemble des instances de GG est clos pour r : l'application de r à une instance de GG produit nécessairement une instance de GG.

2. homomorphismes ou isomorphismes de sous graphes, le cas échéant
3. "specifying a group of systems sharing common features (such as architectural styles)"

En particulier et par définition, toute règle de production d'une grammaire est correcte vis à vis du style architectural qu'elle caractérise. En effet, l'application d'une production p à un graphe pouvant être obtenu à partir de l'axiome AX en appliquant une suite s de règles de production résulte en un graphe pouvant être obtenu à partir de AX en appliquant la suite de productions formée par la concaténation de p à s.

Remarque 4.1. Il est possible de renforcer la notion de correction en imposant aux instances du style d'être en sus consistantes (c'est à dire exempte de terme non terminal). Dans le cadre particulier de la conservation de la correction par des règles, une telle restriction importe peu. Dans ce nouveau contexte, il suffirait, pour qu'une règle soit correcte, d'ajouter aux conditions présentées dans la suite la non-introduction d'un terme temporaire.

4.2.2 Exemple illustratif : une application distribuée à contexte dynamique

4.2.2.1 Description

Afin de clarifier et d'illustrer les concepts introduits dans ce chapitre, une application concrète et réelle y est utilisée comme exemple. DIET [4] [121] est un répartiteur de charge hiérarchique permettant de dispatcher des tâches sur une infrastructure distribuée, telle une grille ou un nuage. Ce logiciel, initialement développé au sein du Laboratoire de l'Informatique du Parallélisme [5], est intégré à SysFera-DS [6], une solution industrielle pour la gestion et la fédération d'environnements hybrides de calcul haute performance.

L'architecture de DIET est composée d'un ensemble d'agents : des agents maîtres, désignés sous le terme MAs [7], gèrent un groupe de serveurs applicatifs fournissant divers services, nommés SEDs [8]. Cette gestion s'effectue à travers aucune, une ou plusieurs strates d'agents de couches, appelés LAs (pour Layer Agents) et assurant la scalabilité de l'architecture. Les communications entre agents sont assurées par un service de nommage *omniRB*, abrégé en OMNI. Les MAs sont à l'écoute de requêtes clients et les assignent, à l'aide des LAs, aux meilleurs SEDs proposant les services demandés. Une telle architecture se prête à l'adoption d'une vue orientée composants. En outre, son caractère distribué et la nature fluctuante des requêtes clients et des disponibilités matérielles rendent son contexte d'exécution changeant, impliquant la nécessité d'adaptations structurelles au travers de reconfigurations du système.

Dans [21], DIET est décrite en utilisant des diagrammes de classes. Cette caractérisation pose deux restrictions majeures. Premièrement, la gestion de la correction s'effectue par la vérification *a posteriori* de contraintes de style. Ainsi, des roll-backs peuvent s'avérer nécessaires lorsque ces dernières ne sont pas satisfaites. Deuxièmement, le fait qu'un LA puisse en gérer un autre ne pouvait pas être traduit dans le diagramme de classes, excluant les configurations exhibant plusieurs strates de LAs.

Il est à noter que, dans une architecture comportant plusieurs MAs, chaque MA gère son propre ensemble de SEDs et de LAs. Ainsi, si l'on excepte les liaisons vers l'OMNI et celles entre MAs, l'architecture globale présente une structure de forêt (i.e., un ensemble d'arbres). Par conséquent, considérer une architecture simplifiée comportant un seul MA n'entraîne pas de perte de généralité.

4. Distributed Interactive Engineering Toolbox
5. www.ens-lyon.fr/LIP/
6. www.sysfera.com/sysfera-ds.html
7. Master Agents
8. SErver Daemons

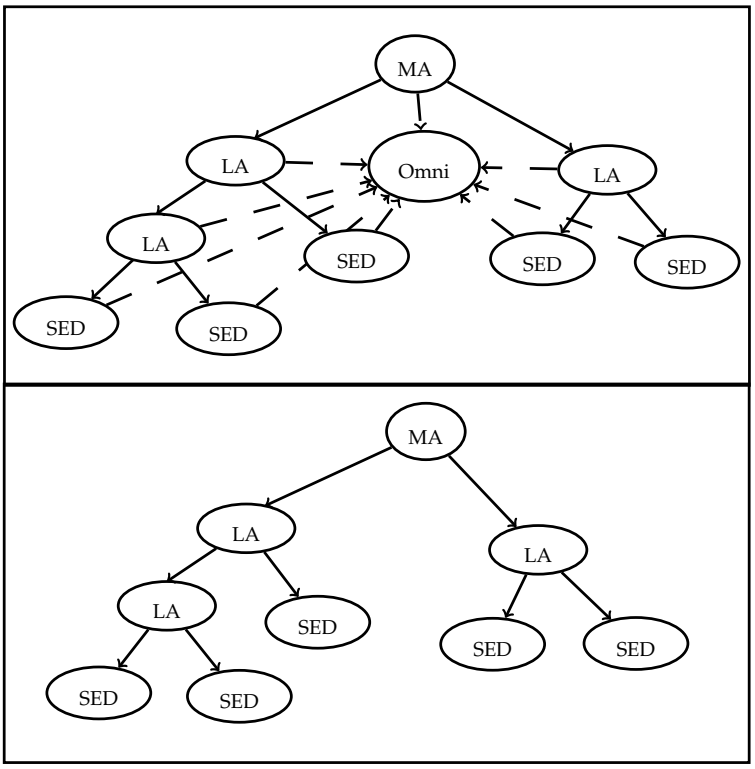


FIGURE 4.1 – Vue logique d'une configuration de DIET, avec et sans service de nommage.

Pour le moment, cette architecture simplifiée sera donc adoptée. Ses caractéristiques principales sont :

1. Lors de son déploiement, chaque composant est déclaré à l'OMNI.
2. Chaque LA et chaque SED a exactement un supérieur hiérarchique, son nœud parent dans le sous arbre du graphe.
3. Chaque MA et chaque LA gèrent au moins un composant. Cette condition peut être trivialement étendu à n'importe quel nombre.

La figure 4.1 offre un exemple de l'apparence d'une configuration de DIET, avec et sans OMNI.

Les attributs suivants seront pris en considération :

1. Un identifiant unique pour chaque composant.
2. La nature de chaque composant.
3. Pour chaque SED, l'ensemble des services qu'il peut fournir.

4.2.2.2 Caractérisation

Cette section est dédiée à la définition de la grammaire de graphe spécifiant sans ambiguïté le style architectural lié à DIET. Il s'agit, dans un premier temps, de définir les archétypes des nœuds intervenant dans la grammaire, puis ses règles de production.

Axiome. Ici, l'axiome est purement théorique et sera représenté par un nœud noté AX.

Termes terminaux. Ces termes définissent des types de nœud en précisant un motif d'attributs qu'ils partagent.

Soit Nat l'ensemble des types de composants, {"OMNI","MA","LA","SED"}. Soit ID l'ensemble des valeurs que peut prendre l'identifiant unique d'un composant. Soit S l'ensemble des services pouvant être assurés. Un SED peut potentiellement offrir plusieurs services, l'attribut correspondant étant par conséquent un élément de $Serv$, l'ensemble des parties de S.

Le service de nommage étant unique, il ne nécessite pas d'identifiant. Son seul attribut est donc relatif à sa nature, et l'archétype $T_o = v_o(("OMNI", Nat))$.

Les agents (LA ou MA) possèdent des attributs de natures similaires : un identifiant et la nature du composant. Ils sont regroupés sous l'archétype $T_a = v_a((id, ID), (nat, Nat))$.

Les SEDs possèdent en sus un attribut relatif aux services fournis, qui est un élément de $Serv$. L'archétype lié aux SEDs est $T_{sed} = v_{sed}((id, ID), ("SED", Nat), (s, Serv))$.

Productions. Les règles de production de la grammaire de graphe formalisent la construction de ses instances en définissant quand et comment un composant peut être déployé. En outre, la grammaire sera définie telle que chaque arc entre deux composants possède deux attributs relatifs à la nature de sa source et de sa destination.

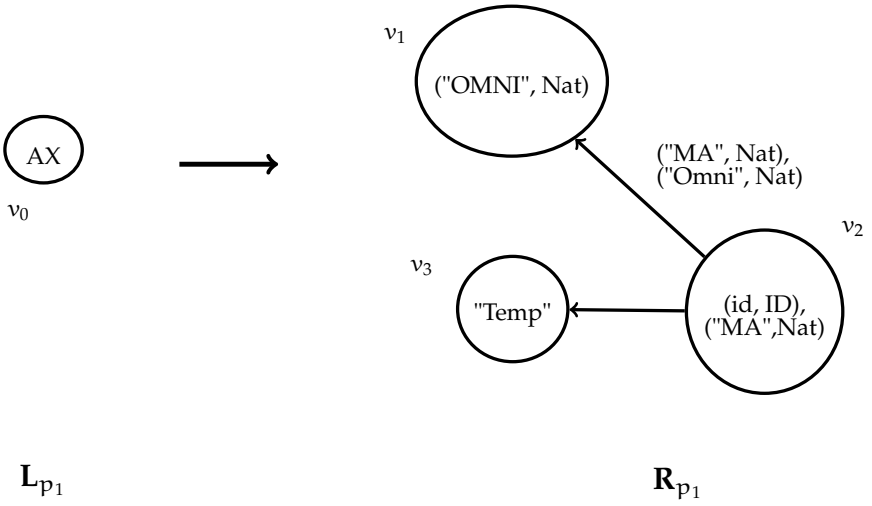


FIGURE 4.2 – Initialisation (p_1)

La première règle (p_1) à définir est l'initialisation consommant le nœud axiomatique. Le service de nommage et le MA sont alors déployés, ainsi qu'un nœud temporaire garantissant que le MA gère au moins un composant. Ce nœud sera instancié plus tard en un LA ou un SED. Finalement, le MA s'enregistre au service de nommage. Cette règle $p_1 = (L_{p_1} \xrightarrow{m_{p_1}} R_{p_1})$ est illustrée dans la figure 4.2. Son morphisme associé, m_{p_1} est caractérisé par la fonction dont le domaine et le codomaine sont l'ensemble vide.

La règle de production p_2, représentée dans la figure 4.3, modélise l'ajout d'un nœud temporaire géré par le MA ou un LA. Ce nœud temporaire a pour vocation à être instancié en un LA ou un SED. Son morphisme associé, m_{p_2} est caractérisé par la fonction associant $v_1 \in V_{L_{p_2}}$ à $v_1 \in V_{R_{p_2}}$.

L'instanciation d'un nœud temporaire en un LA ou un SED est décrite par les règles de production p_3 et p_4, introduites dans les figures 4.5 et 4.4, respectivement. Le déploiement d'un SED ou d'un

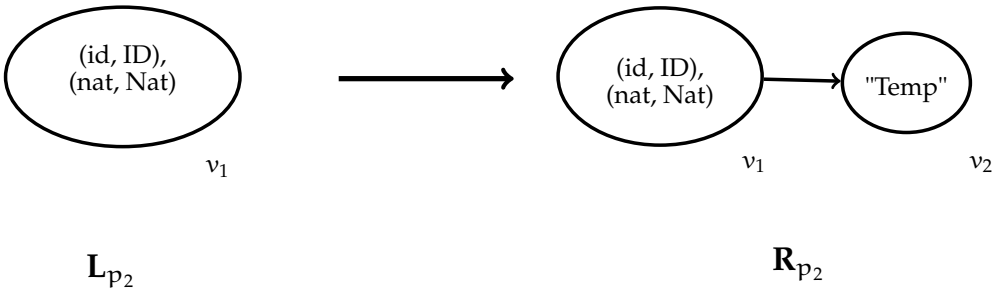


FIGURE 4.3 – Ajout d'un nœud temporaire (p_2)

LA implique son enregistrement au service de nommage. Comme un LA doit gérer au moins un composant, un nœud temporaire est ajouté lors de l'instanciation correspondante. Les morphismes associés à ces règles sont ceux allant du sous graphe de la partie gauche induit par $\{v_1, v_2\}$ dans la partie droite de la règle.

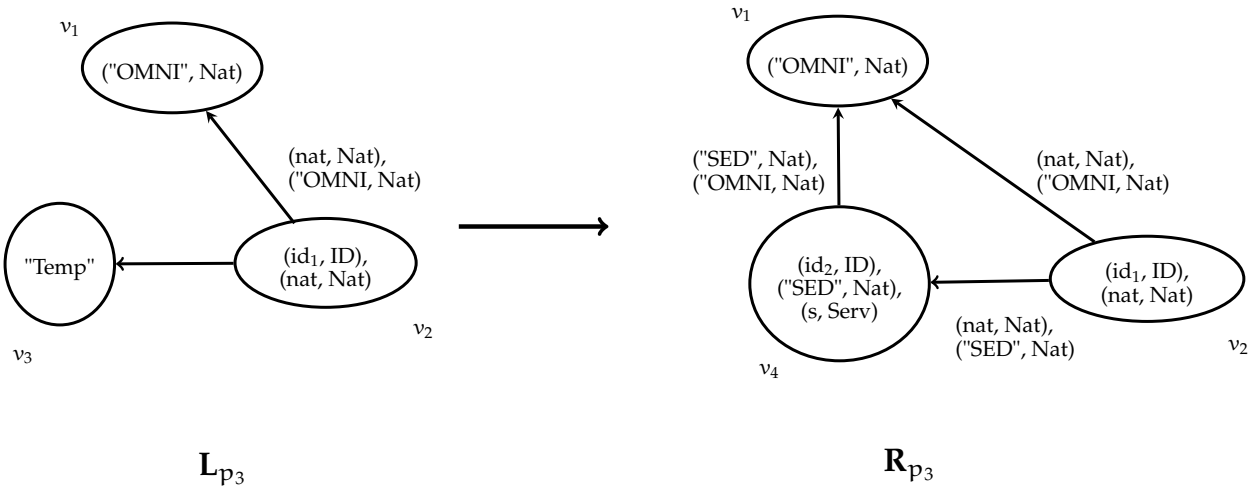


FIGURE 4.4 – Instanciation d'un nœud non terminal en un SED (p_3)

Grammaire de graphe caractérisant DIET Considérant les notations introduites dans cette section, GG_{DIET}, la grammaire de graphe spécifiant formellement DIET est définie telle que GG_{DIET} = $(AX_{DIET}, NT_{DIET}, T_{DIET}, P_{DIET})$ où

$NT_{DIET} = \{v_{temp}(("Temp",\{"Temp"\}))\}$, comme cela n'introduit pas d'ambiguïté, le domaine de définition de l'unique attribut d'un nœud temporaire sera implicite,
$T_{DIET} = \{T_o, T_a, T_{sed}\}$,
$P_{DIET} = \bigcup_{i\in[1,4]} p_i$.

Remarque 4.2. Il est intéressant de noter que, puisqu'une seule règle consomme l'axiome, ce dernier aurait pu être défini comme étant le résultat de celle-ci plutôt qu'une entité purement théorique. L'axiome aurait alors représenté un motif minimal de l'application.

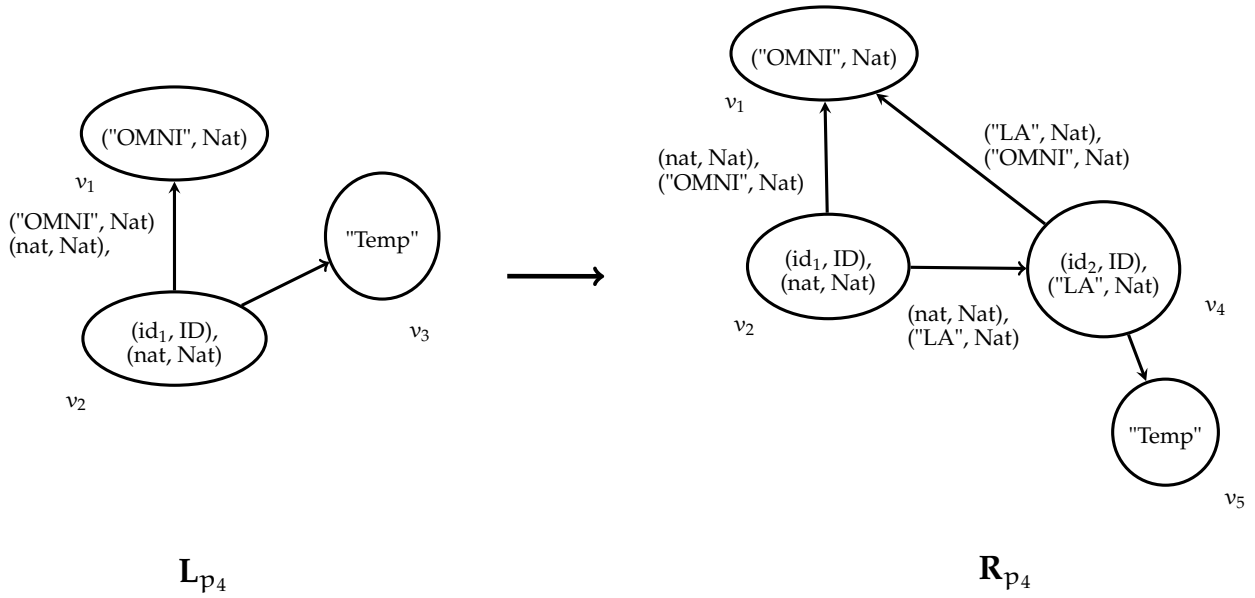


FIGURE 4.5 – Instanciation d'un nœud non terminal en un LA (p_4)

Génération d'instances de la grammaire La figure 4.6 représente un sous-graphe du graphe de génération de DIET qui est étendu et détaillé dans la figure 4.7. Les domaines de définitions des attributs ainsi que les homomorphismes selon lesquels les règles de productions sont appliquées n'ont pas été représentés, cela n'introduisant pas d'ambiguïté.

La configuration finale, notée G_6, contient un serveur de nommage et un MA gérant un LA. Ce dernier gère à son tour un SED et un LA sous lequel se trouve un deuxième SED. G_6 peut être obtenue en appliquant successivement à l'axiome la suite de règles de production $(r_i)_{i\in[1,6]}$ où $r_1 = p_1$, $r_2 = r_5 = p_4$, $r_3 = r_6 = p_3$ et $r_4 = p_2$.

En notant h_i l'homomorphisme selon lequel r_i est appliqué, on obtient donc
$G_6 = p_3\_h_6.p_4\_h_5.p_2\_h_4.p_3\_h_3.p_4\_h_2.p_1\_h_1(AX)$.

Notons G_i chacune des instances intermédiaires. On peut également écrire $AX \xrightarrow{(p_1,h_1)} G_1 \xrightarrow{(p_4,h_2)} G_2 \xrightarrow{(p_3,h_3)} G_3 \xrightarrow{(p_2,h_4)} G_4 \xrightarrow{(p_4,h_5)} G_5 \xrightarrow{(p_3,h_6)} G_6$. Finalement, bien que chaque G_i soit une instance de la grammaire, et donc une configuration correcte de DIET, G_3 et G_6 sont les seules à ne pas contenir de terme non terminal, et donc à être des instances consistantes.

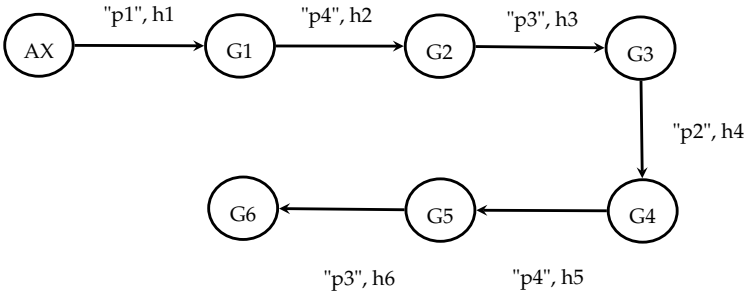


FIGURE 4.6 – Sous graphe du graphe de génération relatif à DIET

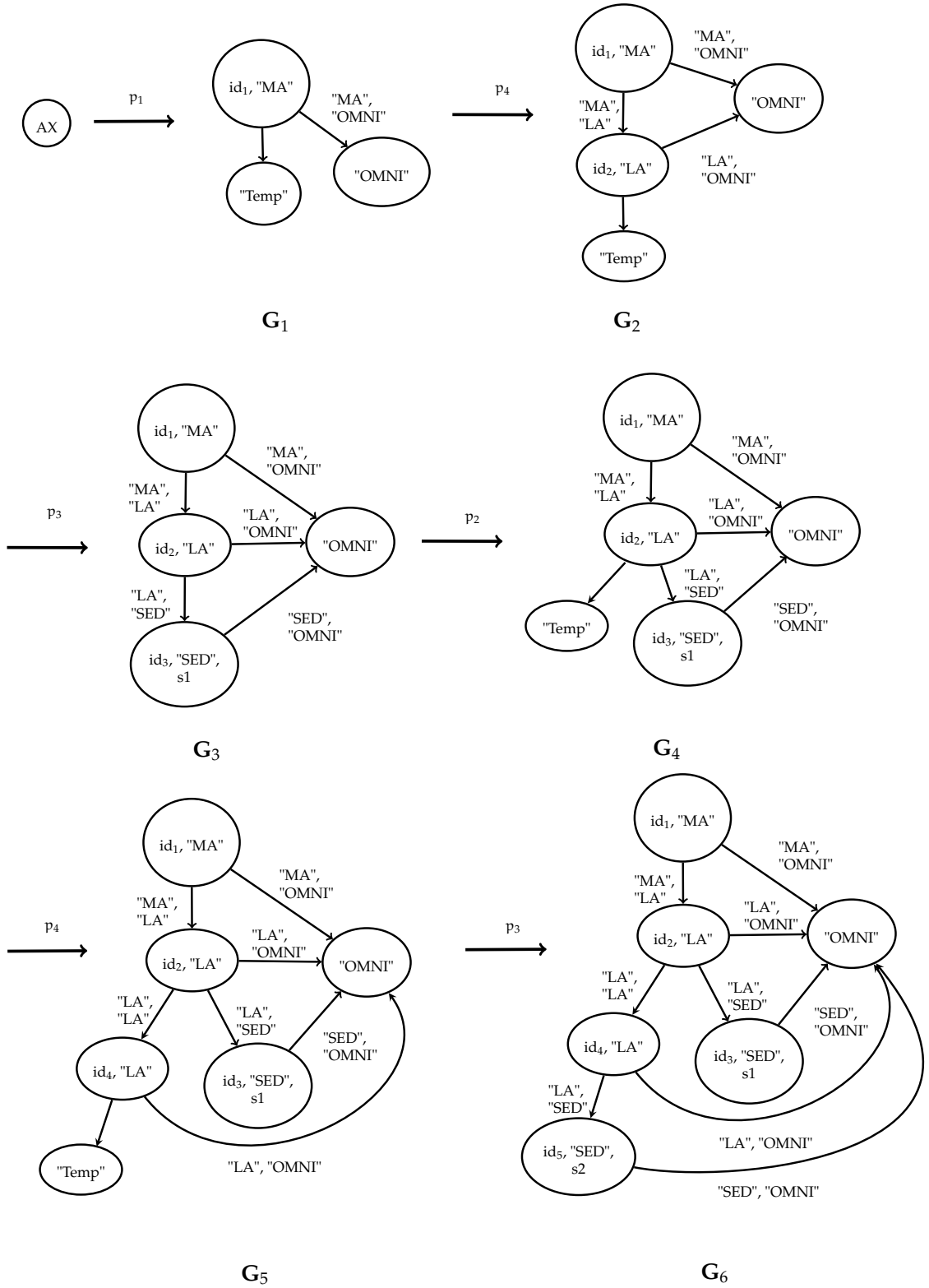


FIGURE 4.7 – Génération d'une configuration correcte de DIET

4.3 Construction et caractérisation générative de transformations correctes

L'auto-protection interne d'un système peut ainsi être assurée si toute transformation intervenant dans une politique de reconfiguration est correcte. Or, la caractérisation d'un style fournit un ensemble de transformations *correctes par définition*, à savoir les productions de la grammaire. L'introduction d'opérations préservant la correction des règles permet donc de générer, à partir de cet ensemble initial, de nouvelles règles correctes. Ce mécanisme permet ainsi la caractérisation générative et la création de règles de transformations correctes par construction pour un style architectural quelconque, étant donné une grammaire de graphe le décrivant.

Les définitions, propriétés et méthodes présentées dans la suite portant sur les règles de réécriture, elles peuvent subtilement varier en fonction de leurs conditions d'applicabilité. En effet, la définition 3.11 stipule que, suivant les approches adoptées, l'occurrence d'un motif dans un graphe hôte peut être conditionnée par l'existence d'un morphisme, d'un homomorphisme ou d'un isomorphisme de sous graphe induit. Par la suite, nous supposons que la recherche de motif est formalisée par l'existence d'un morphisme. Pour obtenir les versions relatives aux autres approches, il faut, lorsqu'il est accompagné d'une astérisque, remplacer le terme morphisme (*morphisme**) par homomorphisme ou isomorphisme de sous-graphe induit.

4.3.1 Inversion

4.3.1.1 Définition et correction

L'inversion exploite la propriété d'inversibilité des règles de réécriture [65]. Elle consiste à définir une transformation inverse annulant les effets d'une autre. Intuitivement, en considérant par exemple une opération consistant à déployer un nouveau serveur pour absorber un pic de charge, l'inversion permet de caractériser la transformation décrivant l'extinction du dit serveur après un retour à la normale. La non injectivité entraînant une perte d'information, une telle opération ne peut s'effectuer que si les morphismes considérés sont injectifs (et donc des homomorphismes).

L'opération d'inversion consiste à créer, à partir d'une règle de réécriture, une règle inverse. Cette opération est classiquement effectuée en échangeant les parties gauche et droite d'une règle, comme formalisé dans la définition suivante.

Définition 4.5. (**Règle de réécriture inverse**) L'inverse d'une règle de réécriture $r = L_r \xrightarrow{h_r} R_r$ avec $h_r = (f_r, \emptyset)$ un homomorphisme de L_r^I dans R_r est une règle de réécriture $r^{-1} = L_{r^{-1}} \xrightarrow{h_{r^{-1}}} R_{r^{-1}}$ avec $h_{r^{-1}} = (f_{r^{-1}}, \emptyset)$ un homomorphisme de $L_{r^{-1}}^I$ dans $R_{r^{-1}}$ telle que :

1. $L_{r^{-1}} = R_r$.
2. $R_{r^{-1}} = L_r$.
3. $V_{L_{r^{-1}}^I} = f_r(Dom(f_r))$. Rappelons que $V_{L_{r^{-1}}^I}$ est égal à $Dom(f_{r^{-1}})$ et que $f_r(Dom(f_r))$ est égal à $Im(f_r) \subseteq Codom(f_r)$.
4. $E_{L_{r^{-1}}^I} = \{e \in E_{R_r} | \exists e_L \in E_{L_r^I}, f_r(e_L) = e\}$.
5. $f_{r^{-1}} : V_{L_{r^{-1}}^I} = Im(f_r) \rightarrow V_{R_{r^{-1}}} \supseteq Dom(f_r)$ telle que $\forall v \in V_{L_{r^{-1}}^I}, f_{r^{-1}}(v)$ est l'unique antécédent de v pour f_r.

Remarque 4.3. Rappelons que pour une règle $r = L \xrightarrow{h_r} R$, avec $h_r = (f_r, I_r) : L^I \to R$, la partie invariante est spécifiée par L^I. Dans les faits, lorsque r est appliquée à un graphe G selon un homomorphisme $h = (f, I)$, les nœuds de G identifiés avec ceux de L^I par f ne se retrouve pas directement dans le graphe résultant : un tel nœud v_G est "remplacé" par un nœud équivalent v_R étant l'image par f_r de l'unique (dans le cas d'un homomorphisme) nœud v_L dont l'image par f est v_G. Afin de simplifier les notations, est comme nous ne sommes pas à un matching $f_r \circ f^{-1}$ (voire f^{-1} si f_r est la fonction identité) près, cette association est négligée dans cette sous-section 4.3.1.

Une telle règle r^{-1} est l'inverse de r et possède la propriété fondamentale suivante.

Propriété 4.1. *Pour toute règle de réécriture* $r = L_r \xrightarrow{h_r} R_r$, $h_r = (f_r, \emptyset) : L_r^I \to R_r$ *et son inverse* $r^{-1} = L_{r^{-1}} \xrightarrow{h_{r^{-1}}} R_{r^{-1}}$, $h_{r^{-1}} = (f_{r^{-1}}, \emptyset) : L_{r^{-1}}^I \to R_{r^{-1}}$, *pour tout graphe* G *:*
s'il existe un homomorphisme (resp. un isomorphisme de sous graphe induit) $h_L^r = (f_L^r, I_L^r)$ *selon lequel* r *est applicable à* G *et si cette application n'entraîne pas l'apparition d'arc suspendu,*
alors il existe un homomorphisme (resp. un isomorphisme de sous graphe induit) $h_L^{r^{-1}}$*, caractérisé par la fonction* $f_L^{r^{-1}}$ *coïncidant avec* $f_L^r \circ f_{r^{-1}}$ *sur* $L_{r^{-1}}^I$ *et avec la fonction identité sur le reste de son ensemble de définition, par rapport auquel* r^{-1} *est applicable à* $r\_h_L^r(G)$ *tel que* G *et* $r^{-1}\_h_L^{r^{-1}}.r\_h_L^r(G)$ *sont isomorphes.*

Notons que, sans la simplification décrite dans la remarque 4.3, $f_L^{r^{-1}}$ aurait coïncidé avec $f_{r^{-1}}$ sur $L^{r^{-1}}$.

L'inversion est la seule opération étudiée qui, seule, ne permet pas nécessairement de garantir la conservation de la correction. Par exemple, en considérant DIET, l'application introduite dans la sous-section 4.2.2, la règle inverse de p_4 (elle-même définie dans la figure 4.5) décrit la suppression d'un LA. Elle pourrait être appliquée à la configuration notée G_4 dans la figure 4.5. Une telle application ne produirait pas une configuration correcte, car elle laisserait un SED sans nœud père. Ce SED est déployé par une application de la règle p_3 postérieure à l'exécution de la production p_4, les effets de cette dernière étant annulés par l'utilisation de la règle inverse considérée. Or, ces deux transformations ne sont pas indépendantes : les conditions d'applicabilité de p_3 sont remplies pour G_2 grâce à cette première réécriture. L'annulation de celle-ci entraîne donc une invalidation des dites conditions.

4.3.1.2 Concrétisation

Trois conditions sont nécessaires pour que l'inversion conserve la correction d'une règle correcte r.

Première condition. L'application de la règle correcte r ne doit pas pouvoir faire apparaître d'arc suspendu.

Seconde condition. La règle inverse ne doit être applicable que si elle annule (théoriquement) l'effet d'une précédente application de r.
C'est à dire que, pour tout graphe G représentant une instance d'un style S et toute règle r^{-1} inverse d'une règle r correcte pour S, si r^{-1} est applicable à G selon un homomorphisme h, alors G peut être obtenu à partir de l'axiome de la grammaire en appliquant une suite de règles correctes, l'une d'entre elles étant r et les autres des productions.
Qui plus est, r^{-1} annule l'effet de l'application de r, c'est à dire que, en notant h_L^r l'homomorphisme

selon lequel r a été appliquée, h est caractérisé par la fonction coïncidant avec $f^r_l \circ f_{r^{-1}}$ sur $L^I_{r^{-1}}$ et la fonction identité sur le reste de son domaine de définition (respectant ainsi la propriété 4.1).

Il convient d'insister sur le fait que si G doit pouvoir être obtenu de cette manière ; il n'est pas nécessaire qu'il ait effectivement été produit ainsi. Le système peut avoir atteint un tel état après un cycle de vie incluant diverses transformations.

Cette propriété est propre à GG et r. Intuitivement, pour un ensemble de productions $p_1 : AX \to a$ et $p_2 : AX \to ab$, il est possible qu'un motif correspondant à la partie droite de la règle p_1 (i.e. a) soit présent dans un mot qui ne peut être dérivé en l'utilisant (e.g. ab).

Ce n'est toutefois pas une propriété liée au style architectural lui-même (c'est à dire de l'ensemble des configurations correctes) mais à sa caractérisation (donc de la grammaire). En effet, la grammaire précédente peut être réécrite de manière à ce que chacune de ses productions vérifie la dite propriété tout en spécifiant le même langage. Une telle réécriture peut par exemple résulter dans la grammaire possédant B un terme non terminal, ϵ le mot vide et l'ensemble de productions : $p_a : AX \to aB$, $p_b : B \to \epsilon$ et $p_c : B \to b$.

Troisième condition. Cette condition est liée à l'exemple d'inversion incorrecte présentée précédemment. Il s'agit, pour une transformation inverse t^{-1} (on considère une transformation t comme étant l'application d'une règle sur un certain graphe selon un certain homomorphisme) annulant l'effet d'une transformation t, de n'invalider les conditions d'applicabilité d'aucune transformation ayant eu lieu entre t et t^{-1}. En particulier, t^{-1} ne doit supprimer aucun élément nécessaire à l'application d'une telle transformation intermédiaire.

Dans la réécriture de graphe en général, cette notion de nécessité se traduit par la présence d'un élément dans l'image de la partie gauche de la règle entrant dans la spécification de la transformation. Il s'agit donc d'implémenter un compteur de reconnaissance gardant une trace, pour chaque élément d'une configuration, du nombre de fois où il a fait partie du motif permettant l'application d'une transformation. Un élément peut ainsi être effacé si un tel compteur est nul.

Une grammaire peut être augmentée automatiquement afin d'implémenter ce mécanisme, sans modifier le style architectural qu'elle caractérise, et ce de manière automatique et transparente pour l'utilisateur en suivant l'algorithme 4.1. Cette augmentation ne peut toutefois avoir lieu que si une règle de production ne peut supprimer un certain nœud uniquement si la valeur de son compteur de reconnaissance est fixée (par défaut à "0").

Ainsi, cette troisième condition théorique peut en général facilement être vérifiée concrètement ; elle se transforme néanmoins en une autre condition (plus facilement vérifiable au niveau de la grammaire) : "Tout élément supprimé par une règle de production a été impliqué dans l'application d'un nombre fixé de règles de productions en apparaissant dans le motif nécessaire à leur application". Pour GG_{DIET} introduite dans la section 4.2.2, par exemple, seuls l'axiome (via la règle p_1) et un nœud temporaire (via p_3 ou p_4) peuvent être supprimés par l'application d'une règle de production. Or, s'ils sont supprimés, leur compteur de reconnaissance est nécessairement nul, puisqu'ils n'apparaissent dans aucune partie invariante d'une production.

En particulier ici, le fait de ne pas considérer des multi-graphes ajoute une contrainte d'applicabilité supplémentaire : un arc ne peut être ajouté s'il existe déjà. Pour ne pas invalider les conditions d'applicabilité des transformations intervenant entre t et t^{-1} , t^{-1} ne doit donc pas ajouter un arc que l'une d'entre elle ajoute. Supposons que ça soit le cas, notons t_2 cette autre transformation et e l'arc ajouté par t_2 et t^{-1}. Comme t^{-1} est applicable, e n'est pas présent dans le graphe au moment de son application et il existe une transformation t_3 intervenant entre t_2 et t_r^{-1} supprimant

Données : $GG = (AX, NT, T, P)$, une grammaire de graphe.
Pré-conditions : Tout élément supprimé par une règle de production a été impliqué dans l'application d'un nombre fixé de règles de productions en apparaissant dans le motif nécessaire à leur application.
début
  pour chaque $terme \in NT \cup T$ **faire**
    $Att_0^{terme} = (n, \mathbb{N})$
    `/* À chaque membre de T et NT est adjoint un attribut` $(n, \mathbb{N})$`. Cet attribut représente le compteur de reconnaissance. Il peut être caché à l'utilisateur et est noté comme étant le 0-ième attribut de chaque élément. */`
  fin
  pour chaque $p = (L_p \xrightarrow{m_p=(f_p,\emptyset)} R_p) \in P, m_p : L_p^I \to R_p$ **faire**
    pour chaque $el_L \in V_{L_p} \cup E_{L_p}$ **faire**
      si $el_L \in V_{L_p^I} \cup E_{L_p^I}$ **alors**
        $A_0^{el_L^I} = n_{el_L}$
        `/* Les compteurs des éléments invariants sont indépendants et peuvent prendre n'importe quelle valeur. */`
      sinon
        $Const(A_0^{el_L}) =$ "0"
        `/* Les compteurs des éléments effacés sont nécessairement constants (et nuls par défaut). */`
      fin
    fin
    pour chaque $el_R \in V_{R_p} \cup E_{R_p}$ **faire**
      si $\exists el_L \in V_{L_p^I} \cup E_{L_p^I}, f_p(el) = el_R$ **alors**
        $A_0^{el_R} = A_0^{el_L} + 1.$
        `/* Le compteur de chaque élément invariant est incrémenté lors de l'application d'une règle. D'une manière similaire à AGG [27], cette modification (ici, une incrémentation) est simplement notée dans la partie droite de la règle. */`
      sinon
        $A_0^{el_R} =$ "0".
        `/* Le compteur est initialisé à "0" lors de l'addition d'un élément. */`
      fin
    fin
  fin
fin

ALGORITHME 4.1 - Augmentation d'une grammaire

e. Pour pouvoir vérifier au niveau d'une grammaire que c'est impossible, on peut par exemple imposer qu'aucune de ses règles de productions ne supprime un arc entre deux nœuds de sa partie invariante[9].

L'inversion préserve donc la correction des règles si ces conditions sont remplies, comme formalisé dans le théorème suivant.

Théorème 4.2. *L'inversion* $r^{-1} = L_{r^{-1}} \xrightarrow{m_{r^{-1}}} R_{r^{-1}}$ *d'une règle correcte* $r = L_r \xrightarrow{m_r} R_r$ *vis à vis d'une grammaire augmentée* GG *est elle même correcte vis à vis de* GG *si :*

1. $\forall p \in P_{GG}, (v_1, v_2) \in V^I_{L_p} \cap E_{L_p} \implies (v_1, v_2) \in E^I_{L_p}$. *Une production de* GG *ne peut supprimer d'arc entre deux nœuds de sa partie invariante.*
2. *l'application de* r *ne peut donner lieu à l'apparition d'arc suspendus.*
3. *Pour toute instance* G *de* GG, $\exists h = (f_h, I_h) : R_r \to G \implies$
   $(\exists n \in \mathbb{N}, \exists((p_i, h_i))_{i\in[1,n]}, G = p_n\_h_n.(\dots).p_1\_h_1(AX_{GG}))\wedge$
   $(\exists k \in [1,n], p_k = r \wedge \forall j \in [1,n], j \neq k \implies p_j \in P_{GG})\wedge$
   $(\forall v \in V_{R_r}, (v \in Dom(f_r) \wedge f_h(v) = f_{h_k}(f_r^{-1}(v))) \vee f_h(v) = v)$. *Un motif homomorphe à* R_p *dans une instance* G *de* GG *implique que* G *peut être obtenue à partir de l'axiome de* GG *en appliquant une suite de règles, l'une d'entre elle étant* p *et introduisant le motif, les autres étant des productions.*

La preuve de ce théorème est détaillée dans l'annexe B.

4.3.1.3 Illustration

La grammaire définie en 4.2.2.2 ne respecte pas la seconde condition de correction. Par exemple, la règle p_2^{-1} pourrait être appliquée sur G_1. Or, la configuration G_1 ne peut être obtenue à partir de l'axiome en appliquant une séquence de règles de productions contenant p_2. Il convient donc dans un premier temps de modifier la grammaire afin d'obtenir cette première propriété. Le non respect des hypothèses vient de la dualité des significations des nœuds temporaires (respect de la cardinalité minimal ou simple ajout). Ils sont soit ajoutés par p_1 ou p_4 et liés au nombre minimal de composants gérés, soit ajoutés par p_2, auquel cas ils figurent l'ajout d'un nouveau composant. Il convient donc d'introduire un second archétype de nœud non terminal. N'importe quel nœud non terminal peut être instancié en utilisant les règles p_3 et p_4. Afin de ne pas perdre d'information lors d'une instanciation, un attribut est également ajouté aux composants pour garder la trace de la nature du nœud temporaire duquel ils découlent.

Dans un deuxième temps, la grammaire est augmentée suivant l'algorithme 4.1. Les ajouts liés à l'augmentation sont représentés en bleu. La grammaire résultante est décrite par :

$\tilde{GG}_{DIET} = (AX_{DIET}, \tilde{T}_{DIET}, \tilde{P}_{DIET})$, où

- $\tilde{NT}_{DIET} = \{v_{temp}((n, \mathbb{N}), (nt, NT = \{"nt_1", "nt_2"\})\}$. En plus de l'addition du compteur, deux types de termes non terminaux ("nt_1" et "nt_2") sont considérés afin de respecter la première condition.
- $\tilde{T}_{DIET} = \{\tilde{T}_a, \tilde{T}_{sed}, \tilde{T}_o\}$ avec $\tilde{T}_a = v_a((n, \mathbb{N}), (id, ID), (nat, Nat), (nt, NT))$,
  $\tilde{T}_{sed} = v_{sed}((n, \mathbb{N}), (id, ID), ("SED", Nat), (s, Serv), (nt, NT))$ et
  $\tilde{T}_o = v_o((n, \mathbb{N}), ("OMNI", Nat))$. Chaque terme terminal est adjoint d'un attribut représentant le compteur de reconnaissance.
- $\tilde{P}_{DIET} = \bigcup_{i\in[1,4]} \tilde{p}_i$ les règles définies dans la figure 4.8.

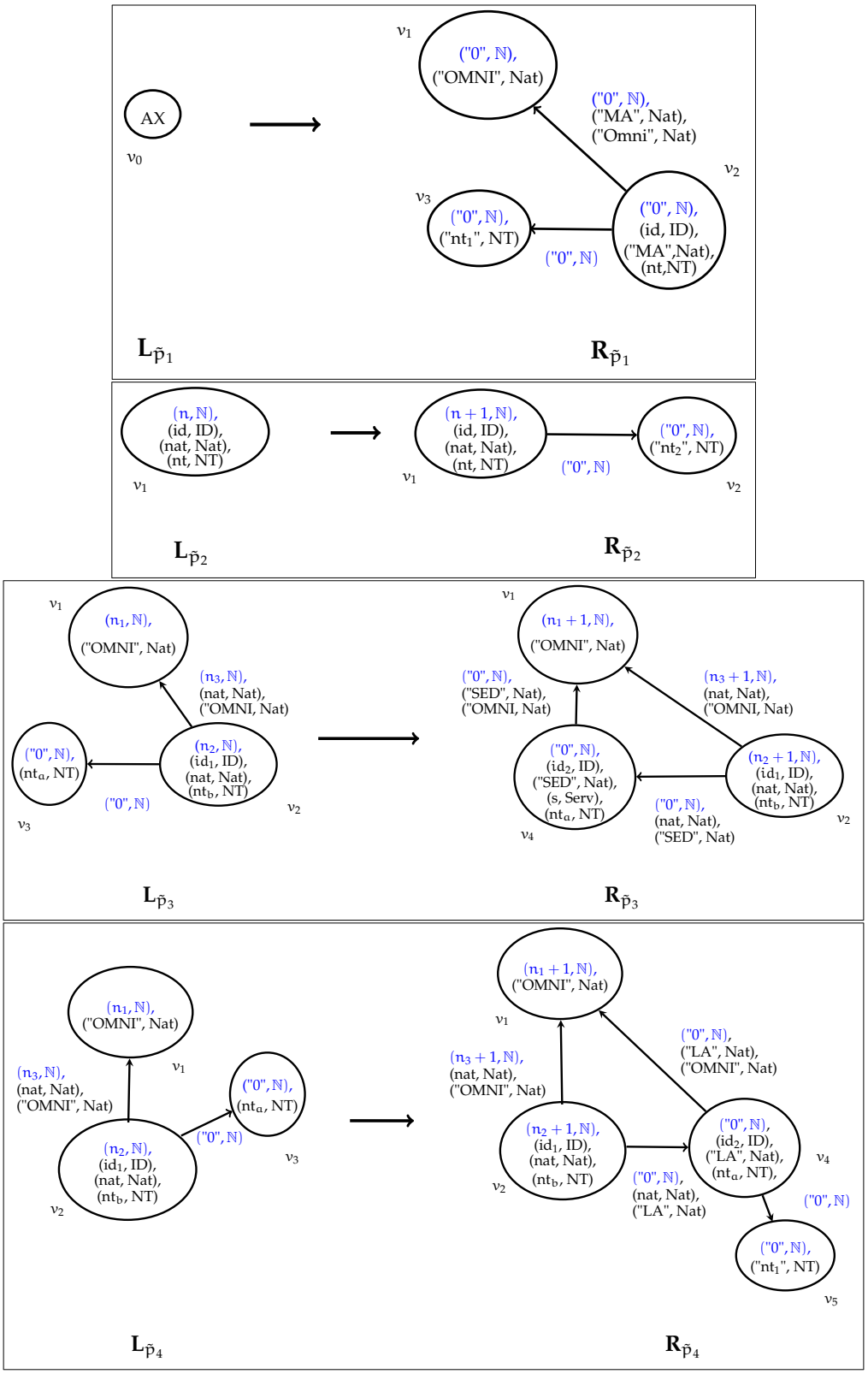


FIGURE 4.8 – Productions de la grammaire augmentée décrivant DIET

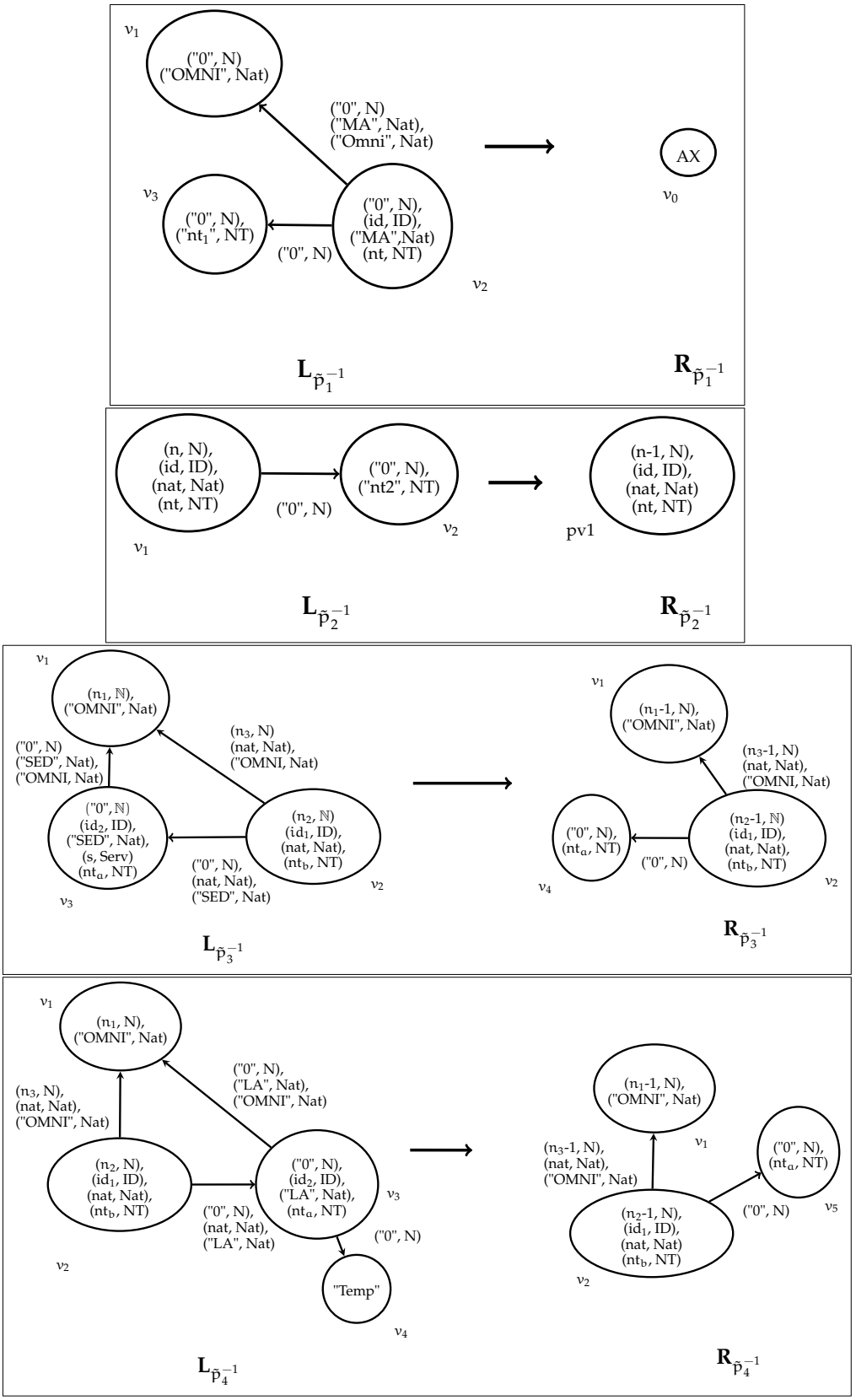


FIGURE 4.9 – Inversion des productions de la grammaire augmentée décrivant DIET

Toute règle de production de cette grammaire augmentée respecte les conditions du théorème 4.2, et l'inversion conserve donc leur correction par rapport au style qu'elle caractérise. L'inversion d'une règle consiste alors simplement à échanger ses parties droite et gauche. Les inverses des productions de la grammaire $\tilde{\mathrm{GG}}_{\mathrm{DIET}}$ sont ainsi présentées dans la figure 4.9.

Notamment, comme le compteur d'un élément ajouté par une production est initialisé à 0, il apparaît comme étant nul dans la règle inverse décrivant sa suppression. De plus, comme le compteur de chaque élément invariant vis à vis d'une production est incrémenté lors de son application, le compteur de chaque élément invariant vis à vis d'une règle inverse est décrémenté lors de son application.

9. Une contrainte plus complexe mais plus faible aurait pû être imposée. Nous privilégions ici la simplicité comme cette contrainte n'est nécessaire qu'en l'absence de multi-graphes.

4.3.2 Composition

4.3.2.1 Définition et correction

D'après les définitions classiques, la composition est un opérateur binaire sur un ensemble d'opérations (en général des applications) visant à conserver l'applicabilité et le résultat d'une exécution séquentielle.

La possibilité de composer les règles de réécriture de graphe dépend du formalisme employé pour leur spécification [122]. Elle favorise la ré-utilisabilité de règles et permet de les décomposer afin d'améliorer leur scalabilité et de faciliter leur compréhension [123, 124]. Elle permet également d'éviter certains états. Un état correct peut comporter des nœuds temporaires qui sont, en général, purement logiques et n'ont pas de correspondance dans le système réel. Si la présence de tels nœuds n'est pas souhaitée, la composition de transformations peut permettre de garantir leur absence.

Une opération de composition est le processus créant une règle de réécriture composée à partir de deux autres règles. La définition suivante formalise la notion de règle composée.

Définition 4.6. (**Règle de réécriture composée**) Une règle de réécriture r est une règle composée de deux règles p et q si, pour tout graphe G, l'existence d'un morphisme* m_L^r selon lequel r est applicable à G implique l'existence d'un couple de morphismes* (m_L^q, m_L^p) de sorte que q est applicable à G selon m_L^q, que p est applicable à $q\_m_L^q(G)$ selon m_L^p et que $r\_m_L^r(G) = p\_m_L^p.q\_m_L^q(G)$.

Contrairement aux définitions classiques, celle-ci n'implique pas l'équivalence des applicabilités ; il n'est pas nécessaire qu'il existe, pour toute application séquentielle, une composition équivalente. Il suffit que l'applicabilité d'une règle composée implique l'applicabilité d'une séquence et que leurs résultats soient similaires. Il s'agit à présent de montrer que la composition conserve effectivement la correction des transformations.

Théorème 4.3. *Une règle composée de deux règles correctes (vis à vis d'un style architectural) est également correcte (vis à vis du même style).*

Démonstration. Soit G un graphe représentant une configuration correcte d'un style architectural S caractérisé par $Gr = (AX, NT, T, P)$, p et q deux règles de réécriture correctes pour S et q une règle composée de p et q.

Supposons que r soit applicable à G selon un morphisme* m_L^r et prouvons que le résultat de son application est nécessairement correct. Selon la définition 4.6, il existe un couple de morphismes* (m_L^q, m_L^p) de sorte que q est applicable à G selon m_L^q, que p est applicable à $q\_m_L^q(G)$ selon m_L^p et que $r\_m_L^r(G) = p\_m_L^p.q\_m_L^q(G)$.

q et G étant correctes par hypothèse, $q\_m_L^q(G)$ est également une configuration correcte. p étant correcte par hypothèse, $p\_m_L^p.q\_m_L^q(G)$, et par conséquent $r\_m_L^r(G)$, est également une configuration correcte. □

4.3.2.2 Concrétisation.

La composition de règles est généralement étudiée d'un point de vue catégoriel abstrait, ce dernier facilitant leur définition et l'élaboration de preuves [71]. Ci-après est proposée une description concrète du processus de composition, c'est à dire des opérations à effectuer afin de construire une

règle composée. Cette approche concrète et opérationnelle facilite l'implémentation et l'automatisation de l'opération.

Comme l'application d'une règle dépend de la reconnaissance d'un motif, la composition dépend également :

1. du choix du formalisme utilisé pour cette reconnaissance (morphisme, homomorphisme ou isomorphisme de sous graphe induit).
2. s'il peut en exister plusieurs, du motif impliqué dans l'application des deux règles à la fois. Ce dernier est la portion de la partie droite de la première règle impliquée dans le motif selon lequel la seconde règle est appliquée.

Les algorithmes 4.2 à 4.8 décrivent les conditions selon lesquelles la construction d'une règle composée en fonction d'un motif commun aux deux règles initiales est possible, ainsi que la construction elle-même. Ils sont donnés en supposant qu'un morphisme conditionne l'applicabilité de la règle. Les modifications à apporter pour les transposer à d'autres formalismes sont toutefois indiquées. Chacune des notations est considérée globale et visible par chacun des algorithmes.

L'algorithme 4.3 décrit les six pré-conditions à la composition de deux règles suivant une certaine compatibilité. La première condition d'applicabilité stipule que s'il existe un arc de L_p dont une seule extrémité a une image dans R_q par f, alors cette image n'est pas ajoutée lors de l'application de q (et correspond donc à un nœud présent avant son application). En effet, directement après l'application d'une règle (et en l'absence d'instruction de connexion), un arc ne peut exister entre un nœud fraîchement ajouté et un nœud présent dans le graphe hôte avant sa transformation.

La seconde porte encore sur les arcs de L_p, mais cette fois sur ceux dont les deux extrémités ont des images dans R_q par f. Elle impose que si au moins une des deux images est ajoutée lors de l'application de q, alors il existe un arc entre ces images dans R_q. En effet, directement après l'application d'une règle (et en l'absence d'instruction de connexion), pour qu'un arc soit présent entre deux nœuds de sa partie droite, il doit soit être présent avant son application (auquel cas ses deux extrémités sont invariantes pour q), soit être ajouté lors de celle-ci. Si l'une des deux extrémités est ajoutée lors de la transformation, la première possibilité est exclue et l'arc doit donc nécessairement être ajouté par la transformation.

La troisième condition porte sur l'applicabilité de p *a priori*.
La condition 3.a stipule que p ne peut ajouter un arc déjà présent du fait de q. La condition 3.b impose que deux nœuds ayant la même image par f n'ont soit pas d'image par f_p soit la même. Cette condition est nécessairement remplie si f est injective, notamment dans le cas où la recherche de motif est formalisée par l'existence d'un homomorphisme ou d'un isomorphisme de sous graphe induit.

La quatrième pré-condition permet de vérifier la seconde condition d'applicabilité des règles de réécriture. Pour rappel, cette dernière est introduite du fait que les graphes considérés ne sont pas de multi-graphes : un arc ne peut être ajouté à un graphe dans lequel il existe déjà. Il faut que p n'efface pas, explicitement ou implicitement, d'arc introduit par q entre deux de ses nœuds invariants. Dans le cas contraire, la composée pourrait être applicable à un graphe possédant un tel arc, alors que q (et par extension la séquence q puis p) ne pourrait lui être appliquée. Cela signifie toutefois qu'il existe des cas où une séquence peut être appliquée sans qu'il existe de composition équivalente.

Les deux autres conditions sont liées à la troisième condition d'applicabilité des règles de réécriture relative aux associations non-injectives. La quatrième impose que tout nœud invariant pour q (et apparaissant dans q) soit invariant pour p. Finalement, la cinquième condition impose que seuls les nœuds ayant la même image par f puissent avoir la même image par f_p.

Données : $p = (L_p \xrightarrow{m_p} R_p)$ et $q = (L_q \xrightarrow{m_q} R_q)$, deux règles de réécriture de graphe avec m_p et m_q des morphismes $(f_p, \emptyset)$ et $(f_q, \emptyset)$ de L_p^I dans R_p et L_q^I dans R_q respectivement.
$C = (f, I)$, une compatibilité telle que L_p et R_q sont C-compatibles. `/* Il existe forcément une telle compatibilité, quitte à ce que` $Dom(f) = Codom(f) = \emptyset$`. Dans ce cas, les deux règles sont appliquées sur des parties indépendantes du graphe. */`
`/* Dans l'approche adoptant des homomorphismes,` f `est en plus injective. En considérant des isomorphismes de sous graphes induits,` (f, I) `caractérise un isomorphisme entre les deux sous graphes induits par` $Dom(f)$ `et` $Im(f)$ `plutôt qu'une compatibilité. */`
Notations
$L_p^C \leftarrow$ le sous graphe de L_p induit par $Dom(f)$.
$R_q^C \leftarrow$ le sous graphe de R_q induit par $Im(f)$.
$R_q^I \leftarrow$ le sous graphe de R_q tel que $\forall el_R \in V_{R_q} \cup E_{R_q}, el_R \in V_{R_q^I} \cup E_{R_q^I}$
$\Leftrightarrow \exists el_L \in V_{L_q^I} \cup E_{L_q^I}, f_q(el_L) = el_R$.
`/* Les éléments de` R_q^I `sont les images des éléments de` L_q^I `par` f_q`, de sorte que` R_q^I `est la partie invariante de q (aux fusions de nœuds près). */`
$L_p^{C_q} \leftarrow$ le sous graphe de L_p^C tel que $\forall el \in V_{L_p^C} \cup E_{L_p^C}, el \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}} \Leftrightarrow f(el) \in V_{R_q} \cup E_{R_q}$.
`/* Les éléments de` $L_p^{C_q}$ `sont les éléments de` L_p^C `ayant une image dans` R_q `par` f`, de sorte que` $L_p^{C_q}$ `est` L_p^C `privé des ses arcs n'ayant pas d'image par` f `dans` R_q `. */`
$L_p^{C_q^I} \leftarrow$ le sous graphe de $L_p^{C_q}$ tel que $\forall el \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, el \in V_{L_p^{C_q^I}} \cup E_{L_p^{C_q^I}} \Leftrightarrow f(el) \in V_{R_q^I} \cup E_{R_q^I}$.
`/* Les éléments de` $L_p^{C_q^I}$ `sont les éléments de` $L_p^{C_q}$ `dont l'image par` f `est dans` R_q^I`, c'est à dire les éléments dont l'image est dans la partie invariante de q. */`
si *Pré-conditions* **alors**
  `/* La composition est possible, et son résultat est la règle` $p \circ_C q$`, appelée ici r, telle que` $r = L_r \xrightarrow{m_r} R_r$ `*/`
  $L_r \leftarrow$ *Gauche*
  $R_r \leftarrow$ *Droite*
  $L_r^I \leftarrow$ (*Inv_V, Inv_E*)
  $f_r \leftarrow$ *Mapping* $m_r \leftarrow$ le morphisme de L_r^I dans R_r caractérisé par la fonction f_r et l'ensemble d'identification vide.
  retourner $r = L_r \xrightarrow{m_r} R_r$
fin

ALGORITHME 4.2 - Composition de règles de réécriture en fonction d'un motif commun, $\circ_C$

Pré-conditions

/* La composition de p et q en fonction de C est définie s'il est possible qu'elles soient séquentiellement appliquées avec le motif commun C. */

/* Pour ce faire, les trois conditions suivantes doivent être remplies */

1. $\forall v_1 \in V_{L_p^C}, \forall v_2 \in V_{L_p} \backslash V_{L_p^C}, ((v_1, v_2) \in E_{L_p} \vee (v_2, v_1) \in E_{L_p}) \implies f(v_1) \in Im(f_q)$.
2. (a) $\forall (v_1, v_2) \in V^2_{L_p^C}, f((v_1, v_2)) \notin Im(f_q)^2 \implies ((v_1, v_2) \in E_{L_p} \implies f((v_1, v_2)) \in E_{R_q})$
   (b) $\forall (v_1, v_2) \in V^2_{L_p^C}, \exists (v_3, v_4) \in V^2_{L_q^I}, f_q(v_3, v_4) = f(v_1, v_2) \implies$
   $((v_1, v_2) \in E_{L_p} \implies (v_3, v_4) \notin E_{R_q} \vee (v_3, v_4) \in E_{R_q^I})$.
3. Les conditions d'applicabilité de p ne sont pas violées *a priori* :
   (a) $\forall (v_1, v_2) \in (V_{L_p^I} \cap V_{L_p^C})^2, ((v_1, v_2) \notin E_{L_p} \wedge f_p((v_1, v_2)) \in E_{R_p}) \implies f(v_1, v_2) \notin E_{R_q}$.
   (b) $\forall (v_1, v_2) \in Dom(f)^2, f(v_1) = f(v_2) \implies (v_1 \notin Dom(f_p) \wedge v_2 \notin Dom(f_p)) \vee ((v_1, v_2) \in Dom(f_p)^2 \wedge f_p(v_1) = f_p(v_2))$.

/* Il faut toutefois inclure trois autres conditions : */

4. $\forall (v_q^1, v_q^2) \in V^2_{L_q^I}, ((v_q^1, v_q^2) \notin E_{L_q} \wedge f_q((v_q^1, v_q^2)) \in E_{R_q}) \implies$
   $(\forall v_p \in V_{L_p^C}, (f(v_p) = f_q(v_q^1) \vee f(v_p) = f_q(v_q^2)) \implies v_p \in V_{L_p^I}) \wedge$
   $(\forall (v_p^1, v_p^2) \in V^2_{L_p^C}, f((v_p^1, v_p^2)) = f_q((v_q^1, v_q^2)) \implies ((v_p^1, v_p^2) \notin E_{L_p} \vee (v_p^1, v_p^2) \in E_{L_p^I})$.
5. $V_{L_p^{C_q^I}} \subseteq V_{L_p^I}$.
6. $\forall (v_1, v_2) \in (V_{L_p^C} \cap V_{L_p^I})^2, f_p(v_1) = f_p(v_2) \implies f(v_1) = f(v_2)$.

ALGORITHME 4.3 - *Pré-conditions* pour la composition

$m_L \leftarrow$ le morphisme de $Aff_I(L_p^{C_q^I}) \rightarrow L_p^{C_q^I} \downarrow_C R_q^I$ caractérisé par la restriction de f à $V_{L_p^{C_q^I}}$ et l'ensemble d'identification vide.

/* m_L est effectivement un morphisme, les éléments de $L_p^{C_q^I}$ ayant par définition une image par f dans R_q^I, la consistance étant assurée par le fait que $C = (f, I)$ caractérise une compatibilité. */

$r_L \leftarrow$ la règle de réécriture $(Aff_I(L_p^{C_q}) \xrightarrow{m_L} L_p^{C_q^I} \downarrow_C R_q^I)$.

$L_p^S \leftarrow r_L\_(id, I)(L_p)$.

/* L_p^S est, aux fusions de nœuds et à un isomorphisme près, L_p privé de sa partie identifiée à des éléments ajoutés lors de l'application de q, soit celle associée à la partie de R_q n'appartenant pas à R_q^I. */

retourner $L_q \uparrow_{(f_q, I)} L_p^S$

/* La partie gauche de la règle composée est la partie gauche de q collée à la fraction de la partie gauche de p qui n'est pas ajoutée lors de l'application de q. */

ALGORITHME 4.4 - *Gauche*, construction de la partie gauche de la règle composée

$m_R \leftarrow$ le morphisme de $L_p^{C_q} \cap L_p^I \rightarrow R_p$ caractérisé par la restriction de f_p aux nœuds de $L_p^{C_q}$ et l'ensemble d'identification vide.
$r_R \leftarrow$ la règle de réécriture $(L_p^{C_q} \xrightarrow{m_L} R_p)$.
retourner $r_R\_(f, I)(R_q)$
/* (f, I) est bien un morphisme de $L_p^{C_q}$ dans R_p, les éléments de $L_p^{C_q}$ ayant par définition une image dans R_p par f, la consistance étant assurée par le fait que C caractérise une compatibilité. La règle r_R est applicable selon ce morphisme grâce aux conditions d'applicabilité 3.(a) et 3.(b). */
/* La partie droite de la règle composée est la partie droite de p collée à la fraction de la partie droite de q non supprimée lors de l'application de p. */

ALGORITHME 4.5 - *Droite*, construction de la partie droite de la règle composée

$P_1 \leftarrow \forall el \in V_{L_r} \cup E_{L_r}, P_1(el) = \forall el_p \in V_{L_p} \cup E_{L_p}, f(el_p) = el \implies el_p \in V_{L_p^I} \cup E_{L_p^I}$
$V_{L_r^I} \leftarrow \emptyset$
pour chaque $v \in V_{L_r}$ **faire**
  /* Par construction de L_r, un de ses nœuds est soit dans L_q, soit dans L_p, soit dans R_q^I (plus précisément dans $L_p^{C_q^I} \downarrow_C R_q^I$), cette dernière partie commune intervenant à la fois dans p et q. */
  si $(v \in L_p \vee v \in L_q)$ **alors**
    /* l'appartenance à L_p ou L_q implique que l'invariance par p et q est équivalente à l'appartenance à L_p^I ou L_q^I respectivement. Le nœud n'intervient alors que dans une règle et il est invariant selon celle-ci. */
    si $(v \in L_p^I \vee v \in L_q^I)$ **alors**
      $V_{L_r^I} \leftarrow \{v\} \cup V_{L_r^I}$
    fin
  sinon
    /* l'appartenance à R_q^I (et donc à $L_p^{C_q^I} \downarrow_C R_q^I$) implique l'invariance selon q par définition de R_q^I. Il est en sus également invariant vis à vis de p si tout nœud dont il est l'image par f est dans $V_{L_p^I}$ */
    si $v \in V_{R_q^I} \wedge P_1(v)$ **alors**
      $V_{L_r^I} \leftarrow \{v\} \cup V_{L_r^I}$
    fin
  fin
fin
retourner $V_{L_r^I}$

ALGORITHME 4.6 - *Inv_V*, construction des nœuds invariants

$E_{L_r^I} \leftarrow \emptyset$
pour chaque $e = (v_1, v_2) \in E_{L_r} \cap (V_{L_r^I})^2$ **faire**
  si $e \in (V_{L_p^I})^2 \vee e \in (V_{L_q^I})^2$ **alors**
    /* si ses deux extrémités appartiennent à $V_{L_p^I}$ ou à $V_{L_q^I}$, l'invariance est équivalente à l'appartenance à $E_{L_p^I}$ ou $E_{L_q^I}$ respectivement. */
    si $e \in E_{L_p^I} \cup E_{L_q^I}$ **alors**
      $E_{L_r^I} \leftarrow \{e\} \cup E_{L_r^I}$
    fin
  fin
  si $e \in (V_{R_q^I})^2$ **alors**
    /* si ses deux extrémités appartiennent à $V_{R_q^I}$, il est au moins l'image d'un arc de E_{L_q} par f_q ou l'image d'un arc de E_{L_p} par f. Pour qu'il soit invariant, il faut que tout arc dont il est l'image par f appartienne à $E_{L_p^I}$ et que tout arc dont il est l'image par f_q appartienne à $E_{L_q^I}$. */
    si $P_1(e) \wedge (\forall e_q \in E_{L_q}, f_q(e_q) = e \implies e_q \in E_{L_q^I})$ **alors**
      $E_{L_r^I} \leftarrow \{e\} \cup E_{L_r^I}$
    fin
  sinon
    /* si une seule extrémité, v_a, est dans $V_{R_q^I}$, l'autre, notée v_b, est soit dans L_q soit dans L_p. Il faut alors, pour que l'arc soit invariant, que tout arc entre v_b et un nœud dont v_a est l'image (par f ou f_q) soit dans la partie invariante de p ou q (soit $E_{L_p^I}$ ou $E_{L_q^I}$) */
    si $(v_1 \in V_{L_q^I} \wedge v_2 \in V_{R_q^I} \wedge (\forall v_q \in V_{L_q^I}, f_q(v_q) = v_2 \wedge (v_1, v_q) \in E_{L_q} \implies (v_1, v_q) \in E_{L_q^I}) \vee$
    $(v_1 \in V_{R_q^I} \wedge v_2 \in V_{L_q^I} \wedge (\forall v_q \in V_{L_q^I}, f_q(v_q) = v_1 \wedge (v_q, v_2) \in E_{L_q} \implies (v_q, v_2) \in E_{L_q^I}) \vee$
    $(v_1 \in V_{L_p^I} \wedge v_2 \in V_{R_q^I} \wedge (\forall v_p \in V_{L_p^I}, f(v_p) = v_2 \wedge (v_1, v_p) \in E_{L_p} \implies (v_1, v_2) \in E_{L_p^I}) \vee$
    $(v_1 \in V_{R_q^I} \wedge v_2 \in V_{L_p^I} \wedge (\forall v_p \in V_{L_p^I}, f(v_p) = v_1 \wedge (v_p, v_2) \in E_{L_p} \implies (v_p, v_2) \in E_{L_p^I})$ **alors**
      $E_{L_r^I} \leftarrow \{e\} \cup E_{L_r^I}$
    fin
  fin
fin

ALGORITHME 4.7 - *Inv_E*, construction des arcs invariants

Notons en particulier qu'il n'existe pas, pour toute application séquentielle, une composition équivalente constructible selon l'algorithme 4.2, du fait de la condition 4. et de la non injectivité de f_q. En effet, ce second fait implique qu'un nœud de L_r peut être la fusion de plusieurs nœuds de L_p. La composée n'est donc pas applicable si ces nœuds ne sont pas identifiés avec un seul nœud du graphe hôte.

Il s'agit à présent de montrer que la règle construite selon l'algorithme 4.2 est conforme à la définition 4.6, et donc que c'est effectivement une règle composée de p et q.

Théorème 4.4. *Pour tout couple de règles* (p, q) *et toute compatibilité* C *vérifiant les conditions d'applicabilité de l'algorithme 4.2,* $r = p \circ_C q$ *est telle que pour tout graphe* G*, s'il existe un morphisme* selon lequel* r *est applicable à* G *:*
si $\forall v_1 \in V_{L_r} \cap V_{L_q}, \forall v_2 \in V_{L_r} \cap V_{L_p}, f_L^R(v_1) \neq f_L^R(v_2)$ [10]

10. Notons que cette hypothèse est nécessairement remplie si f_L^R est injective, en particulier si la recherche de motif se fait à l'aide d'homomorphismes ou d'isomorphismes de sous graphes induits

$f_r \leftarrow$ la fonction de $V_{L_r^I}$ dans V_{R_r} telle que : **pour chaque** $v \in V_{L_r^I}$ **faire**
  si $v \in V_{L_p^I}$ **alors**
    $f_r(v) = f_p(v)$
  fin
  si $v \in V_{L_q^I}$ **alors**
    $f_r(v) = f_q(v)$
  fin
  /* La fonction f_r coïncide avec f_p et f_q sur $V_{L_p^I}$ et $V_{L_q^I}$ respectivement. */
  si $v \in V_{L_p^{C_q^I}} \downarrow_C R_q^I$ **alors**
    $f_r(v) = f_p(v_p)$ avec n'importe quel $v_p \in V_{L_p^{C_q^I}}$ tel que $f(v_p) = v$
  fin
  /* L'unicité de v_p est conditionnée par l'injectivité de f. Toutefois, d'après la condition 3.(b), si v_p n'est pas unique, chaque v_p potentiel a la même image par f_p. La fonction f_r est donc bien définie. */
fin

ALGORITHME 4.8 - *Mapping*, construction de f_r

alors il existe un couple de morphismes (m_L^q, m_L^p) de sorte que* q *est applicable à* G *selon* m_L^q, *que* p *est applicable à* $q\_m_L^q(G)$ *selon* m_L^p *et que* $r\_m_L^r(G) = p\_m_L^p.q\_m_L^q(G)$.

La preuve de ce théorème est détaillée dans l'annexe B.

En particulier, et d'après le théorème 4.3, si p et q sont correctes pour un certain style architectural, une telle règle $r = p \circ_C q$ est alors une règle composée de p et q et correcte vis à vis de ce même style si tout morphisme* selon laquelle elle peut être appliquée vérifie l'hypothèse du théorème 4.4. Notons que cette hypothèse est immédiatement remplie si la recherche de motif se fait à l'aide d'homomorphisme ou d'isomorphisme de sous graphe induit.

4.3.2.3 Illustration

Considérons les règles $\tilde{p}_2$ et $\tilde{p}_3$ de la grammaire augmentée décrivant DIET et définies dans la figure 4.8. Elles décrivent respectivement l'ajout d'un élément temporaire sur un agent et l'instanciation d'un nœud temporaire en un SED. Ces règles peuvent être composées de sorte à produire une règle r décrivant l'ajout d'un SED sur un agent, comme présenté dans la figure 4.10.

Tout d'abord, la partie gauche de $\tilde{p}_3$, $L_{\tilde{p}_3}$ et la partie droite de $\tilde{p}_2$, $R_{\tilde{p}_2}$ contiennent toutes deux un agent et un nœud non terminal. $L_{\tilde{p}_3}$ et $\tilde{p}_2$ sont C-compatibles, avec $C = (f, I)$ où f et I sont représentées en bleu.

Illustration des notations. $L_{\tilde{p}_3}^C$, le sous graphe de $L_{\tilde{p}_3}$ induit par $Dom(f)$ est $L_{\tilde{p}_3}$ privé de v_2^1 et de l'arc (v_2^3, v_2^3).
$R_{\tilde{p}_2}^C$, le sous graphe de $R_{\tilde{p}_2}$ induits par $Im(f)$ est $R_{\tilde{p}_2}$, comme $Im(f) = V_{R_{\tilde{p}_2}}$.
$R_{\tilde{p}_2}^I$, le sous graphe de $R_{\tilde{p}_2}$ dont les éléments sont les images des éléments de $L_{\tilde{p}_2}^I$ par f_q (= Id), est le sous graphe de $R_{\tilde{p}_2}$ comportant uniquement le nœud v_1^1.
$L_{\tilde{p}_3}^{C_{\tilde{p}_2}}$, le sous graphe de $L_{\tilde{p}_3}^C$ dont tous les éléments ont une image dans R_q par f, est en fait $L_{\tilde{p}_3}^C$.

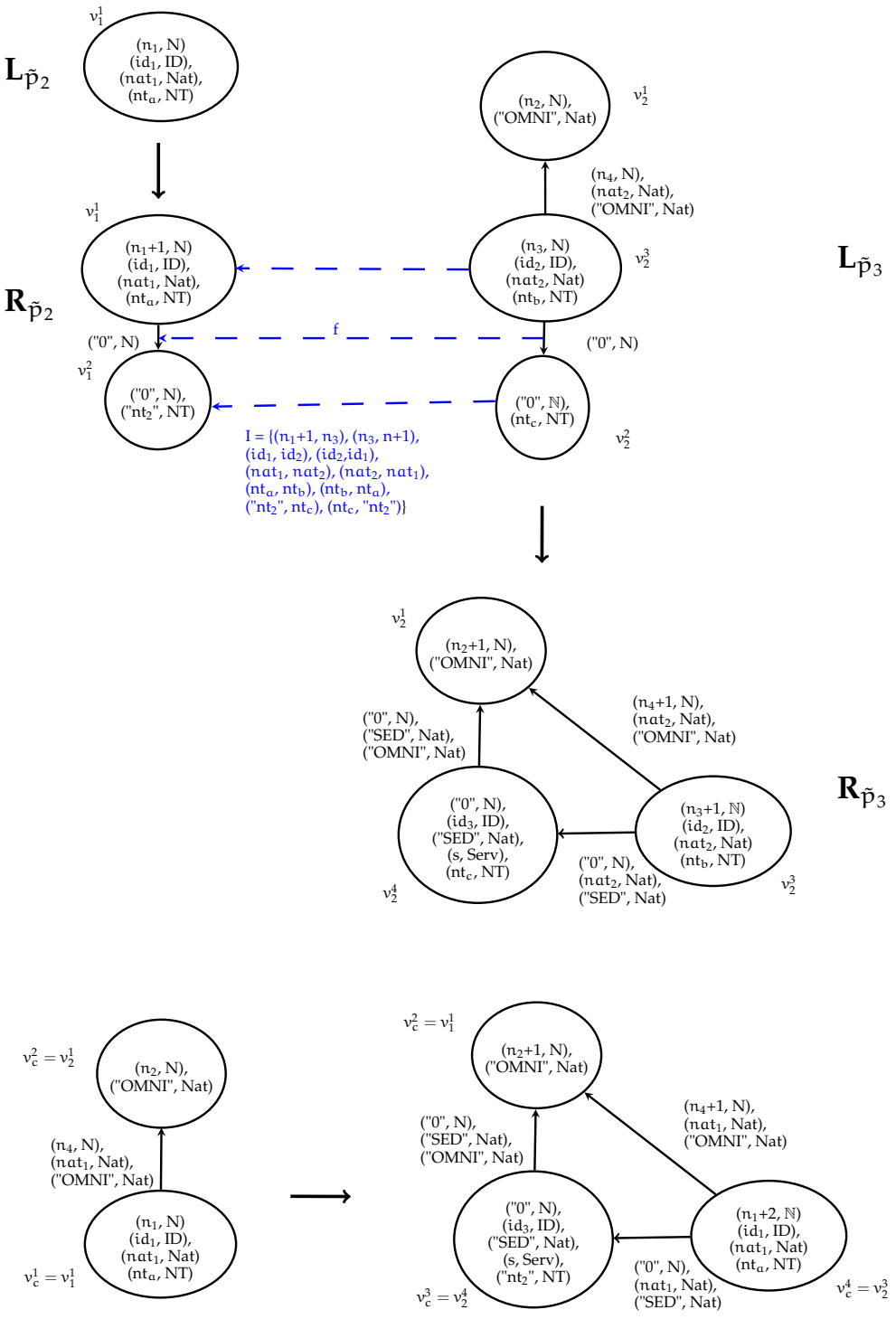


FIGURE 4.10 – Composition : $r = \tilde{p}_3 \circ_C \tilde{p}_2$

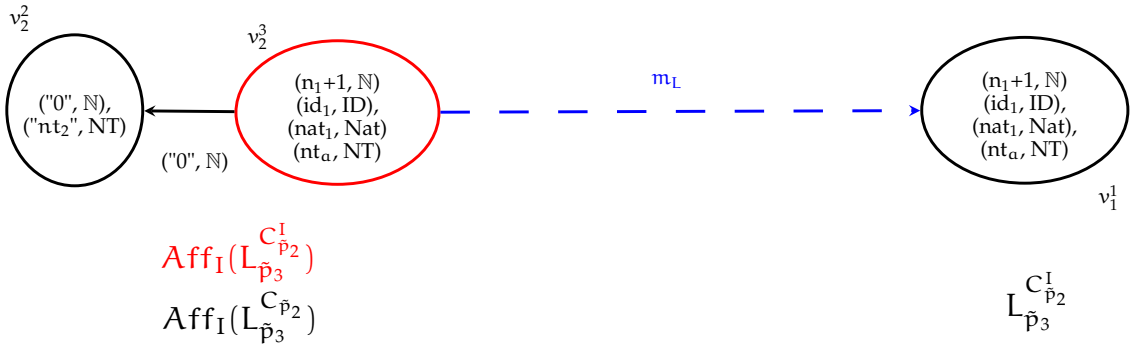


FIGURE 4.11 – Règle $r_L = (Aff_I(L_{\tilde{p}_3}^{C_{\tilde{p}_2}}) \xrightarrow{m_L} L_{\tilde{p}_3}^{C^I_{\tilde{p}_2}} \downarrow_C R^I_{\tilde{p}_2})$

$L_{\tilde{p}_3}^{C^I_{\tilde{p}_2}}$, le sous graphe de $L_{\tilde{p}_3}^{C_{\tilde{p}_2}}$, constitué de tous les éléments ayant une image dans $R^I_{\tilde{p}_2}$ par f, est le sous graphe comportant uniquement le nœud v_2^3.

Applicabilité. La première condition d'application de la composition, "S'il existe un arc de $L_{\tilde{p}_3}$ dont une seule extrémité a une image dans $R_{\tilde{p}_2}$ par f, alors cette image n'est pas ajoutée lors de l'application de $\tilde{p}_2$", est remplie. En effet, les deux extrémités du seul arc de $L_{\tilde{p}_3}$, (v_2^3, v_2^2), ont une image par f dans $R_{\tilde{p}_3}$, respectivement v_1^1 et v_1^2.
La seconde condition d'application, "Pour tout arc de $L_{\tilde{p}_3}$ dont les deux extrémités ont des images dans $R_{\tilde{p}_2}$ par f, si au moins une des deux images est ajoutée lors de l'application de $\tilde{p}_2$, alors il existe un arc entre ces images dans $R_{\tilde{p}_2}$", est également respectée. Le seul couple $(v_1, v_2) \in V^2_{L_{\tilde{p}_3}^C}$ tel que $f((v_1, v_2)) \notin Im(f_{\tilde{p}_2})^2 \wedge ((v_1, v_2) \in E_{L_{\tilde{p}_3}}$ étant (v_2^3, v_2^2), qui vérifie $f((v_2^3, v_2^2)) \in E_{R_{\tilde{p}_2}}$.
La règle $\tilde{p}_3$ n'ajoutant d'arc entre aucun des nœuds qu'elle laisse invariant, la condition 3.(a) est nécessairement remplie, de même que la condition 4. La condition 3.(b) est automatiquement remplie, f étant injective.
Il n'est pas nécessaire que les conditions 5. et 6. soient remplies (la condition 5. ne l'est d'ailleurs pas). En effet, les morphismes liés aux règles à composer sont tous deux injectifs.

Application. Dans ce contexte, le morphisme $m_L : Aff_I(L_{\tilde{p}_3}^{C^I_{\tilde{p}_2}}) \to L_{\tilde{p}_3}^{C^I_{\tilde{p}_2}} \downarrow_C R^I_{\tilde{p}_2}$ est caractérisé par la restriction de f à v_2^2 et l'ensemble d'identification vide. Il a pour ensemble de départ v_2^2 muni de ses attributs intégrant l'affectation induite par I, soit, après un choix arbitraire de représentants favorisant les variables apparaissant dans $R_{\tilde{p}_2}$, $((n1 + 1, \mathbb{N}), (id_1, ID), (nat_1, Nat), (nt_a, NT))$. Son ensemble d'arrivée est le nœud v_1^1.
La règle de réécriture $r_L = (Aff_I(L_{\tilde{p}_3}^{C_{\tilde{p}_2}}) \xrightarrow{m_L} L_{\tilde{p}_3}^{C^I_{\tilde{p}_2}} \downarrow_C R^I_{\tilde{p}_2})$ est représentée dans la figure 4.11.

La règle r_L permet de générer le graphe L_r^S, fruit de son application à $L_{\tilde{p}_3}$ selon (id, I), comme illustré dans la figure 4.12. L_r^S est, à un isomorphisme près, $L_{\tilde{p}_3}$ privé de sa partie identifiée à des éléments ajoutés lors de l'application de $\tilde{p}_2$, soit le nœud temporaire v_2^2 et l'arc (v_2^3, v_2^2).

La partie gauche de la règle composée est représentée dans la figure 4.10. Cette dernière est égale à $L_{\tilde{p}_2} \uparrow_{(f_{\tilde{p}_2}, I)} L_p^S$, qui est à son tour égal à L_p^S, $L_{\tilde{p}_2}$ étant restreint à un seul nœud qui a une image dans L_p^S par $f_{\tilde{p}_2}$ $(= Id)$.

Concernant la partie gauche de la règle composée, le morphisme m_R est ici tel que $L_{\tilde{p}_3}^{C_{\tilde{p}_2}} \cap L_{\tilde{p}_3}^I \to R_{\tilde{p}_3}$. Il est caractérisé par l'ensemble d'identifications vide et la restriction de $f_{\tilde{p}_3}$ à l'intersection de son

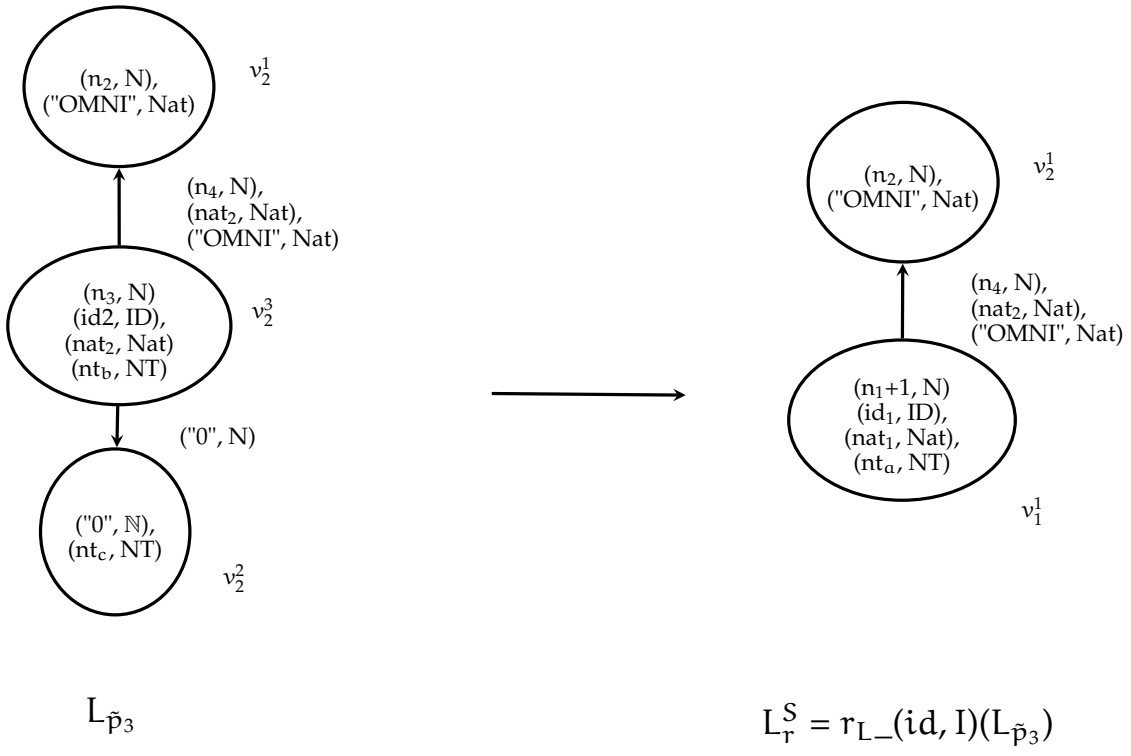


FIGURE 4.12 – Application de r_L à $L_{\tilde{p}_3}$ suivant (id, I) : L^S_r

domaine de définition avec les nœuds de $L^{C_{\tilde{p}_2}}_{\tilde{p}_3}$. Le seul nœud de cette intersection est v^3_2. La règle de réécriture r_L ($L^{C_{\tilde{p}_2}}_{\tilde{p}_3} \xrightarrow{m_L} R_{\tilde{p}_3}$) est présentée dans la figure 4.13.

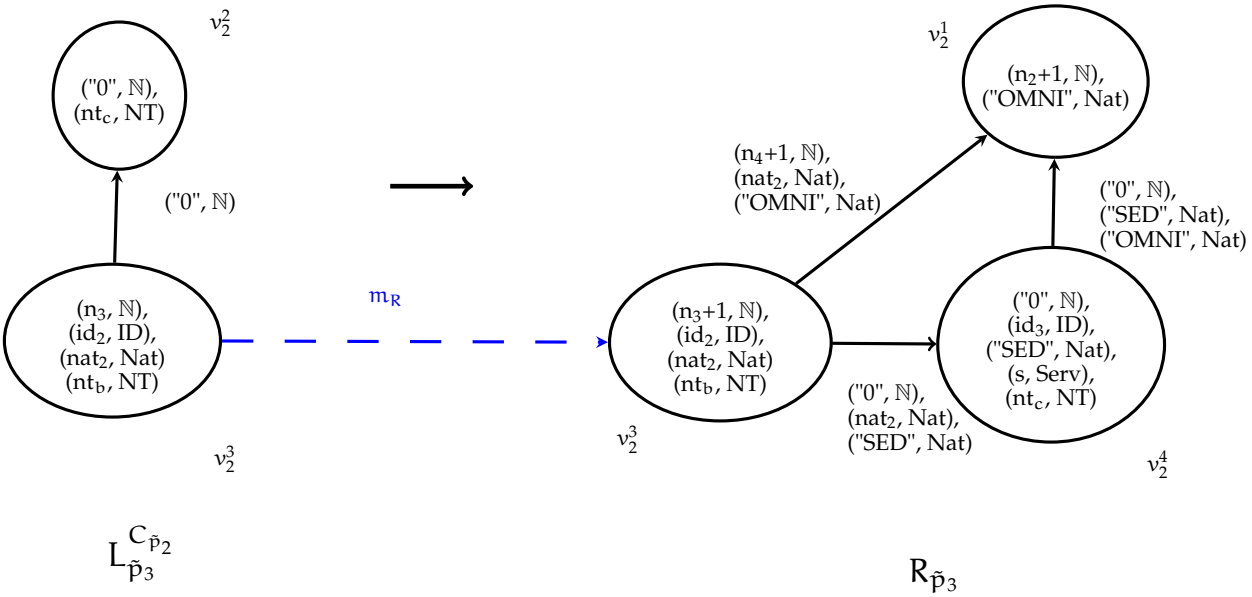


FIGURE 4.13 – Règle r_L :($L^{C_{\tilde{p}_2}}_{\tilde{p}_3} \xrightarrow{m_L} R_{\tilde{p}_3}$)

L'application de la règle r_R entraîne la suppression d'un nœud temporaire ainsi que l'ajout d'un SED et un OMNI. Son application à $R_{\tilde{p}_2}$ selon (f, I) produit la partie droite de la règle composée représentée dans la figure 4.10.

La règle composée, r, ne supprime aucun élément car $L_r = L_r^I$. En effet,
$v_c^2 = v_2^1 \in V_{L_{\tilde{p}_3}^I}$,
$v_c^1 = v_1^1 \in V_{R_{\tilde{p}_2}^I}$, et son unique antécédent par f est $v_2^3 \in L_{\tilde{p}_3}^I$, d'où $P_1(v_c^1)$,
(v_1^1, v_2^1) vérifie P_4, car v_2^3 est l'unique antécédent de v_1^1 par f et $(v_2^3, v_2^1) \in E_{L_{\tilde{p}_3}^I}$.

Finalement f_r est la fonction associant $v_c^2 = v_2^1$ à $f_{\tilde{p}_3}(v_2^1) = v_2^1$ et $v_c^1 = v_1^1$ à v_2^3 comme v_2^3 est l'unique nœud tel que $f(v_2^3) = v_1^1$.

Rappelons que :

1. r est le résultat de $\tilde{p}_3 \circ_C \tilde{p}_2$ et, trivialement, tout morphisme selon lequel elle est applicable vérifie l'hypothèse du théorème 4.4. C'est donc une règle composée d'après le théorème 4.4,
2. $\tilde{p}_3$ et $\tilde{p}_2$ sont correctes pour la grammaire $\tilde{GG}_{DIET}$.

La règle r est par conséquent correcte pour $\tilde{GG}_{DIET}$ selon le théorème 4.5.

4.3.3 Spécialisation

4.3.3.1 Définition et correction.

L'opération de spécialisation consiste à renforcer les conditions d'application d'une règle et/ou de réduire le panel de ses résultats possibles. Cette opération a reçu moins d'attention que l'opération inverse : AGG [27], par exemple, propose de relaxer automatiquement les contraintes d'applicabilité d'une règle si celles ci sont nécessairement satisfaites par la grammaire. La restriction d'une règle est toutefois un outil pratique pour la gestion de systèmes. Elle permet, entre autres, de restreindre l'application d'une règle à un contexte particulier ou à une entité ayant une nature ou un identifiant précis (se rapportant à un composant signalé comme étant défaillant, par exemple).

Une opération de spécialisation est le processus créant une règle dite spécialisé à partir d'une autre. Un dit également que cette première règle est une spécialisation de la seconde. La définition suivante formalise la relation de spécialisation.

Définition 4.7. (**Spécialisation d'une règle**) Une spécialisation d'une règle p est une règle q telle que pour tout graphe G, s'il existe un morphisme* m_L^q selon lequel q est applicable à G, alors il existe un morphisme* m_L^p selon lequel p est applicable à G tel que $q\_m_L^q(G)$ est isomorphe selon $(f_i, \emptyset)$ à $p\_m_L^p(G)$.

Il s'agit à présent de montrer que cette opération conserve la correction des transformations.

Théorème 4.5. *Une spécialisation d'une règle correcte (vis à vis d'un style architectural) est également correcte (vis à vis du même style).*

Démonstration. Soit G un graphe représentant une configuration correcte d'un style architectural S caractérisé par $Gr = (AX, NT, T, P)$, p une règle de réécriture correcte pour S et q une spécialisation de p.

Supposons que q est applicable à G selon un morphisme* m_q et prouvons que le résultat de son application est nécessairement correct. Selon la définition 4.7, il existe un morphisme* m_p par rapport auquel p est applicable à G tel que $q\_m_L^q(G)$ est isomorphe à $p\_m_L^p(G)$ selon un certain isomorphisme $(f, \emptyset)$.

p et G étant corrects par hypothèse, $p\_m_p(G)$ est également une configuration correcte. Il existe donc par définition un graphe G_c pouvant être obtenu à partir de AX en appliquant des règles de P et un isomorphisme (f_i, I_i) entre $p\_m_p(G)$ et G_c tel que $Aff_{I_i}(p\_m_2(G)) = p\_m_p(G)$.

Donc il existe deux isomorphismes, $q\_m_q(G) \xrightarrow{(f,\emptyset)} p\_m_p(G) = Aff_{I_i}(p\_m_p(G)) \xrightarrow{(f_i,I_i)} G_c$. D'après la remarque 3.1, il existe un isomorphisme $q\_m_q(G) \xrightarrow{(f\circ f_i,I_i)} G_c$. Comme $q\_m_q(G) \xrightarrow{(f,\emptyset)} Aff_{I_i}(p\_m_p(G))$, $q\_m_q(G)$ est nécessairement invariant pour Aff_{I_i}.

Par conséquent $q\_m_q(G)$, est également une configuration correcte pour S.

□

4.3.3.2 Concrétisation.

Le théorème suivant décrit un ensemble de contraintes sur deux règles de réécriture permettant la vérification automatique de la satisfaction de la relation de spécialisation. Cette vérification peut appuyer la création d'une règle spécialisée ou servir à prouver automatiquement la correction d'une règle.

Théorème 4.6. *Pour toutes règles* $p = (L_p \xrightarrow{m_p} R_p)$ *et* $q = (L_q \xrightarrow{m_q} R_q)$*, avec* $m_p = (f_p, I_p)$ *et* $m_q = (f_q, I_q)$ *des morphismes de* L_p^I *et* L_q^I *vers* R_p *et* R_q*, si :*

1. *Il existe un morphisme** $m_L = (f_L, I_L) : L_p \xrightarrow{m_L} L_q$, f_L *pouvant être la fonction identité, tel que :*
   (a) L_q *est invariant pour l'affectation induite par* I_L, Aff_{I_L}. *Ainsi,* L_q *a nécessairement autant ou moins de variables libres que* L_p.
   (b) *la restriction de* m_L *à* L_p^I *est un morphisme* de* L_p^I *vers* L_q^I.
   (c) $(V_{L_q} \backslash V_{L_q^I}) = f_L(V_{L_p} \backslash V_{L_p^I})$. *Les nœuds effacés par* q *sont exactement ceux supprimés par* p.
   (d) $f_L(E_{L_p} \backslash E_{L_p^I}) \subseteq (E_{L_q} \backslash E_{L_q^I})$. *Tout arc explicitement effacé par* p *est effacé par* q. *Comme tout nœud effacé par* p *l'est également par* q*, les arcs suspendus implicitement effacés par* p *sont également effacés par* q.
   (e) $\forall e = (v_1, v_2) \in (E_{L_q} \backslash E_{L_q^I}), e \in f_L(E_{L_p} \backslash E_{L_p^I}) \vee (v_1 \in (V_{L_q} \backslash V_{L_q^I}) \vee v_2 \in (V_{L_q} \backslash V_{L_q^I}))$. *Tout arc explicitement supprimé par* q *est soit explicitement supprimé par* p*, soit supprimé implicitement car devenant suspendu lors de la transformation (et donc également supprimé par* p*).*
2. *Il existe un homomorphisme* $h_R = (f_R, I_R) : R_p \xrightarrow{h_R} R_q$ *tel que :*
   (a) R_q *est invariant pour l'affectation induite par* I_R, Aff_{I_R}. *Ainsi,* L_q *a nécessairement autant ou moins de variables libres que* R_p.
   (b) $\forall v \in Dom(f_p), f_R(f_p(v)) = f_q(f_L(v))$.
   (c) $f_R(R_p \backslash f_p(L_p^I)) = R_q \backslash f_q(L_q^I)$. *L'image par* f_R *de l'ensemble des éléments de* R_p *non identifiés avec un élément de* L_p^I *à travers* f_p *est égal à l'ensemble des éléments de* R_q *non identifiés avec un élément de* L_q^I *via* f_q. *Cela signifie que les éléments ajoutés par* p *sont exactement les éléments ajoutés par* q *(à* f_R *près).*
3. f_q *est injective sur* $Dom(f_q) \backslash (f_L(Dom(f_p)))$. *Ainsi,* q *n'implique pas plus de fusion d'éléments que* p.

alors q *est une spécialisation de* p.

La preuve de ce théorème est détaillée dans l'annexe B.

4.3.3.3 Illustration

Considérons la règle $r = \tilde{p}_3 \circ_C \tilde{p}_2$ illustrée dans la figure 4.10 et dont la construction est décrite dans la sous section 4.3.2.3. Celle-ci spécifie une transformation permettant l'ajout d'un SED sur un agent. Cette règle peut être spécialisée afin de caractériser la duplication d'un SED fournissant un ensemble de services fixé sous la forme de la règle d représentée dans la figure 4.14. Cette dernière permet, par exemple, de palier à une augmentation des demandes clients pour un service particulier, ou d'anticiper une panne potentielle d'un SED en mitigeant ses conséquences.

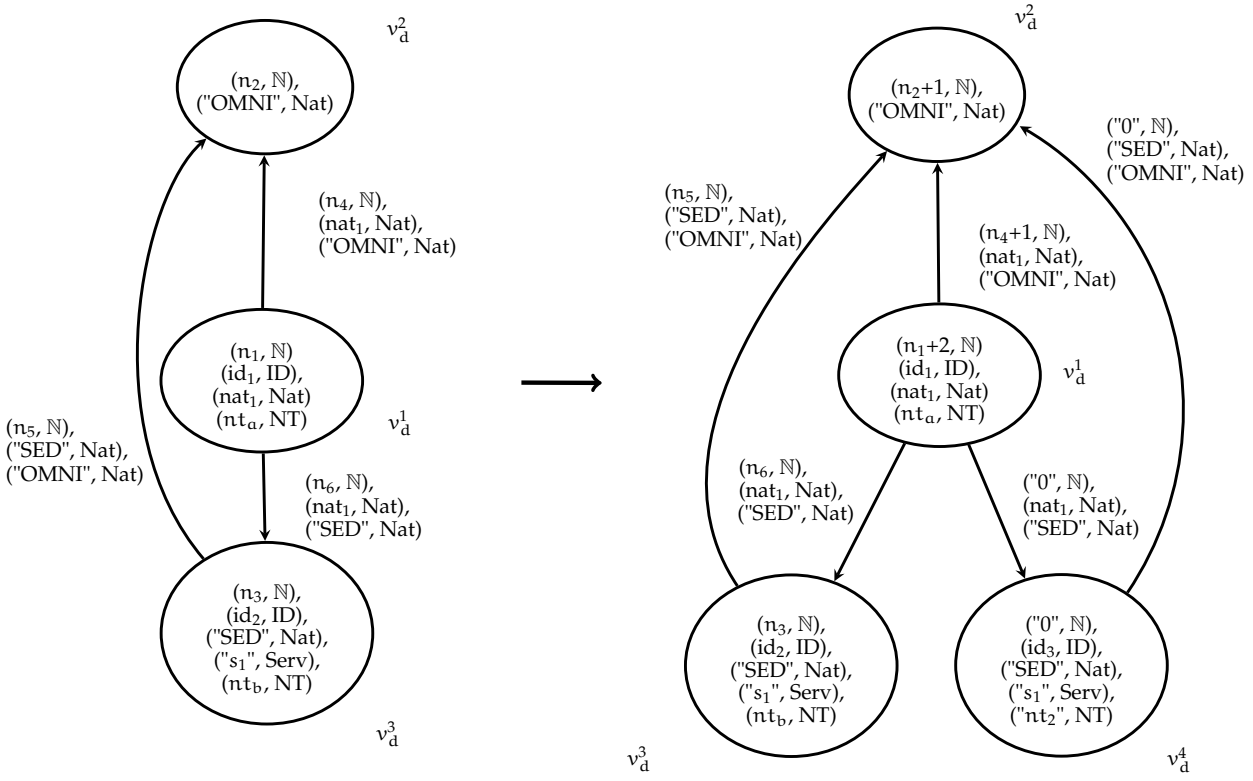


FIGURE 4.14 – d, une spécialisation de r :
duplication d'un SED fournissant un ensemble de services fixé

Les conditions d'applications de d sont plus contraignantes que celles de r : L_d est construit à partir de L_r en ajoutant le SED à dupliquer ainsi que les liaisons le concernant. La relation entre L_r et L_d, et en particulier le morphisme les liant, est illustré dans la figure 4.15. La partie gauche de r étant un sous graphe de la partie gauche de d, les conditions 1.(a) et 1.(b) du théorème 4.6 sont respectées. De plus, comme ni p ni r ne suppriment d'élément, ces règles remplissent également les conditions 1.(c), (d) et (e).

Le résultat de l'application de d est équivalente à celle de r : R_d est construit à partir de R_r de la manière suivante. Premièrement, l'enrichissement de la partie invariante, consistant en l'ajout du SED à dupliquer et des liaisons le concernant, est appliqué à R_r afin de donner R_d. Ceci valide la condition 2. (c) du théorème 4.6. Deuxièmement, le nombre de variables libres, en respectant ainsi la condition 2. (b), est diminué ; l'ensemble des services du SED dupliqué est assigné au SED nouvellement déployé. La relation entre R_r et R_d, et en particulier le morphisme les liant (voir condition 2. (a)), est illustré dans la figure 4.16.

Rappelons que :

1. comme d respecte les conditions du théorème 4.6, c'est une spécialisation de r.

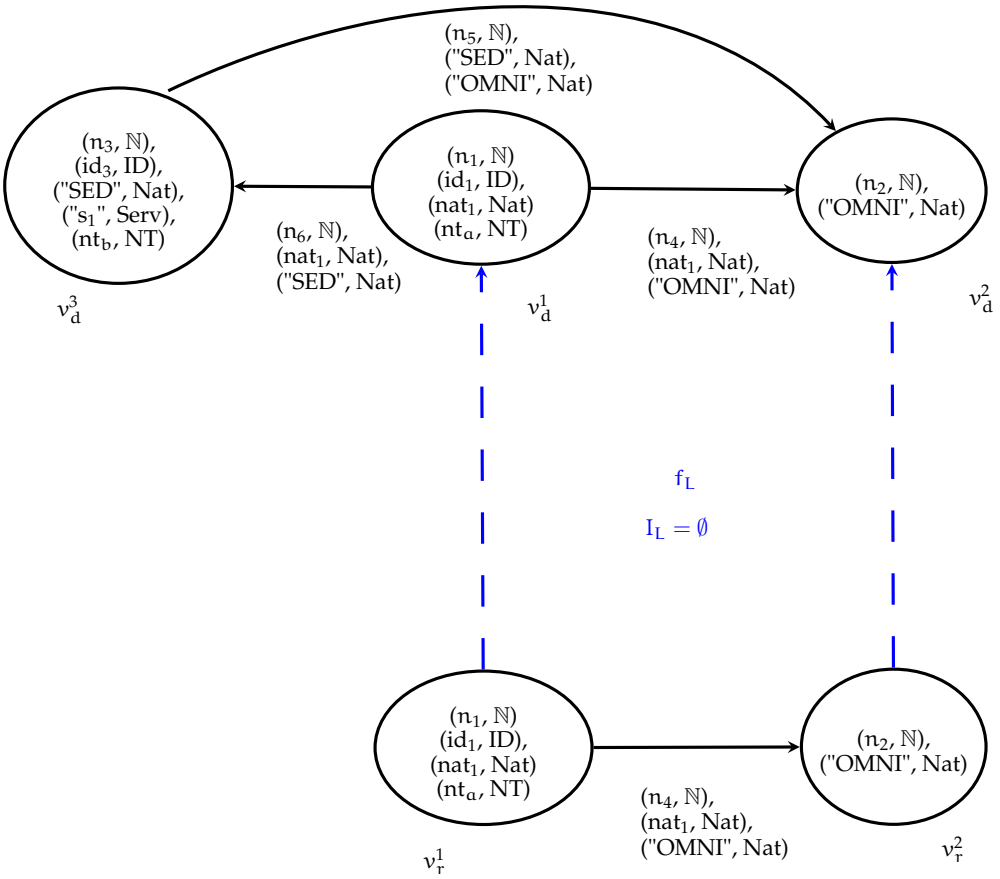


FIGURE 4.15 – d, une spécialisation de r :
relation entre parties gauches

2. r est correcte vis à vis de $\tilde{GG}_{DIET}$ (voir sous section 4.3.2.3).

La règle d est donc correcte pour $\tilde{GG}_{DIET}$ selon le théorème 4.5

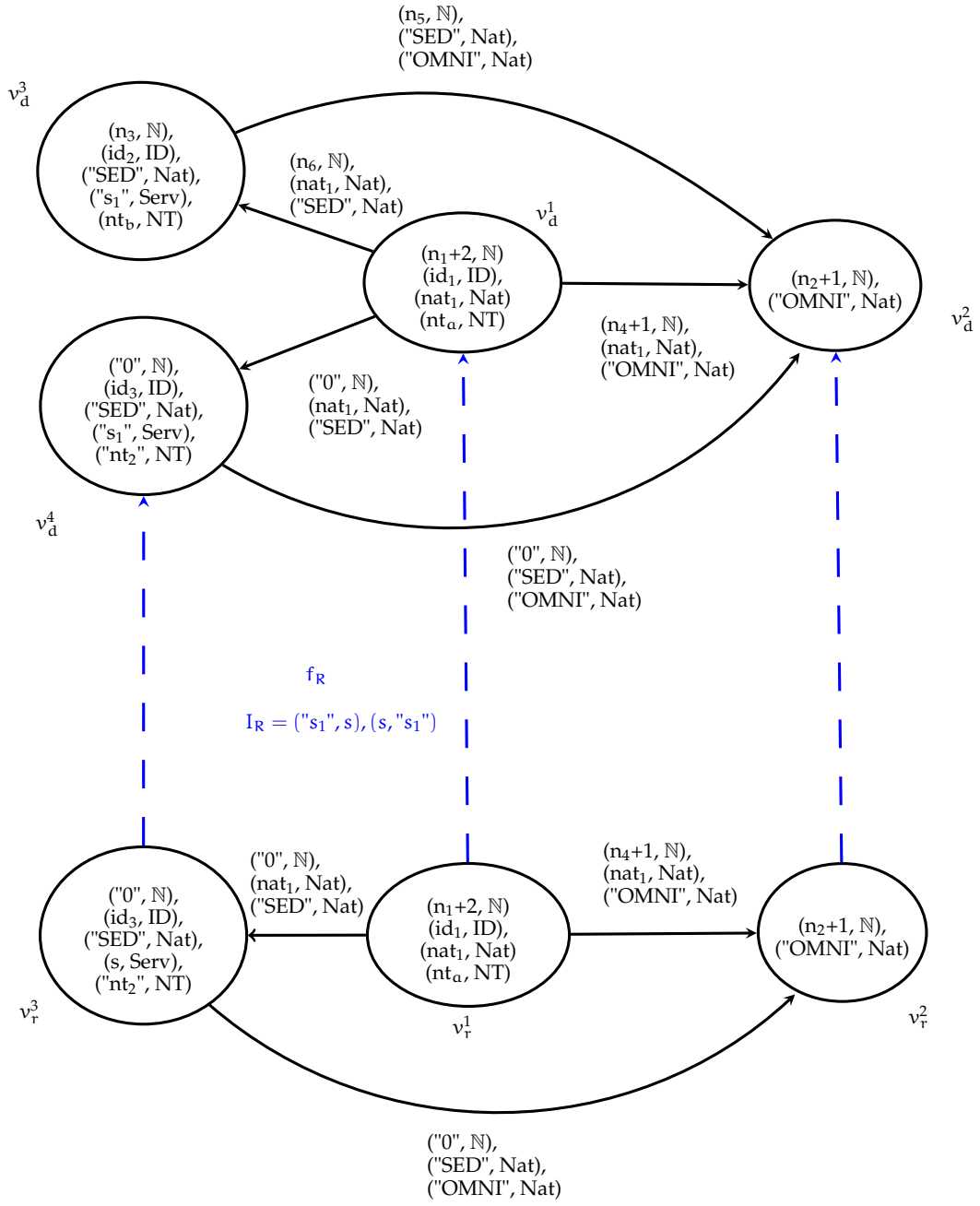


FIGURE 4.16 – d, une spécialisation de r : relation entre parties droites

Remarque 4.4. Il est intéressant de noter que l'application d'une spécialisation incrémente uniquement les compteur de matching des éléments apparaissant également dans la règle originale. Dans l'exemple développé, le compteur de v_d^3 n'est ainsi pas incrémenté. Ce comportement est normal et cohérent avec l'équivalence des applications d'une règle et de sa spécialisation.

4.4 Conclusion du chapitre

Dans ce chapitre, nous élaborons une **méthodologie permettant la construction et la caractérisation générative de règles de transformations *correctes par construction* dont l'application préserve nécessairement la correction du système**.

Dans un premier temps, nous définissons les diverses notions de correction considérées dans ces travaux, comme récapitulé dans le tableau 4.1. Les grammaires de graphes sont adoptées en tant que méta-modèle (*model kind*) pour la formalisation de styles architecturaux dont la spécification fournit la base axiomatique de la notion de correction ; la validité de cette caractérisation n'est elle même pas questionnée. Une configuration est considérée correcte vis à vis d'un style si elle en est une instance. Une transformation est à son tour considérée correcte vis à vis d'un style si elle le préserve, c'est à dire que son application à une configuration correcte produit *nécessairement* une configuration correcte.

Concept	**Formalisation**	**Signification**
Style architectural	Grammaire de graphe	Caractérisation d'un ensemble de graphes
Correction : Configuration	Instance de la grammaire	Instance du style architectural
Transformation	Transformation pour laquelle l'ensemble des instances de la grammaire est stable	Transformation préservant la correction du système

TABLE 4.1 – Récapitulatif : style architectural et correction

La méthodologie proposée se base sur l'aspect génératif du formalisme adopté pour la description d'un style architectural. En effet, la spécification d'un style fournit un ensemble initial de transformations axiomatiquement correctes (ses productions). L'introduction d'opérations préservant la correction des règles permet donc de générer, à partir de cet ensemble initial, de nouvelles règles correctes.

Dans ce but, nous nous intéressons à **trois opérations sur les règles de réécriture** et **nous montrons qu'elles préservent leur correction** sous certaines conditions que nous étudions. Nous proposons également des **algorithmes et méthodes opérationnelles facilitant leur emploi**.
La première, l'inversion, consiste à définir une transformation inverse annulant les effets d'une autre. C'est également la seule opération ne conservant la correction d'une règle que sous certaines conditions. Après avoir adapté sa définition opérationnelle au contexte de ces travaux, nous présentons ces conditions. Finalement, nous les relaxons en présentant un algorithme implémentant un compteur de reconnaissance.
La seconde opération étudiée est une composition dans son sens classique. Après avoir démontré qu'elle conserve la correction, nous proposons un algorithme opérationnel permettant la construction d'une règle composée.
Finalement, la spécialisation permet de restreindre les conditions d'applicabilité d'une règle de réécriture et/ou de ses résultats possibles. Après avoir démontré que la spécialisation d'une règle correcte est également correcte, nous proposons une manière concrète de vérifier automatiquement la satisfaction de la relation de spécialisation.

La méthodologie élaborée ici permet ainsi la caractérisation générative et la création de règles de transformations correctes par construction pour un style architectural, étant donné une grammaire de graphe le décrivant. L'utilisation de telles transformations dans le cadre de politiques autonomes permet d'**efficacement garantir le maintien d'un système dans un état correct** lors de leur application, et donc son auto-protection interne. En effet, cette technique présente plusieurs avantages.

Premièrement, nulle vérification ou validation n'est nécessaire pour garantir le maintien de la cohérence vis à vis d'une évolution. Les processus de validation et vérification sont ainsi transposés de la phase d'exécution à la phase de conception.
Deuxièmement, une transformation correcte ne pouvant par définition produire un état incorrect, la possibilité d'avoir à recourir à des procédures d'annulation ("roll-backs") suite à une transformation est complètement supprimée.

Chapitre 5

Intégration dans une approche multi-modèles

Application à l'auto-guérison de systèmes Machine-à-Machine

5.1 Introduction

La propriété d'auto-guérison [1] permet à un système de gérer pannes et dysfonctionnements de ses composants, qu'ils soient logiciels ou matériels. Dans un environnement distribué comportant de multiples machines, la probabilité d'occurrence de tels troubles ne peut être négligée. Il est primordial que le système puisse les surmonter et maintenir son activité. Idéalement, ces dysfonctionnements devraient être anticipés et ainsi évités : ce comportement est qualifié d'auto-guérison pro-active.

Nous concevons dans la suite un gestionnaire autonome basé sur le formalisme précédemment introduit et implémentant, entre autre, cette propriété. Ce gestionnaire, développé dans le cadre du projet OM2M [2], porte sur tout système Machine-à-Machine (M2M) se conformant au standard ETSI M2M [125].

Au cours des dernières années, la prolifération des appareils et réseaux de communication sans fil a conduit à l'émergence du paradigme M2M, une des solutions les plus prometteuses pour le futur des communications intelligentes [126]. Ce paradigme s'applique aux systèmes comprenant un grand nombre d'appareils intelligents partageant des informations et élaborant collectivement des décisions, et ce avec peu ou prou d'intervention humaine. La section 5.2 détaille le contexte des travaux présentés dans ce chapitre en fournissant un tour d'horizon des solutions actuelles pour les systèmes M2M et en introduisant le projet OM2M.

Dans un premier temps, il convient d'élaborer la vue architecturale sur laquelle s'appuie le gestionnaire. Afin de concilier fonctionnalités et gestion autonome du système, nous proposons dans la section 5.3 une vue composée de deux modèles distincts : représentation du standard ETSI et graphes. Les mécanismes répondant aux impératifs de cohérence dans les approches multi-modèles y sont également introduits.

1. en anglais : *self-healing*
2. *Open Machine to Machine*, eclipse.org/om2m/

Cette représentation a permis la conception de plusieurs politiques de gestion, dont un échantillon relatif à l'auto-guérison pro-active du système est présenté dans la section 5.4. Les politiques détaillées incluent des éléments d'auto-optimisation. Ayant été élaborées conformément à la méthodologie introduite dans le chapitre précédent, elles respectent également la propriété d'auto-protection interne.

Enfin, la section 5.5 démontre la faisabilité de l'approche proposée à des applications concrètes. Se reposant sur un premier effort d'implémentation, une politique autonome est menée et évaluée sur un cas d'utilisation de compteur intelligent.

5.2 Contexte : Le paradigme Machine-à-Machine

5.2.1 Le paradigme Machine-à-Machine

Le paradigme M2M couvre une grande variété de systèmes ; il peut être utilisé afin de décrire toute technologie permettant à des appareils de communiquer. Son but sous-jacent est la mise en réseau d'appareils interconnectés et contrôlés à distance à l'aide de technologies fiables, bon marché et extensibles à de grandes échelles. Les systèmes et communications M2M ont été exploités avec succès pour obtenir un gain d'efficacité et réduire (ou même supprimer) les interventions humaines dans de nombreux contextes. Ils interviennent par exemple dans la spécification et l'implémentation d'applications intervenant dans les automobiles, les compteurs intelligents, la domotique et les réseaux de distributions d'électricité intelligents. Intrinsèquement, les systèmes M2M sont sujets à évolution, des applications étant démarrées et interrompues, des machines découvertes et arrêtées... Ils se doivent par conséquent d'être autonomes.

Les systèmes M2M possèdent des caractéristiques et contraintes communes [126], telles que l'hétérogénéité du réseau et l'intelligence des appareils. Les données produites et manipulées évoluent au cours du temps, sont mobiles et sujettes à des contraintes temporelles (accès rapide...). Néanmoins, les solutions techniques pour le M2M sont nombreuses, extrêmement fragmentées et généralement dédiées à une seule application. Le développement du marché M2M en est sensiblement affecté et freiné, tandis que les coûts de développement, de maintenance et de recherche augmentent. Face à ces similarités et contraintes, la standardisation s'impose comme le moteur clé de la levée des verrous techniques et de l'interopérabilité des services et réseaux de M2M.

Un grand nombre d'organismes de standardisation, tels que TIA [3], ESNA [4] et IPSO [5], tentent de définir des architectures et services standards répondant aux problématiques de communication M2M. Ceux-ci adoptent deux types d'approches :

— en étendant des standards existants pour répondre à ces nouveaux besoins, tels que les protocoles internet (IP) *Internet Protocol* pour les objets connectés intelligents.
— en créant des standards de toute pièce, comme le fait l'OMA [6].

Cependant, l'offre actuelle demeure insatisfaisante. Premièrement, de multiples groupes se restreignent à certaines des propriétés M2M et n'ont pas pour objectif de couvrir leur ensemble. L'OMA, par exemple, s'intéresse uniquement à la gestion des appareils. Deuxièmement, la plupart de ces propositions ne sont pas abouties mais plutôt en phase d'ébauche préliminaire. Les travaux

3. *the Telecommunications Industry Association*, tiaonline.org
4. *Energy Services Network Association*, esna.org
5. *IP-for the connection of smart objects*, ipso-alliance.org
6. *Open Mobile Alliance*, http ://openmobilealliance.org/about-oma/work-program/m2m-enablers/

liés aux M2M et conduits par l'IETF [7] et la TIA sont par exemple à ranger dans cette dernière catégorie.

5.2.2 Le standard ETSI M2M et le projet OM2M

L'ETSI [8], et plus particulièrement son groupe smartM2M [9], est un des seuls à avoir développé une vue de bout en bout mature pour les communications M2M. Cette dernière facilite le déploiement d'applications verticales et l'innovation inter-industries en exposant données et services au travers d'interfaces standardisées. Ces interfaces confèrent également une grande extensibilité aux systèmes considérés en implémentant des échanges de données conformes au style architectural REST [10]. L'architecture de ce standard est définie au travers d'une multitude de rapports et spécifications techniques [11] précisant entre autres :

— les contraintes comportementales et fonctionnelles de chaque élément du réseau pour fournir une vue de bout en bout [127].
— l'architecture fonctionnelle considérée, composée de capacités de services M2M [128].
— les protocoles des diverses interfaces [129].

C'est dans ce contexte que le projet OM2M [130] [12] a vu le jour. Initié par le LAAS puis diffusé par la fondation Eclipse, il vise à fournir une implémentation open-source de ce standard prometteur. Les contributions présentées dans la suite de ce chapitre s'inscrivent dans ce cadre. Elles constituent plus spécifiquement la conception et un premier effort d'implémentation d'un gestionnaire autonome pour tout système se conformant au standard ETSI M2M. En effet, eût égard au contexte extrêmement changeant dans lequel ceux-ci évoluent, le standard prévoit une ressource "gestionnaire" chargée de mener à bien adaptations et évolutions. Il ne fournit néanmoins aucune précision à ce sujet, notamment sur la manière de le concevoir et de l'implémenter.

Il convient avant tout de souligner que la représentation sous-jacente au standard n'est pas appropriée à la spécification d'opérations d'auto-gestion. En effet, elle se concentre sur les fonctionnalités du système, et n'offre par conséquent que peu de moyens de raisonnement et de modélisation liés à ses transformations. Des approches multi-modèles [21, 59, 131] peuvent être adoptées lorsqu'une seule description ne peut répondre à l'intégralité des besoins. Afin de permettre l'auto-gestion de systèmes M2M, les concepts du standard ETSI M2M peuvent ainsi être associés à un modèle approprié, le résultat bénéficiant de chacune des deux représentations.

5.3 Une approche multi-modèles conciliant fonctionalités et gestion autonome

Cette section est dédiée à la solution de modélisation élaborée dans le cadre de l'approche proposée. Afin de concilier fonctionnalités et gestion autonome du système, nous proposons une vue architecturale composée de deux modèles distincts, basés sur deux différents méta-modèles. Le premier, s'intéressant aux aspects fonctionnels du système, est une traduction directe du standard

7. *The Internet Engineering Task Force*, ietf.org
8. *The European Telecommunications Standards Institute*, etsi.org
9. etsi.org/index.php/technologies-clusters/technologies/m2m
10. Representational state transfer
11. Ces rapports sont disponibles à l'adresse : etsi.org/technologies-clusters/technologies/m2m
12. *Open M2M*, eclipse.org/om2m/

ETSI M2M. Le second, se reposant sur une meta-grammaire de graphe, permet d'assurer la gestion autonome du système. Un modèle et les mécanismes lui étant liés seront qualifiés de "couche".

Dans le cas d'une vue architecturale comportant plusieurs modèles, et comme le précise le standard ISO/IEC/IEEE-42010 [12], deux points doivent être pris en considération.

Premièrement, le but n'est pas de dupliquer l'intégralité des informations disponibles dans les deux modèles. Seules celles nécessaires à l'accomplissement des objectifs d'un modèle doivent y figurer.

Deuxièmement, la multiplicité des modèles pose le problème de leur cohérence. Le système étudié étant dynamique, chaque modèle doit évoluer pour correspondre à son état effectif. De plus, des évolutions peuvent être initiées à travers les processus de découvertes et de reconfigurations, et donc trouver leur origine dans chacune des deux couches du modèle. Pour assurer la cohérence interne de la vue architecturale proposée, et donc des deux modèles qui la composent, ces transformations doivent être impactées de manière bidirectionnelle. Nous élaborons en ce but un ensemble de communications et mises à jour inter-couches.

5.3.1 Modèle fonctionnel : le standard ETSI M2M

Dans la vision de l'ETSI, les composants de systèmes M2M sont considérés comme des ressources. Ils sont représentés dans une structure arborescente manipulable de manière RESTful, c'est à dire conformément au style architectural REST. Cet arbre de ressources composé de divers groupes logiques permet d'assurer un adressage simple des ressources, une grande flexibilité et extensibilité dans les échanges de données applicatives ainsi que des interfaces aisément exploitables.

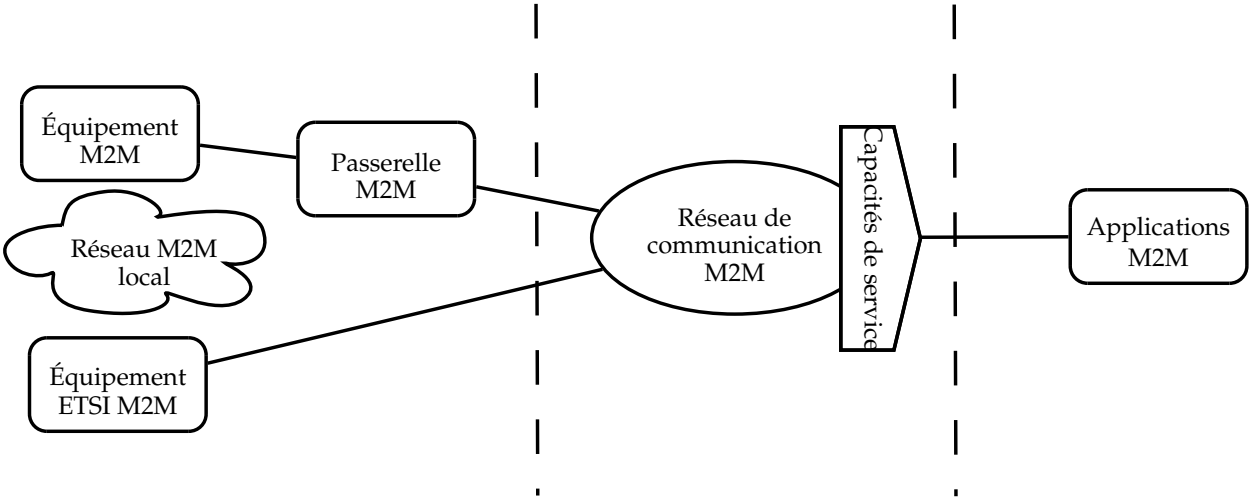


FIGURE 5.1 – Architecture logique en domaines du standard ETSI M2M

L'ETSI a divisé les systèmes M2M en trois domaines illustrés dans la figure 5.1 :

— Le *domaine applicatif*[13] appliquant la logique de service. Les données relatives aux applications sont considérées comme des ressources définies dans une structure arborescente et manipulables de manière RESTful. Cet arbre de ressources expose fonctions et services à travers un ensemble d'interfaces ouvertes regroupées sous le terme "capacités de service".

13. en anglais : *application domain*

— Le *domaine des équipements M2M* [14] inclus toutes les sources et destinations de données tels que capteurs, senseurs intelligents, microprocesseurs, passerelles... Ce sont, en résumé, les machines communicantes du système. Selon le standard ETSI M2M, il existe trois types d'équipement M2M : passerelles, appareils dits ETSI pouvant communiquer sur le réseau sans passer par une passerelle et appareils contraints devant se connecter à une passerelle pour communiquer. Ces derniers sont traités comme des ressources de la passerelle sur laquelle ils se connectent.
— Le *domaine réseau* [15] désigne une technologie permettant l'inter-connectivité et la mise en réseau des appareils du domaine des équipements M2M.

Les capacités de services sont subdivisées en :
— *NSCL* [16] faisant référence aux capacités de service relatives au domaine réseau.
— *GSCL* [17] faisant référence aux capacités de service des passerelles M2M.
— *DSCL* [18] faisant référence aux capacités de services des équipements et appareils M2M.
— *SCL* [19], faisant référence à n'importe lequel des termes précédents (NSCL, GSCL et DSCL).

Dans les systèmes M2M, les données proviennent d'un grand nombre d'appareils distincts. Elles sont échangées entre diverses applications au moyen de conteneurs de données. Ces derniers sont des médiateurs/tampons stockant les données et servant d'intermédiaire lors de leur utilisation. Ils suppriment la nécessité de connections directes et autorisent l'échange de données entre des entités n'étant pas connectées simultanément.

Les procédures d'enregistrement et d'annonce permettent les interactions entre les composants (équipements et applications) distribués du système en gérant la visibilité de leurs ressources.

L'*enregistrement* comprend deux sous-cas :
— L'*enregistrement de SCLs* définit l'ensemble des procédures permettant à un GSCL ou un DSCL de s'enregistrer sur un autre SCL. Après la découverte de l'équipement concerné, celui-ci doit s'enregistrer sur le réseau (NSCL) afin d'initier la gestion de ses ressources. Les enregistrements pairs à pairs (i.e., équipement/équipement, passerelle/passerelle ou passerelle/équipement) permettent aux équipements de communiquer directement (i.e. sans passer par le domaine réseau) lorsque cela est possible, améliorant ainsi l'extensibilité du système.
— L'*enregistrement des applications* définit l'ensemble des procédures permettant à une application de se déclarer localement afin d'être prise en compte dans son SCL local. Cette étape est requise pour que la dite application soit connue et visible, et qu'elle puisse ainsi échanger des données en usant des processus RESTful.

L'*annonce* définit l'ensemble des procédures permettant à une ressource, c'est à dire une application ou un conteneur, de se déclarer sur un SCL distant. Afin d'annoncer une ressource à une machine distante, sa machine hôte et la machine cible doivent pouvoir se voir, i.e. être enregistrées l'une à l'autre.

La représentation d'une configuration selon le modèle ESTI peut être symbolisée sur la base de l'arbre de ressource du réseau, celui-ci voyant les SCLs de chaque équipement du système. Les applications et conteneurs non-annoncés au réseau sont stipulés dans cette représentation bien

14. en anglais : *M2M device domain*
15. en anglais : *network domain*
16. Network Service Capabilities Layer
17. Gateway Service Capabilities Layer
18. Device Service Capabilities Layer
19. Service Capabilities Layer

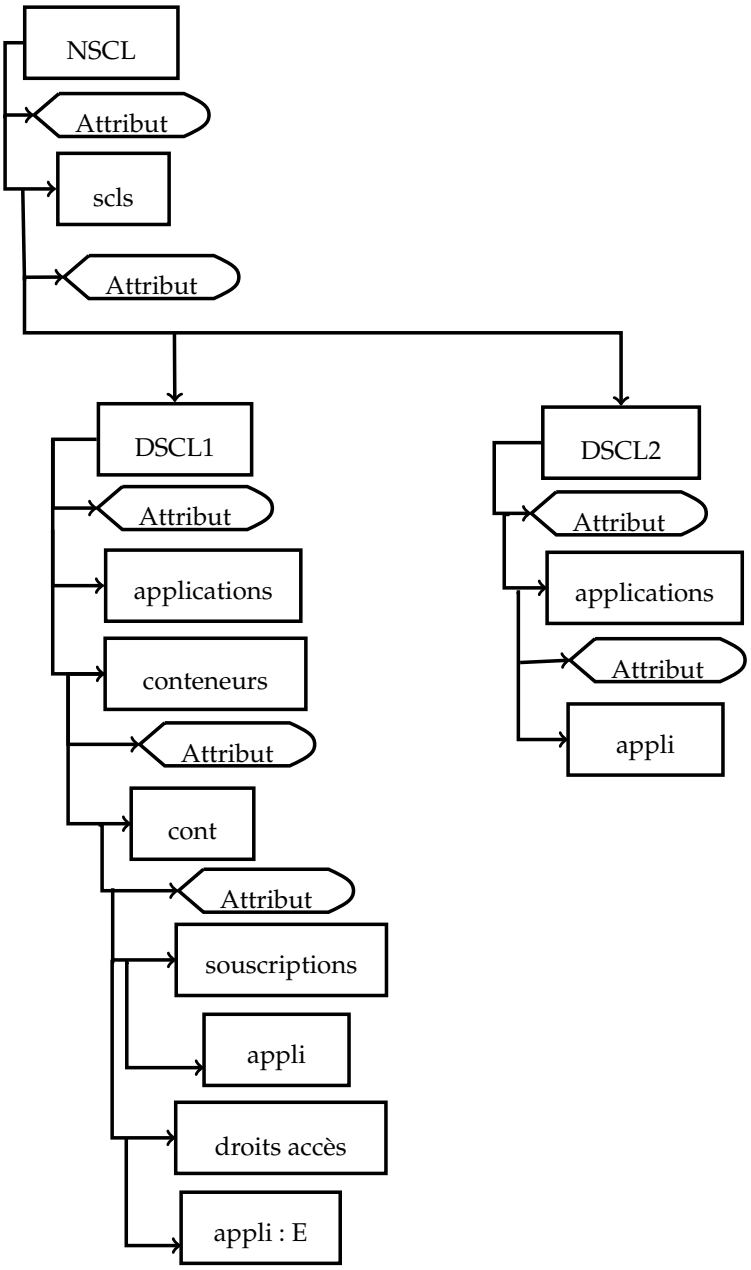


FIGURE 5.2 – Représentation d'une configuration M2M d'après le modèle ETSI

qu'absents du NSCL. La figure 5.2 illustre cette représentation pour une configuration simple comprenant deux équipements dont les SCLs sont notés DSCL1 et DSCL2. Un conteneur et une application, notés `cont` et `appli`, sont déployés sur le premier et le second appareil, respectivement. Notons que cette représentation ne permet pas, *a priori*, de savoir si les deux équipements sont enregistrés l'un à l'autre ou si certaines de leurs ressources sont annoncées. Ces informations ne sont disponibles qu'en connaissant les SCLs des deux appareils. Dans ce cas particulier, `appli` a souscrit à, et écrit sur, le conteneur `cont`. Les appareils sont donc enregistrés et `cont` est annoncé au deuxième équipement.

5.3.2 Modèle formel pour la gestion autonome

Cette couche formelle est introduite pour faciliter le raisonnement et la gestion d'applications M2M. Elle repose sur une grammaire de graphe "générique" caractérisant un système M2M dont le fonctionnement est régulé par le standard ETSI M2M. Ces deux modèles doivent donc être consistants ; les applications construites selon le paradigme M2M peuvent être vues comme des instances des deux modèles à la fois, i.e. du standard ETSI M2M et de la grammaire générique élaborée ici. Dans ce dernier cas, le terme instanciation peut prêter à confusion. Précisons donc que la dite application est alors caractérisée par une grammaire de graphe qui est une instanciation de la grammaire générique.

5.3.2.1 Analyse préliminaire : informations nécessaires

Afin de mener à bien la gestion autonome du système, les informations suivantes sont considérées dans la couche formelle :

- Les équipements déployés et le type d'applications qu'ils peuvent supporter. En effet, une application ne peut pas nécessairement être exécutée sur tout type d'appareil ; l'acquisition de données, par exemple, ne peut être conduite sur un microprocesseur mais uniquement sur des capteurs dédiés.
- Les connexions établies. Plus précisément, il s'agit de la visibilité que chaque appareil possède en lien avec les processus d'enregistrement et d'annonce. Les délais de propagation dus aux routes et équipements physiques sous jacents à chaque connexion.
- Les conteneurs déployés, leur hôte et les appareils auxquels ils se sont annoncés.
- Les applications déployées, leur hôte, leur type, les appareils auxquels elles se sont annoncées, les conteneurs qu'elles utilisent et le type d'utilisation (lecture et/ou écriture) qu'elles en font.

5.3.2.2 Caractérisation par une grammaire de graphe

Axiome. L'axiome est encore une fois purement théorique et représenté par un nœud noté *AX*.

Termes terminaux. Dans le contexte présent, ces termes caractérisent les types d'entités présentes dans les systèmes M2M, ainsi que les informations leur étant liées. Notons ID l'ensemble des valeurs que peut prendre l'identifiant unique d'un composant. D'après l'analyse préliminaire précédente, il existe quatre types de composants :

- Les conteneurs dont l'archétype est $T_c = \nu_c((\mathrm{id}, \mathrm{ID}))$.
- Les applications. Les attributs de ces dernières précisent leur identifiant et leur type. En notant Nat_t l'ensemble des types d'applications, l'archétype des applications est $T_a = \nu_a($ $(\mathrm{id}, \mathrm{ID}), (nat, \mathrm{Nat}_t))$.
- Les équipements communicants, c'est à dire les passerelles et appareils ETSI. En plus d'un identifiant, ces derniers possèdent deux attributs reflétant leur nature et le type d'applications qu'ils peuvent supporter. En notant $\mathrm{Nat}_e = \{$"pass", "ETSIeq"$\}$ et Exec (comme exécutable) l'ensemble des parties de Nat_t, l'archétype représentant les équipements communicants est $T_e = \nu_e((\mathrm{id}, \mathrm{ID}), (nat, \mathrm{Nat}_e), (exec, \mathrm{Exec}))$.

— Le réseau, une entité abstraite figurant un réseau de communication non-local. Ce dernier peut également englober divers fournisseurs de ressources allant jusqu'aux data center. Le réseau étant unique, il ne nécessite pas d'identifiant. Son archétype est alors $T_r = \nu_r(\text{"réseau"}, \{\text{"réseau"}\}, (exec, Exec))$.

Dans le cadre de l'augmentation décrite par l'algorithme 4.1, les termes terminaux sont enrichis d'un attribut compteur dont le domaine de définition est $\mathbb{N}$.

Productions. Avant d'introduire les productions elles mêmes, considérons les relations pouvant être établies entre les différentes entités, et donc les attributs pouvant apparaître sur des arcs.

— enregistré, noté "e" : deux instances de l'archétype équipement ou une instance de ce dernier et une de l'archétype réseau peuvent être enregistrées l'une à l'autre.
— déployé sur, noté "d" : d'un conteneur ou une application vers l'appareil sur lequel il est déployé. Il apparaît alors dans le SCL du dit appareil.
— annoncé à, noté "a" : d'un conteneur ou une application vers un équipement distant. Cette relation décrit la nature annoncée et visible d'une ressource vis à vis d'un appareil distant.

Rappelons que les productions de la grammaire décrivent les conditions dans lesquelles une entité peut être déployée ou une relation initiée, ainsi que la manière de le faire. Par soucis de clarté et de lisibilité, les domaines de définition des attributs sont implicites dans la suite, cela n'introduisant pas d'ambiguïté.

La règle de production p_1, illustrée dans la figure 5.3, décrit l'initialisation d'un système M2M consistant dans le déploiement du réseau abstrait.

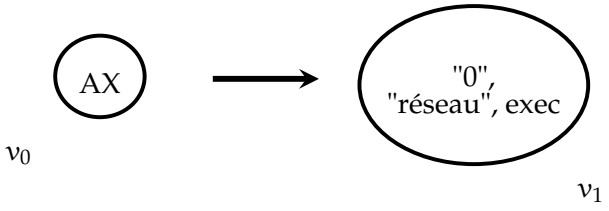


FIGURE 5.3 – Initialisation (p_1)

L'addition (ou démarrage) d'une passerelle ou d'un appareil ETSI est caractérisée par la règle p_2 représentée dans la figure 5.4. Un équipement doit s'enregistrer sur le réseau et voit les ressources du réseau de la même façon. Les délais de transmissions, notés δ, de chaque connexion établie sont également pris en considération.

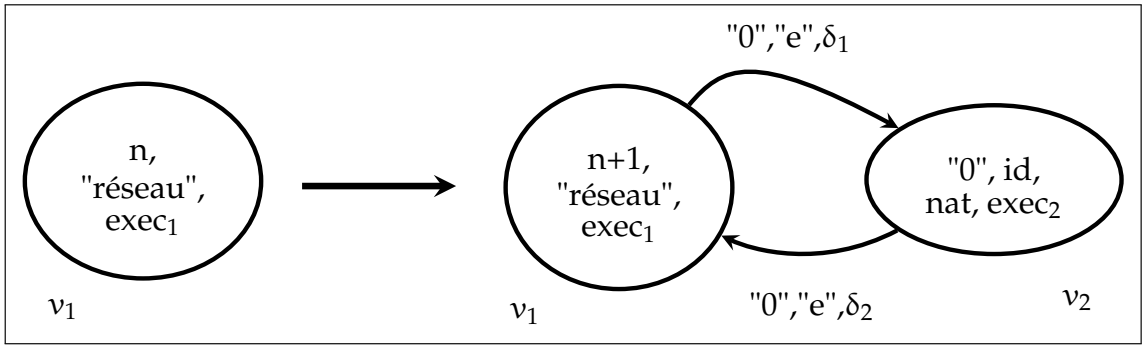
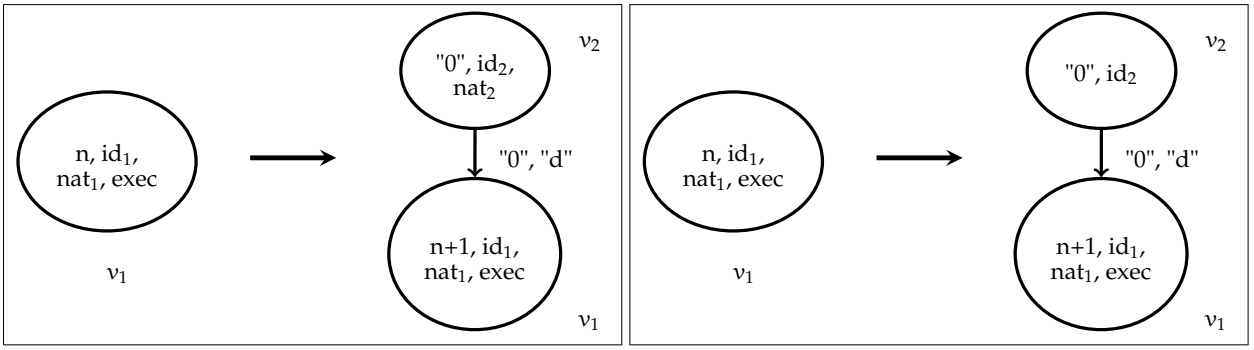
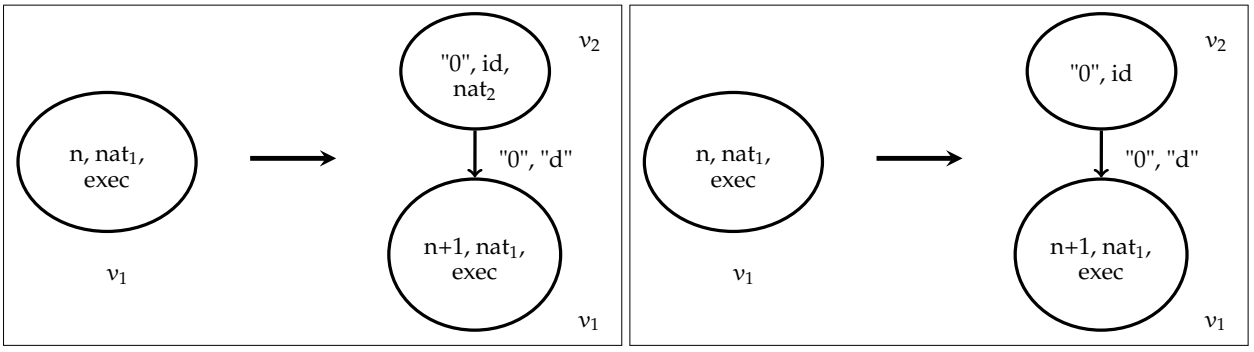


FIGURE 5.4 – Addition d'un équipement communicant (p_2)

Les règles p_3, p_4, p_5 et p_6, définies respectivement dans les figures 5.5a, 5.5b, 5.5c et 5.5d, présentent de fortes similitudes. Ces transformations formalisent le démarrage d'une application (p_3 et p_5) ou le déploiement d'un conteneur (p_4 et p_6) sur un équipement communicant (p_3 et p_4) ou sur le réseau (p_5 et p_6). Notons qu'une application devrait être déployée sur un appareil pouvant l'exécuter (i.e., $nat_2 \in exec$).



(A) Démarrage d'une application (p_3) (B) Création d'un conteneur (p_4) sur un équipement communicant.



(C) Démarrage d'une application (p_5) (D) Création d'un conteneur (p_6) sur le réseau.

FIGURE 5.5 – Ajout d'une entité

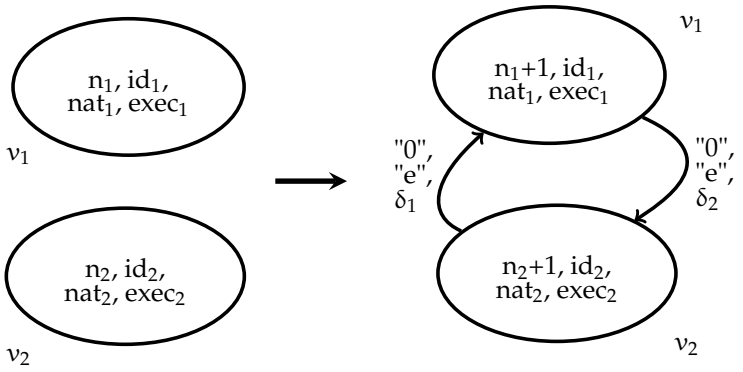


FIGURE 5.6 – Enregistrement pair à pair (p_7)

La production p_7, représentée dans la figure 5.6, décrit l'enregistrement pair à pair d'un équipement communicant à un autre. L'application de cette règle entraîne l'ajout de deux arcs (la relation d'enregistrement étant réflexive).

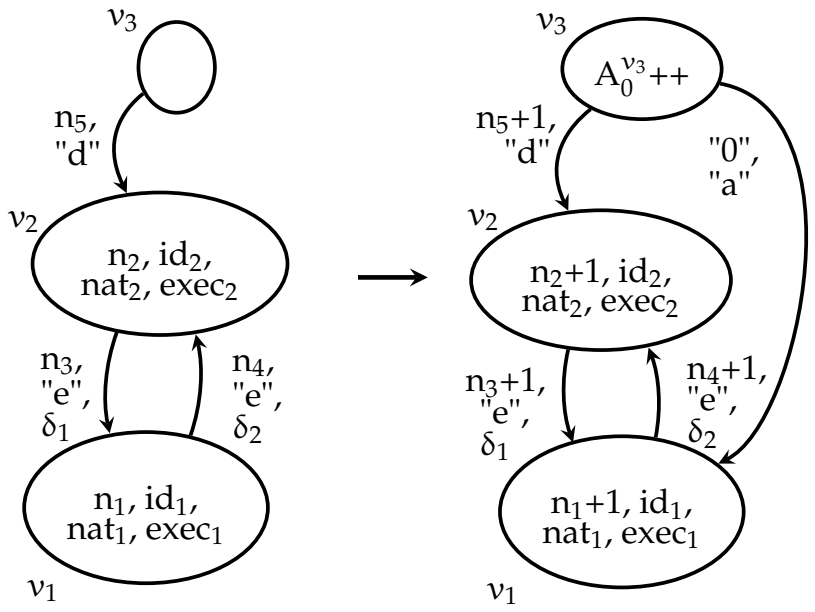


FIGURE 5.7 – Annonce d'une application ou d'un conteneur (p_8)

Une application ou un conteneur peut s'annoncer à tout appareil enregistré à son équipement hôte. Cette condition et l'annonce elle-même sont formalisées par la production p_8, illustrée dans la figure 5.7. Le noeud ν_3 n'est pas attribué et peut par conséquent être identifié à n'importe quel nœud. Néanmoins, ce nœud étant à la source d'un arc de type "déployé sur", seule une application ou un conteneur peut lui être associé (lorsque p_8 est appliquée à une instance de la grammaire définie ici). Bien que ν_3 ne soit pas attribué, le noeud lui étant associé l'est et son compteur de matching est incrémenté lors de l'application de la règle.

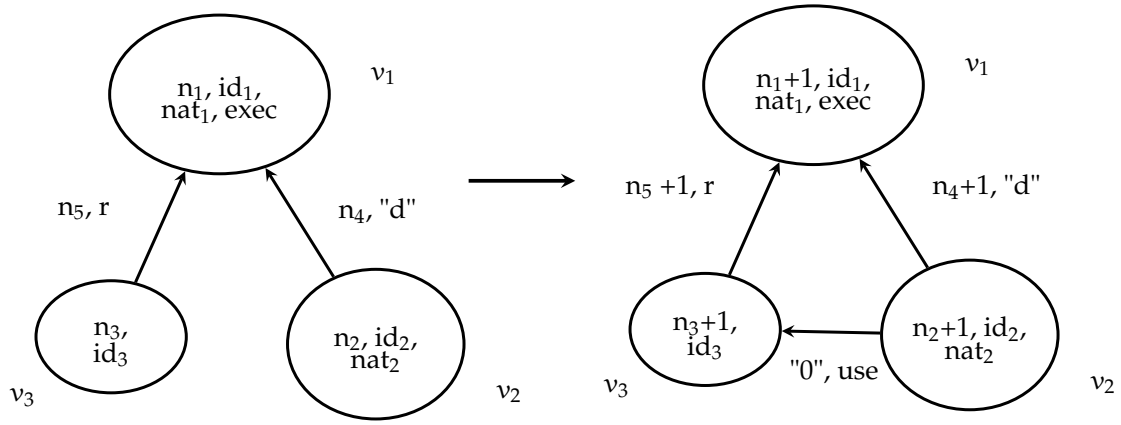


FIGURE 5.8 – Initialisation d'une relation d'utilisation d'une application vers un conteneur (p_9)

Une application peut utiliser un conteneur, c'est à dire le lire et/ou y écrire, si l'une de ces conditions est remplie :

— Les deux sont déployés sur le même hôte.
— L'application est exécutée sur un appareil auquel le conteneur est annoncé.

En d'autres termes, une application peut utiliser un conteneur si son hôte est la destination d'un arc dont le conteneur est la source, et dont le type est soit "déployé sur" ou "annoncé à". Comme tout arc d'un conteneur vers un équipement possède un de ces deux types, une variable r peut couvrir les deux cas sus-cités. La production p_9 illustrée dans la figure 5.8 décrit l'établissement d'une

relation d'utilisation. L'attribut variable use reflète le type d'utilisation initiée, soit "L", "E", "L/E" pour lecture, écriture et lecture/écriture.

Grammaire. Finalement, la grammaire de graphe caractérisant les systèmes M2M est (AX, NT, T, P) où :
$NT = \emptyset$,
$T = \{T_e, T_r, T_c, T_a\}$,
$P = \bigcup_{i \in [1,9]} p_i$.

Illustration. La figure 5.9 représente le graphe symbolisant la configuration dont l'illustration utilisant le modèle ETSI est proposée dans la figure 5.2. Les compteurs de reconnaissances, invisibles pour l'utilisateur, ne sont pas explicités. On y retrouve le réseau, les deux équipements, l'application et le conteneur présents dans la représentation orientée arbre de ressources. Les relations d'enregistrement et d'annonce sont ici explicites. Les deux couches offrent deux visions cohérentes (c'est à dire qu'elles représentent la même configuration) et non pas identiques mais complémentaires : les informations contenues dans chaque couche ne sont pas les mêmes. Par exemple, au niveau de la couche de gestion autonome, les attributs δ, nat et $exec$ fournissent des informations inédites. Elles sont relatives aux délais de transmission, aux types d'applications et aux types d'applications supportées, respectivement.

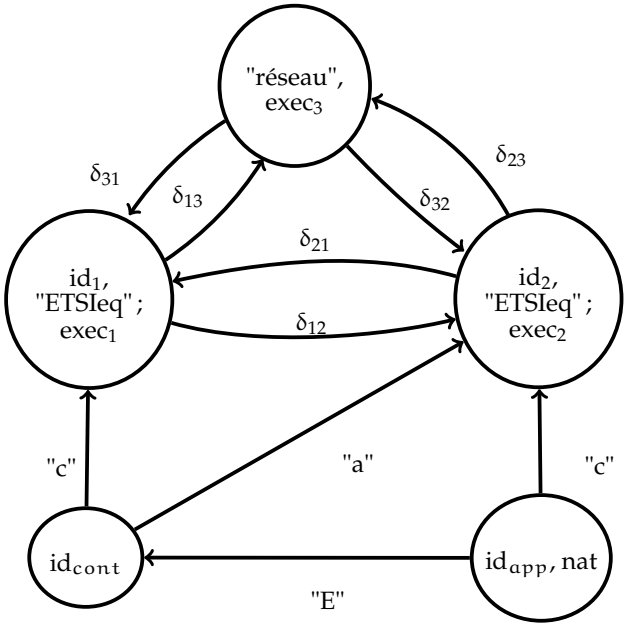


FIGURE 5.9 – Représentation d'une configuration M2M d'après le modèle graphe

5.3.3 Cohérence des modèles : communications et mises à jour bidirectionnelles

La multiplicité des modèles pose le problème de leur cohérence. Des évolutions peuvent trouver leur origine dans chacune des couches à travers les processus de découvertes et de reconfigurations. Pour assurer la cohérence interne de la vue architecturale proposée, et donc des deux modèles qui la composent, ces transformations doivent par conséquent être impactées de manière bidirectionnelles.

Il sera en plus considéré que toute évolution est la conséquence d'un de ces deux types de processus. En effet, une troisième origine pourrait être envisagée, à savoir le système réel. Néanmoins, une évolution ayant un tel point d'origine, telle qu'une panne de machine, peut être vue comme étant à l'initiative du gestionnaire autonome, en la ramenant à un événement géré par le gestionnaire. Celui-ci lancera alors les reconfigurations nécessaires, incluant les conséquences de l'événement sur le système.

5.3.3.1 De la couche fonctionnelle à la couche de gestion autonome

Lorsqu'une nouvelle entité est déployée ou rejoint le système, elle est repérée et prise en considération grâce au protocole de découverte mené à bien dans la couche fonctionnelle. Dans les systèmes M2M, une telle entité peut être de trois différents types : conteneur, application ou appareil. Notons qu'un conteneur ou une application peut être découvert soit sur un appareil pré-existant, soit au moment de la découverte de l'appareil sur lequel il est déployé.

Découverte d'un appareil. Un événement est envoyé au gestionnaire autonome, plus précisément à son module de monitoring. Celui-ci contient l'identifiant unique de l'appareil, le type d'application qu'il peut exécuter, et sa nature (i.e. passerelle ou appareil intelligent). Les phases de monitoring, d'analyse et de planification sont triviales : l'action finalement effectuée lors de la phase d'exécution est la mise à jour du modèle basé graphe à travers l'application de la règle de production p_2. Les valeurs des attributs id, nat et $exec_2$ de p_2 sont fixées en fonction des informations contenues dans l'événement initial.

Remarque 5.1. Il est intéressant de noter qu'une règle dont certains attributs initialement variables sont instanciés est en fait, à proprement parler, une spécialisation de la règle initiale. Une telle spécialisation est extrêmement simple, les morphismes impliqués étant caractérisés par la fonction identité accompagnée de l'ensemble d'identification fixant les valeurs concernées.

Découverte d'un conteneur. Un événement contenant les identifiants du conteneur et de sa machine hôte est envoyé au gestionnaire autonome. Encore une fois, les diverses phases du processus de gestion autonome sont triviales, l'action finale ne dépendant que de la nature de l'hôte.

Si l'hôte est le réseau, l'action finalement effectuée est la mise à jour du modèle basé sur les graphes via l'application de la production p_6. Dans le cas contraire, la production p_4 est appliquée, la valeur de id_1 étant fixée à l'identifiant reçu.

Découverte d'une application. Un événement est envoyé au gestionnaire autonome, en précisant les identifiants de l'application et de son hôte, ainsi que le type de l'application. Après réception et suivant la nature de l'hôte, la production p_3 ou p_5 est appliquée au graphe modèle par le module d'exécution de gestionnaire autonome.

5.3.3.2 De la couche de gestion autonome à la couche fonctionnelle

Actions, transformations et reconfigurations peuvent être effectuées à la suite de l'application d'une politique d'auto gestion. Leurs implications doivent alors être impactées sur le système réel et ses

représentations. Ici, le système réel peut être assimilé à la couche ETSI M2M, celle-ci assurant la gestion fonctionnelle des équipements M2M.

Ces transformations sont modélisées dans la couche de gestion autonome par des règles de réécriture de graphe. Les effets de ces dernières peuvent facilement être décomposés en deux ensembles d'éléments (nœuds et arcs) ajoutés et supprimés. Leur analyse et la détermination des actions à effectuer sur la couche fonctionnelle peut se baser sur une telle décomposition. Chaque élément concerné amène ainsi son lot d'opérations en fonction de son ensemble d'appartenance et de ses attributs.

Nature des actions. Les actions à effectuer sont en fait des appels REST. À chaque élément est associé un ensemble d'actions dépendant de sa nature comme suit :

Appareil. L'ajout ou la suppression d'un appareil dans le modèle basé sur les graphes n'implique pas son ajout ou sa suppression dans la couche fonctionnelle, et, a fortiori, dans le système réel. Cela entraîne plutôt un changement de son statut notifiant s'il est actif ou non. Ce mécanisme permet de conserver les informations et enregistrements relatifs à l'appareil lors de son arrêt.

— *Ajout/Activation.* Un appel UPDATE avec le statut "active" est envoyé, dans la couche fonctionnelle, à l'arbre de ressources de l'appareil concerné.

— *Suppression/Désactivation.* Un appel UPDATE avec le statut "idle" est effectué de façon similaire.

Application et/ou conteneur.

— *Ajout.* Un appel CREATE contenant les informations nécessaires est envoyé, dans la couche fonctionnelle, à l'arbre de ressources de l'appareil sur lequel l'application ou le conteneur est en cours de déploiement.

— *Suppression.* Un appel DELETE contenant les informations nécessaires est effectué de manière similaire.

Enregistrement. Seuls sont concernés ici les enregistrements pair-à-pair. Lorsqu'un appareil s'enregistre à un autre, deux ressources, une sur chacun des protagonistes, sont créées.

— *Ajout/Enregistrement.* Afin d'ajouter un canal de communication pair-à-pair, un appel de type CREATE<scl> est envoyé, dans la couche fonctionnelle, d'un équipement à l'autre.

— *Suppression/Désenregistrement.* Pour effacer un canal de communication pair-à-pair, un appel DELETE<scl> doit être effectué, dans la couche fonctionnelle, d'un équipement à l'autre.

Annonce.

— **Ajout/Annonce.** Un appel de type CREATE avec pour argument la ressource "applicationAnnc" ou "containerAnnc" est généré dans la couche fonctionnelle et envoyé à l'arbre de ressources de l'appareil cible.

— **Suppression/Désannonce.** Un appel de type DELETE ayant pour argument la ressource "applicationAnnc" ou "containerAnnc" est envoyé, dans la couche fonctionnelle, à l'arbre de ressources de l'appareil cible.

Utilisation. Lorsqu'une application modifie son utilisation d'un conteneur, deux appels UPDATE doivent être effectués avec les informations correspondantes. Le premier appel, dirigé vers le SCL de la machine sur laquelle l'application est déployée, met à jour la ressource "conteneur" utilisée par l'application. Le second appel, dont la cible est le SCL de l'hôte du conteneur, entraîne la mise à jour de ses droits d'accès, "accessRights".

5.3.3.3 Priorités

Pour assurer la cohérence des transformations, découvertes et actions de reconfigurations doivent être traitées dans un ordre spécifique.

Dans le cas d'un élément ajouté ou découvert :

1. Les actions relatives aux entités doivent être traitées en premier, dans l'ordre suivant :
   (a) appareils
   (b) conteneurs
   (c) application
2. Les actions liées aux relations peuvent alors être exécutées, dans l'ordre suivant :
   (a) enregistrement
   (b) annonce
   (c) utilisation

Les priorités sont simplement inversées pour les éléments supprimés (i.e., relation d'utilisation, puis d'annonce jusqu'à la suppression d'appareils).

Il n'existe pas d'ordre particulier entre les actions découlant d'additions d'éléments et celles découlant de suppressions. Les actions de même type ne sont pas ordonnées ; par exemple, si deux applications sont ajoutées par la même transformation, elles peuvent être démarrées dans n'importe quel ordre (ou même simultanément).

5.4 Politiques de gestion autonomes

Cette section exploite la représentation proposée précédemment et démontre son adéquation en détaillant un échantillon des politiques d'auto-gestion que nous avons élaborées dans le cadre du projet OM2M [20]. Ces dernières peuvent s'appliquer à tout système se conformant au standard ETSI M2M. Elles rentrent dans le cadre de l'auto-guérison pro-active du système, de manière à respecter son auto-protection interne et en incluant des éléments d'auto-optimisation.

La première politique vise à prévenir des saturations du réseau. Un trop grand nombre d'accès distants à un conteneur peut entraîner une saturation des canaux de communication de son appareil hôte. Afin de prévenir une telle issue, le conteneur ciblé doit être migré sur l'entité abstraite "réseau". Celle-ci possède les meilleures capacités de communication du système ainsi que de multiples points d'accès, la prémunissant *a priori* contre de telles saturations. Chaque application utilisant le conteneur initial doit être redirigée vers le nouveau, ce dernier devant avoir une visibilité en conséquence.

La seconde politique a pour objectif la prévention de pertes de données. L'extinction d'un appareil communicant suite à l'affaiblissement de sa batterie peut entraîner la perte des données qu'il contient. Pour empêcher une telle perte, tout conteneur déployé sur un appareil ayant une batterie faible est déplacé sur un autre appareil. La cible de cette migration peut impacter la qualité du système ; une attention particulière doit par conséquent être portée à son choix. Afin de garantir la continuité des services, chaque application utilisant un conteneur migré doit être redirigée vers sa nouvelle localisation. L'hébergement de conteneur entraînant un surcoût énergétique, cette politique permet également de prolonger la durée de vie de l'équipement en ménageant sa batterie.

20. Des politiques supplémentaires sont consultables dans [63].

Notons que l'algorithme à appliquer peut être conçu et décomposé de différentes manières entre chaque module de gestion. Le graphe représentant le modèle de gestion autonome est noté $G = (V, E, ATT)$.

5.4.1 Observation

Nous supposons le niveau de charge des batteries de chaque appareil et les accès aux conteneurs surveillés. Ces informations sont collectées par le module d'observation afin de générer, le cas échéant, les deux symptômes suivants :

— "Nombreux accès distants" lorsque des applications distantes ont effectué plus de x accès à un conteneur c en un intervalle de temps t. Outre son nom (sa catégorie), ce symptôme est accompagné de l'identifiant id_c du conteneur c.
— "Batterie faible" lorsque la batterie d'un appareil a tombe en dessous d'un seuil de x% de charge. Ce symptôme contient l'identifiant id_a de l'appareil a. Notons que a ne peut être le réseau, ce dernier n'étant pas alimenté par une batterie à durée de vie limitée.

Chacun de ces symptômes doit être traité par le gestionnaire dans les politiques d'auto-gestion décrites dans la suite.

5.4.2 Analyse

5.4.2.1 Réception d'un symptôme "Nombreux accès distants"

À la réception d'un symptôme "Nombreux accès distants", le module d'analyse doit créer une requête de migration vers le réseau si le conteneur concerné n'y est pas déjà déployé. Ce processus est décrit dans l'algorithme 5.1.

Données : id_c, l'identifiant du conteneur concerné.
début
  si id_c *n'est pas déployé sur le réseau* **alors**
    créer RFC *type* "Migration de conteneur" *arguments* id_c et "réseau"
  fin
fin

ALGORITHME 5.1 - Analyse d'un symptôme "Nombreux accès distants"

Remarque 5.2. Cette condition peut facilement être évaluée dans le contexte d'une modélisation par des graphes ; elle est équivalente à l'existence d'un homomorphisme entre la partie droite de p_4, la valeur de id_2 étant fixée à id_c, et G.

5.4.2.2 Réception d'un symptôme "Batterie faible"

Chaque conteneur déployé sur un appareil ayant une "Batterie faible" doit être déplacé vers un appareil approprié, comme décrit dans l'algorithme suivant.

Données : $\mathtt{id}_a$, l'identifiant de l'appareil concerné.
début
  pour chaque *conteneur* $\mathtt{id}_c$ *déployé sur* $\mathtt{id}_a$ **faire**
    créer RFC *type* "Migration de conteneur" *arguments* $\mathtt{id}_c$ et rechercheCible($\mathtt{id}_c$)
  fin
fin

ALGORITHME 5.2 - Analyse d'un symptôme "Batterie faible"

Remarque 5.3. Encore une fois, les conteneurs déployés sur un certain équipement peuvent facilement être trouvés dans le contexte d'une modélisation par des graphes ; il s'agit de chercher des homomorphismes entre la partie droite de p_4, la valeur de $\mathtt{id}_1$ étant fixée à $\mathtt{id}_a$, et G.

La fonction rechercheCible($\mathtt{id}_c$) cherche l'équipement le plus approprié pour servir de cible à une migration du conteneur $\mathtt{id}_c$. Idéalement, cette fonction devrait être menée à bien par un module de placement. Nous proposons ci-après un algorithme de recherche simple optimisant les temps de transmission. Ce processus, décrit par l'algorithme 5.3, découle des considérations suivantes.

Le conteneur devrait être atteignable le plus rapidement possible. Par conséquent, l'emplacement le plus approprié pour sa migration est considéré être celui minimisant la somme des délais de transmissions à partir de chaque utilisateur (i.e., chaque application utilisant le conteneur) tout en ayant un niveau de batterie suffisant. Ici, seuls sont connus les délais de transmission entre appareils enregistrés l'un à l'autre. Nous considérons par conséquent uniquement les équipements visibles depuis chaque appareil exécutant une application utilisant le conteneur initial. Dans les faits, les enregistrements nécessaires pourraient être effectués après le choix de la cible.

Dans les systèmes réels, il est hautement probable que l'appareil approprié exécute une application utilisant $\mathtt{id}_c$, puisqu'il annule ainsi un des termes de la somme à minimiser. Par conséquent, les appareils potentiellement visés sont restreints à ceux vérifiant cette condition. Cet ensemble peut être construit en cherchant des homomorphismes de la partie droite d'une certaine spécialisation

de p_9 vers G. Il est supposé connu et noté ciblePotentielle(id_c). Les identifiants des appareils correspondants sont regroupés dans l'ensemble ciblePotentielleId(id_c).

Fonction rechercheCible
Données : id_c, l'identifiant du conteneur concerné.
début
  $G^S \leftarrow$ le sous graphe de G induit par ciblePotentielle(id_c)
  pour chaque $id_1 \in$ *ciblePotentielleId(*id_c*)* **faire**
    $C \leftarrow$ ciblePotentielleId(id_c) $\setminus \{id_1\}$
    si $(\forall id_2 \in C, (id_2, id_1) \in E_{G^S}) \wedge$ *batterie(*id_1*)* $> x$ **alors**
      somme(id_1) $\leftarrow \sum_{id_3 \in C} (A_{G^S})_3^{(id_1, id_2)}$
    sinon
      somme(id_1) $\leftarrow \infty$
    fin
  fin
  cible $\leftarrow$ id, somme(id) = $\min_{i \in ciblePotentielle(id_c)}$ somme(i)
  si *somme(cible)* $\neq \infty$ **alors**
    retourner cible
  sinon
    retourner "réseau"
  fin
fin

ALGORITHME 5.3 - Fonction cherchant un hôte approprié pour migration, rechercheCible

5.4.3 Planification

Afin de mener à bien une RFC "Migration de conteneur", le module correspondant planifie les actions suivantes :

1. copie du conteneur à migrer vers l'appareil cible.
2. redirection des applications du conteneur initial vers sa copie.
3. suppression du conteneur initial.

L'établissement de ce plan d'actions ne nécessite pas de raisonnement particulier, comme décrit dans l'algorithme suivant.

Données : id_c, l'identifiant du conteneur à migrer.
cible, "réseau" ou l'identifiant d'un appareil communicant.
début
  $id_{new} \leftarrow$ un identifiant unique libre qui sera associé au nouveau conteneur.
  créer action *type* "Copie de conteneur" *arguments* id_c, id_{new} et cible
  créer action *type* "Redirection conteneurs" *arguments* id_c et id_{new}
  créer action *type* "Suppression de conteneur" *argument* id_c
fin

ALGORITHME 5.4 - Planification d'une "Migration de conteneur"

L'ordre des actions dans cet algorithme correspond également à leurs priorités : "Copie de conteneur" doit être exécutée la première, puis "Redirection conteneur", etc.

5.4.4 Exécution

Le module d'exécution effectue finalement les actions reçues. Les transformations lancées à la fin du processus de décision sont décrites en utilisant des règles de réécritures correctes par construction, conformément à la méthodologie proposée dans le chapitre précédent.
Dans la suite, "MAJ couche fonctionnelle" réfère au processus décrit dans la sous section 5.3.3.2.

5.4.4.1 Copie de conteneur

Cette action, décrite dans l'algorithme suivant, repose sur deux spécialisations de p_4 et p_6. Le processus de spécialisation dont ces deux règles découlent est simple ; il n'est constitué que d'une affectation. Notons :

— $p_4(id_a, id_b)$ la spécialisation de p_4 obtenue en lui appliquant [21] l'affectation induite par l'ensemble d'identifications $\{(id_1, id_a), (id_a, id_1), (id_2, id_b), (id_b, id_2)\}$.
— $p_6(id_a)$ la spécialisation de p_6 obtenue en lui appliquant l'affectation induite par $\{(id, id_a), (id_a, id)\}$.

Données : id_c, l'identifiant du conteneur à copier.
id_{new}, l'identifiant qui sera associé au nouveau conteneur.
cible, "réseau" ou l'identifiant d'un appareil communicant.
début
  si *cible = "réseau"* **alors**
    appliquer $p_6(id_c)$ à G
    MAJ couche fonctionnelle
  sinon
    appliquer $p_4(id_c, cible)$ à G
    MAJ couche fonctionnelle
  fin
  copier les données de id_c vers id_{new}
fin

ALGORITHME 5.5 - Exécution d'une "Copie de conteneur"

Le premier "MAJ couche fonctionnelle" intervient après l'application de $p_6(id_c)$, cette dernière entraînant l'ajout d'un nœud de type "conteneur" et d'un arc de type "déployé sur". Dans ce cas particulier, la mise à jour consiste donc à effectuer les actions décrites sous "Conteneur-ajout", soit un appel CREATE.

5.4.4.2 Redirection conteneurs

Il s'agit, dans un premier temps, d'effectuer les enregistrements et annonces rendant visible un conteneur d depuis tout appareil exécutant une application utilisant un conteneur s. Ces applications sont ensuite redirigées vers d.

Ce processus repose sur :

— redirection(id_s, id_d), décrivant la redirection d'une entrée et/ou d'une sortie d'une application de s vers d. Cette règle, illustrée dans la figure 5.10, est obtenue en spécialisant le résultat de la composition de p_9 avec p_9^{-1}. Le motif commun dont dépend la composition est

21. Plus précisément, en appliquant à ses parties gauche et droite.

l'application ciblée par la redirection, son hôte et leur relation de déploiement. La spécialisation consiste à fixer les valeurs des identifiants des conteneurs intervenant dans p_9 et son inverse à id_s et id_d, respectivement.

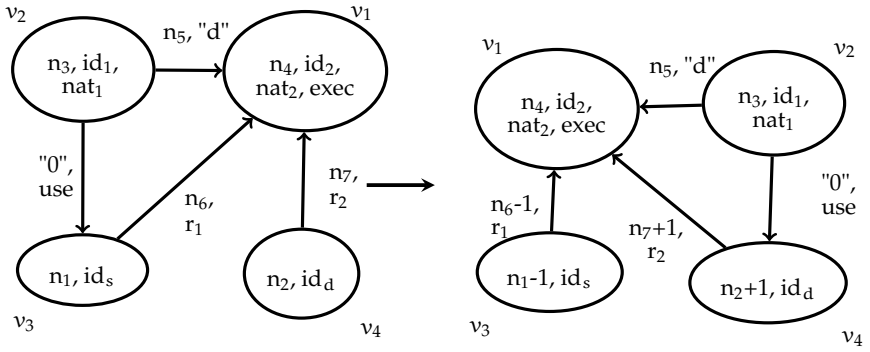


FIGURE 5.10 – Redirection d'une entrée et/ou d'une sortie d'une application (redirection(id_s, id_d) spécialisation de $p_9 \circ p_9^{-1}$)

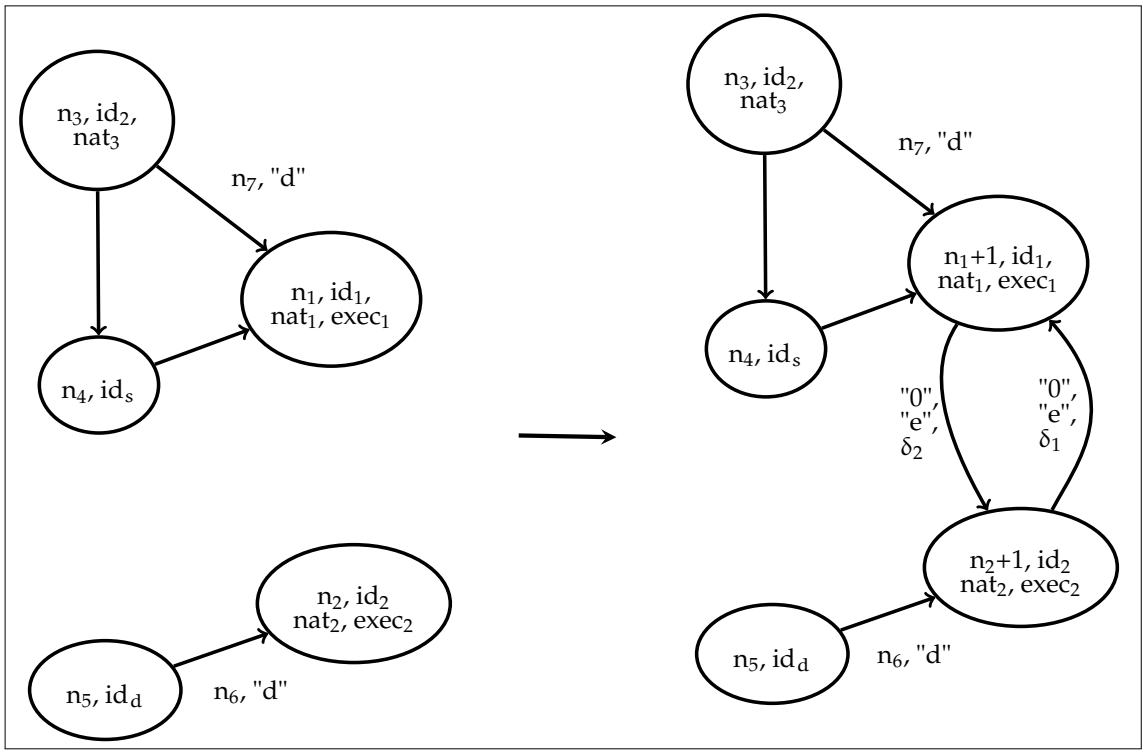
— enregistrement(id_s, id_d), la spécialisation de p_7 représentée dans la figure 5.11a. Cette règle décrit un enregistrement pair à pair entre l'hôte de d et tout hôte exécutant une application utilisant s.
— annonce(id_s, id_d), la spécialisation de p_8 représentée dans la figure 5.11b. Cette règle décrit l'annonce de d à tout hôte exécutant une application utilisant s.

Données : id_s, l'identifiant du conteneur source.
id_d, l'identifiant du conteneur destination.
début
  si id_d *n'est pas déployé sur le réseau* **alors**
    tant que *il existe un homomorphisme* h *selon lequel enregistrement*(id_s, id_d) *est applicable à* G **faire**
      appliquer enregistrement(id_s, id_d) à G selon h
      MAJ couche fonctionnelle
    fin
  fin
  tant que *il existe un homomorphisme* h *selon lequel annonce*(id_s, id_d) *est applicable à* G **faire**
    appliquer annonce(id_s, id_d) à G selon h
    MAJ couche fonctionnelle
  fin
  tant que *il existe un homomorphisme* h *selon lequel redirection*(id_s, id_d) *est applicable à* G **faire**
    appliquer redirection(id_s, id_d) à G selon h
    MAJ couche fonctionnelle
  fin
fin

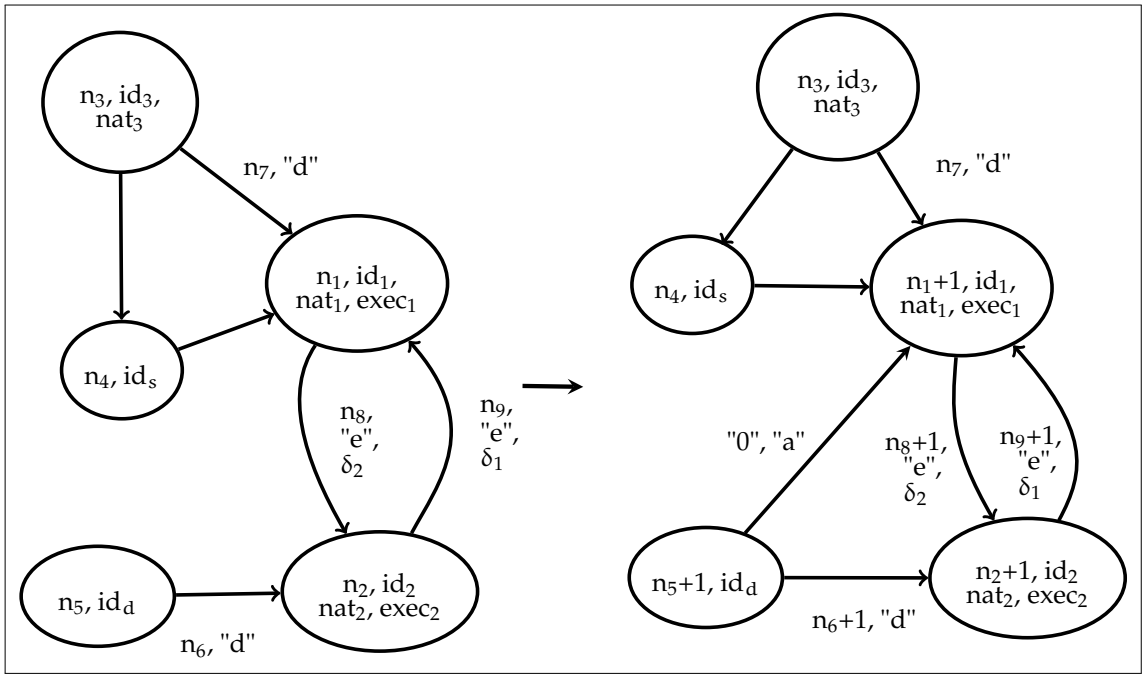
ALGORITHME 5.6 - Exécution d'une "Redirection de conteneur"

5.4.4.3 Suppression de conteneur

Cette action, décrite dans l'algorithme ci-dessous, se base sur une spécialisation de l'inversion de la règle p_4. Encore une fois, le processus de spécialisation est simple car restreint à l'application d'une affectation. Notons sup(id_a) la spécialisation de p_4^{-1} obtenue en lui appliquant l'affectation induite par $\{(id_2, id_a), (id_a, id_2)\}$.



(A) Enregistrement pair à pair avant redirection
(enregistrement(id_s, id_d) spécialisation de p_7)



(B) Annonce du nouveau conteneur avant redirection
(annonce(id_s, id_d) spécialisation de p_8)

FIGURE 5.11 – Gestion de la visibilité avant redirection

Données : id_c, l'identifiant du conteneur à supprimer.
début
  appliquer sup(id_c) à G
  MAJ couche fonctionnelle
fin

ALGORITHME 5.7 - Exécution d'une "Suppression de conteneur"

5.5 Cas d'utilisation et expérimentations : compteurs intelligents

Cette section se repose sur une implémentation effectuée dans le cadre du projet OM2M. L'objectif est de prouver la faisabilité de l'approche proposée et d'étudier son extensibilité.

5.5.1 Configurations expérimentales

Les expérimentations présentées se basent sur le cas d'utilisation décrit dans [132]. Il s'agit d'une application de domotique, plus précisément un système de compteur intelligent. Un tel compteur se repose sur une infrastructure de comptage avancée [22] contenant divers équipements hétérogènes. Ils sont mis en réseau et coopèrent pour mener à bien des objectifs pré-définis tels que l'observation de la consommation énergétique et sa facturation.

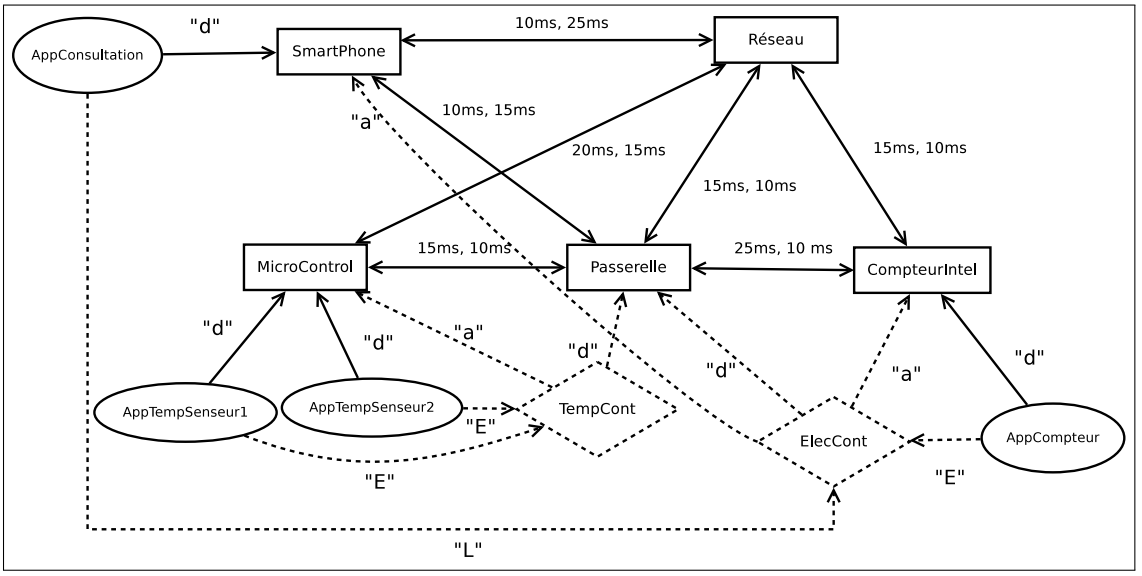
L'installation comporte un compteur intelligent, une passerelle et un micro-contrôleur communicants localement et mis en réseau. Les informations observées peuvent être consultées par l'utilisateur via une application de son smartphone.

Des senseurs de température non intelligents sont mis en réseau via le micro-contrôleur. Ils sont vus comme des applications déployées sur ce dernier. Le compteur intelligent exécute quant à lui des applications de comptage.

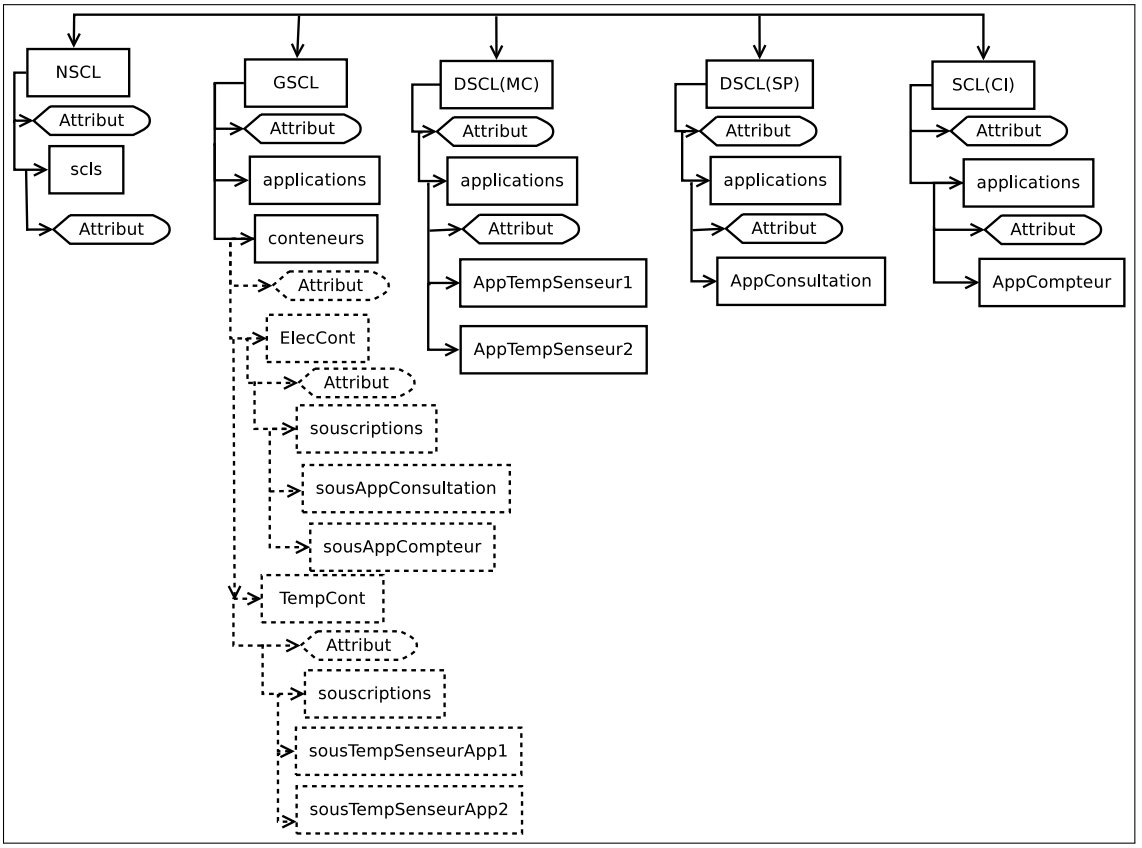
Des conteneurs stockant les informations relatives à la température et la consommation électrique sont déployés sur la passerelle, celle-ci possédant d'importantes capacités de stockage.

Les configurations considérées ont un nombre variable de conteneurs et d'applications collectant ou consultant des données. La configuration la plus simple, comportant trois applications de collecte et deux conteneurs, est illustrée dans la figure 5.12. Afin de clarifier la figure 5.12a relative au modèle basé sur les graphes, un certain nombre de simplifications ont été adoptées. Les relations d'enregistrement ont été modélisées par une flèche double plutôt que deux flèches simples. Les attributs "e" d'arcs représentant une relation d'enregistrement et les attributs des nœuds sont implicites. La couche fonctionnelle est représentée dans la figure 5.12b par l'arbre de ressources du réseau auquel a été ajouté, à des fins illustratives, les ressources qu'il ne voit pas (conteneurs et applications).

22. en anglais, *Advanced Monitoring Infrastructure*



(A) Couche de gestion : Représentation basée sur les graphes



(B) Couche fonctionnelle : Arbres de ressources

(C) Légende

FIGURE 5.12 – Configuration initiale minimale

5.5.2 Scénario de reconfiguration

Dans ces expériences, le scénario de transformation effectué et évalué est l'application de la politique de gestion faisant suite à l'affaiblissement de la batterie de la passerelle. La configuration résultant de l'application de cette politique à la configuration initiale la plus simple (voir figure 5.12) est illustrée dans la figure 5.13.

En l'état, l'extinction de la passerelle entraînerait la perte des données déjà collectées et rendrait impossible le stockage des futurs relevés. Conformément à la politique définie dans la section précédente, une requête de migration est générée pour chacun des conteneurs déployés sur la passerelle (soit tous les conteneurs du système).

Le choix de l'appareil cible est conduit par la fonction rechercheCible.
Les conteneurs relatifs à la température sont utilisés uniquement par les applications la mesurant. Celles-ci étant toutes déployées sur le micro-contrôleur, il sera sélectionné par la fonction rechercheCible, annulant ainsi la somme des délais de transmissions liés à l'utilisation du conteneur.
Les historiques de consommation énergétique sont utilisés à la fois par l'application de comptage intelligent et l'application mobile de consultation. Les hôtes de ces applications, le compteur et le smartphone, n'étant pas enregistrés l'un à l'autre, le réseau sera la cible de cette migration.

Par la suite, chacun des conteneurs est copié sur son hôte désigné. Dans l'exemple représenté dans la figure 5.13a, deux conteneurs relatifs à la température et à la consommation électrique sont ainsi ajoutés sur le micro-contrôleur et le réseau, respectivement.

Les enregistrements et annonces nécessaires sont effectués, avant la redirection effective des applications les utilisant. Dans le cas de la configuration illustrée, le conteneur de consommation électrique est annoncé au smartphone et au compteur intelligent comme montré dans la figure 5.13a. Les relations d'écriture et de lecture vers les conteneurs initiaux sont supprimés (voir figure 5.12a) et ré-établis vers les nouveaux conteneurs (voir figure 5.13a).

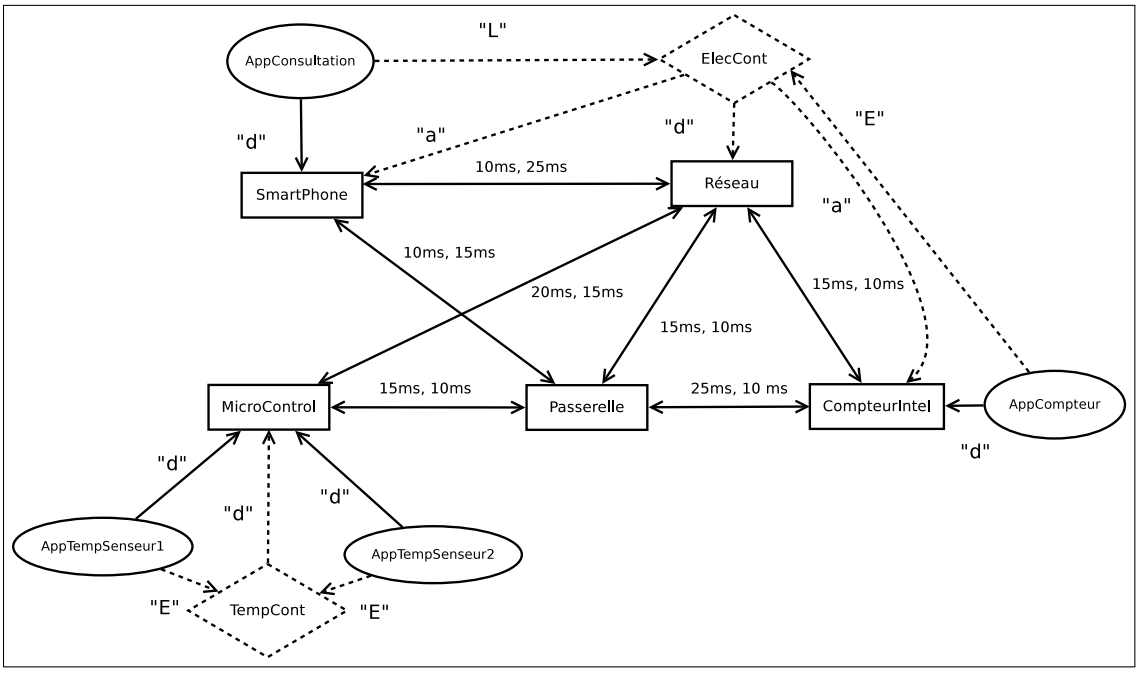
Les conteneurs initiaux sont finalement supprimés, comme précisé dans la figure 5.12a.

Chaque transformation est impactée sur la couche fonctionnelle dès qu'elle est effectuée. Le modèle basé sur le standard ETSI subit donc également des modifications dont le résultat est représenté dans la figure 5.13b.

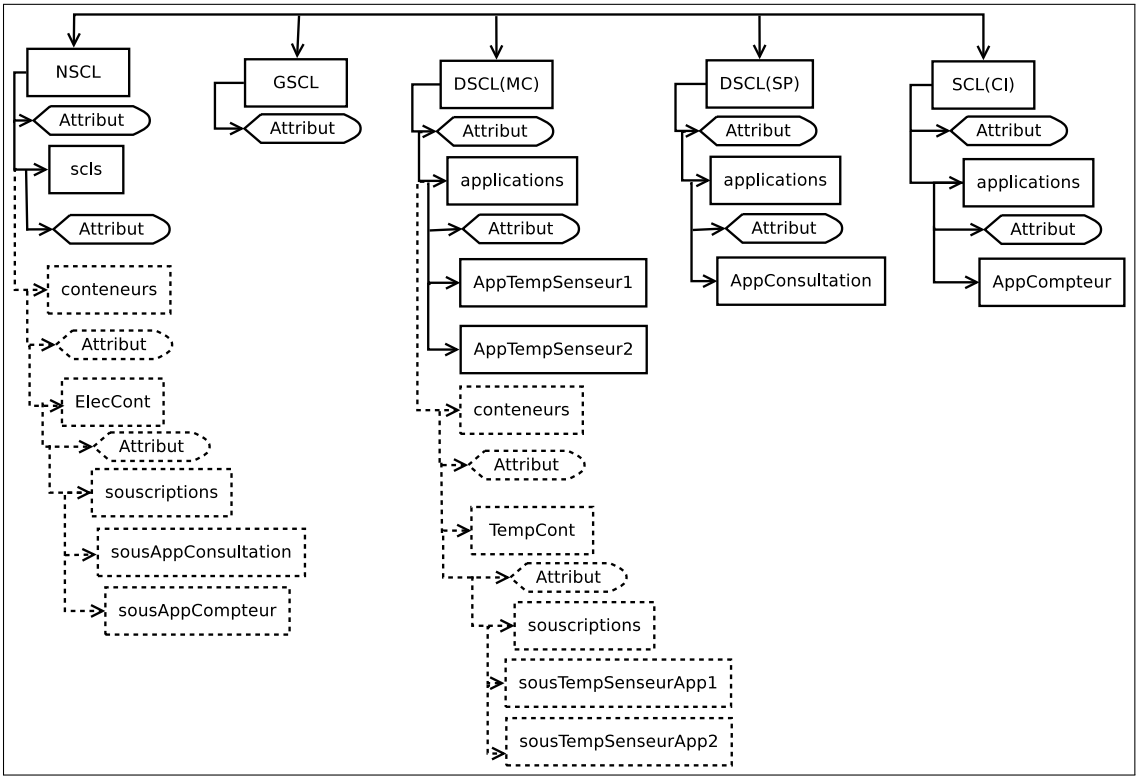
5.5.3 Résultats expérimentaux

Les transformations de graphes utilisées dans les politiques de reconfigurations ont été implémentées et exécutées en utilisant l'outil GMTE [6]. Les expériences ont été effectuées sur une machine possédant un processeur Intel Xeon à deux cœurs (3.2 GHz, 6M cache) et 8 Go de RAM.

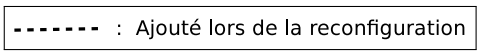
Ces expériences consistent à générer l'événement déclenchant le scénario précédemment décrit et observer son déroulement. Cela constitue une première forme de validation, la politique ayant toujours été menée à bien comme prévu. Afin de tester l'extensibilité de l'approche, l'expérience a été conduite sur trois différentes configurations initiales. Le nombre de conteneurs déployés sur la passerelle, utilisé par un nombre d'applications variant de 4 à 50, évolue entre 2 et 3. Les temps d'exécution de la politique autonome sur ces diverses configurations sont présentés dans le tableau 5.1.



(A) Couche de gestion : Représentation basée graphe



(B) Couche fonctionnelle : Arbres de ressources



(C) Légende

FIGURE 5.13 – Configuration finale minimale

	Conteneurs	Applications	Temps d'exécution (ms)
Configuration 1	2	4	850
Configuration 2	3	10	1531
Configuration 3	3	50	12655

TABLE 5.1 – Résultats expérimentaux

Évolution du temps d'exécution. De manière prévisible, le temps croît avec le nombre de conteneurs et d'applications. En effet, l'application d'une règle de transformation et les politiques autonomes reposent sur la recherche de morphisme, un problème dépendant de la taille du graphe. En plus, un plus grand nombre de conteneurs implique un plus grand nombre de migrations (et donc plus de transformations). Similairement, plus d'applications entraînent plus de redirections.

Valeurs obtenues. Les temps d'exécution sont de l'ordre de la seconde, mais augmentent rapidement jusqu'à une dizaine de secondes dans le dernier cas. Ces résultats de prime abord élevés doivent être mitigés par les faits suivants :

— le cas où 50 applications utilisent simultanément des conteneurs déployés sur le même équipement (hors réseau) est extrêmement improbable. Il a été traité dans un cadre purement théorique afin de tester l'extensibilité de la solution

— le temps d'exécution de la politique autonome comprend la prise de décision optimisant la configuration résultante, la transformation des deux modèles incluant la redirection des applications, ainsi que la garantie d'un résultat correct de sorte qu'aucun roll-back ne puisse être nécessaire.

— comme la concurrence d'événements et la parallélisation de leur traitement n'ont pas été abordées dans cette étude, la reconfiguration est conduite sur un seul thread, bien que chaque conteneur et chaque redirection d'application puissent être traités simultanément.

5.6 Conclusion du chapitre

Ce chapitre illustre l'intégration du modèle introduit précédemment au sein d'une **vue architecturale multi-modèles intégrant des problématiques de cohérence interne** et démontre son adéquation vis à vis de l'implémentation de la propriété d'auto-guérison. Pour ce faire, **nous concevons un gestionnaire autonome apte à piloter tout système M2M se conformant au standard ETSI M2M** qui a fait l'objet de deux publications [63, 133].

L'ETSI est un des seuls instituts de standardisation proposant un standard de bout en bout mature pour les systèmes M2M. Celui-ci facilite le déploiement d'applications verticales et l'innovation inter-industries en exposant données et services au travers d'interfaces standardisées. Les travaux présentés dans ce chapitre ont été effectués dans le cadre du projet OM2M, qui vise à fournir une implémentation open-source de ce standard. Les systèmes M2M évoluant dans un contexte changeant, le standard prévoit une ressource "gestionnaire autonome" chargée de mener à bien adaptations et évolutions. Il ne fournit néanmoins aucune précision à ce sujet, notamment sur la manière de le concevoir et de l'implémenter.

Il est tout d'abord nécessaire de définir la représentation du système sur laquelle repose le gestionnaire autonome. La représentation sous-jacente au standard ETSI M2M n'étant pas appropriée pour la spécification de politiques d'auto gestion, nous associons ses concepts à une formalisation

des systèmes M2M basée sur les graphes. La vue architecturale résultante bénéficie de chacune des représentations, conciliant ainsi fonctionnalités et gestion autonome. La multiplicité des modèles pose le problème de leur cohérence, des évolutions pouvant en plus trouver leur origine dans chacune des couches à travers les processus de découvertes et de reconfigurations. Pour répondre à cette problématique, nous définissons un mécanisme de communications et mises à jour bi-directionnelles assurant la cohérence interne de la vue architecturale proposée, et donc des deux modèles qui la composent.

Cette représentation a permis l'**élaboration de diverses politiques d'auto-gestion** dont un aperçu est fourni dans ce chapitre. Nous détaillons en particulier deux politiques permettant de prévenir des saturations du réseau ou des pertes de données. Elles rentrent ainsi dans le cadre de l'auto-guérison pro-active du système en incluant des éléments d'auto-optimisation. Élaborées conformément à la méthodologie conçue dans le chapitre précédent, elles respectent de plus la propriété d'auto-protection interne du système.

Finalement, le gestionnaire proposé est intégré à un système de compteur intelligent en simulation. Une première **expérimentation démontre la faisabilité de l'approche**. Elle consiste à générer un événement déclenchant une des politiques autonomes définies, et observer son déroulement. Les résultats de cette étude préliminaire sont toutefois mitigés quant à l'extensibilité de l'approche. Ils appellent à une étude plus approfondie de l'extensibilité de la transformation de graphe et de son application à la gestion autonome, qui sera conduite dans le chapitre suivant.

Chapitre 6

Extension des systèmes de réécriture

Application à l'auto-configuration d'architectures dynamiques

6.1 Introduction

La propriété d'auto-configuration [1] permet à un système de se paramétrer et se mettre à jour. Cette vaste propriété embrasse un large panel de considérations et procédures ; installation, démarrage, désinstallation et arrêt de composants logiciels, allocation de ressources... Deux types de problématiques se dégagent alors.
Premièrement, la conduite des actions ayant trait au système réel. Il s'agit d'opérations typiquement équivalentes à l'appel et à l'exécution d'une méthode ou d'un script. Le démarrage et l'enregistrement d'un composant logiciel, par exemple, entrent dans cette catégorie.
Deuxièmement, les considérations relatives à la représentation du système inhérente à la base de connaissance. Il s'agit d'y inclure les actions de la catégorie précédente et d'y modéliser, paramétrer et mettre à jour ses propriétés.

Nous intéressant toujours, et en particulier, à la représentation du système, nous traitons dans ce chapitre cette dernière problématique. Pour ce faire, nous élaborons une extension des systèmes de réécriture de graphe pour la prise en compte de domaines logiciels complexes.

Ce chapitre est articulé autour d'exemples concrets reprenant l'application DIET. La section 6.2 décrit dans un premier temps des propriétés et contraintes dont l'expression facilite l'évaluation et la gestion de cette application. Il est alors montré que les méthodes existantes basées sur les graphes possèdent des lacunes vis à vis de l'expression et de la mise à jour de telles propriétés et contraintes.

Une extension formelle des systèmes de réécriture de graphe est ensuite proposée dans la section 6.3 afin de lever ces restrictions. Cette extension est construite autour de l'intégration de contraintes et de mutateurs, ces derniers constituant une méthode légère pour la modification des caractéristiques du système.

Ce modèle amélioré est utilisé dans la section 6.4 afin de caractériser DIET de manière à prendre en compte toutes les considérations exprimées lors de son introduction. L'adéquation de cette représentation vis à vis de l'évaluation et de la gestion de DIET est alors illustrée.

1. en anglais : *self-configuration*

En particulier, les caractéristiques du système sont plus facilement manipulables grâce à la combinaison de techniques d'évaluation à la demande, et de mise à jour à chaud lors de modifications. Une propriété peut être évaluée lorsque sa valeur doit être connue. Afin d'éviter de trop fréquents calculs, cette valeur peut également être gardée en mémoire et mise à jour lorsqu'elle change. La pertinence relative de ces deux options dépend de la complexité et de la fréquence des mises à jour et des évaluations.
Ensuite, cette représentation permet de rapidement juger une configuration et d'identifier les objectifs pouvant être améliorés ainsi que les composants impliquant des viols de contraintes. La gestion du système et de ses évolutions en est d'autant facilitée.

Finalement, l'approche proposée est extensivement évaluée expérimentalement dans la section 6.5. Une étude de l'exécution d'un scénario de reconfiguration avec différents outils et méthodes y est proposée. Les résultats expérimentaux dénotent un gain significatif en terme d'efficacité et d'extensibilité en faveur de la solution proposée.

6.2 Exemple illustratif et énoncé du problème

6.2.1 DIET : contraintes et critères d'évaluation

Les limitations des approches de réécriture existantes, ainsi que les concepts introduits afin d'y répondre, sont une nouvelle fois illustrés par l'intermédiaire de DIET. Nous proposons ici une vision plus complète et complexe de l'application DIET introduite dans la section4.2.2, en y intégrant des contraintes ainsi que des critères fonctionnels et non fonctionnels facilitant l'évaluation d'une configuration. Une telle évaluation permet de juger de la qualité d'une configuration relativement à une autre. Ce processus est à la base des solutions d'(auto-)optimisation.

6.2.1.1 Contraintes structurelles et applicatives

Exception faite des liaisons vers l'OMNI et de celles entre MAs, l'architecture globale présente une structure de forêt (i.e., un ensemble d'arbres). On considère dans un premier temps une architecture simplifiée comportant un seul MA. De plus :

1. Lors de son déploiement, chaque composant est déclaré à l'OMNI.
2. Chaque LA et chaque SED a exactement un supérieur hiérarchique, son nœud parent dans le sous arbre du graphe.
3. Chaque MA et chaque LA gèrent au minimum $minFilsMA$/$minFilsLA$ et au maximum $maxFilsMA$ / $maxFilsLA$ composants. Dans l'illustration des concepts, ces valeurs sont fixées à 1 et 10, respectivement.
4. Du fait d'hypothétiques restrictions logicielles et d'un nombre limité de machines disponibles, cette architecture ne peut comporter plus de $maxAgents$ agents, dont la valeur est fixée à 100 dans les exemples.

6.2.1.2 Critères d'évaluation des configurations

Les propriétés introduites précédemment constituent des contraintes à coupler aux besoins nés des requêtes clients. Naturellement, bien qu'une configuration satisfasse ces prérequis à un instant

donné et dans un contexte particulier, une autre peut les satisfaire d'une "meilleure manière". C'est là toute la problématique sous-jacente à la notion d'(auto-)optimisation liée aux contraintes non-fonctionnelles. En particulier, considérons les trois critères suivants :

1. la consommation énergétique. Conformément aux travaux de Borgetto et al. [134], elle est supposée dépendre uniquement du nombre de machines utilisées et du nombre de composants logiciels déployés.
2. la robustesse, c'est à dire la tolérance aux pannes matérielles ou logicielles. La robustesse est un critère non fonctionnel. Elle est ramenée à trois sous-critères fonctionnels se référant, pour chaque service s, à l'ensemble S_s des SEDs le proposant :
   (a) degré de redondance (i.e., cardinalité de S_s). Il s'agit du nombre de SEDs proposant le service s. Plus celui-ci est élevé, moins s est vulnérable aux pannes logicielles affectant un ou plusieurs SED(s).
   (b) dispersion physique (i.e., nombre de machines sur lesquelles un SED $\in S_s$ est déployé). Plus celui-ci est élevé, moins s est vulnérable aux pannes matérielles affectant une machine sur laquelle au moins un SED de S_s est déployé.
   (c) dispersion logique des SEDs au sein de la structure hiérarchique arborescente et étalement physique des LAs sans lien de descendance. Il s'agit ici premièrement de réduire le nombre d'ancêtres communs à chaque couple de SEDs de S_s. Cela réduit la vulnérabilité de s face à une panne logicielle d'un LA. Deuxièmement, il faut que les LAs n'ayant pas de relation de descendance (c'est à dire n'appartenant pas à un même chemin du MA vers un SED) soient déployés sur des machines distinctes. Ainsi, s est plus robuste face à une panne matérielle affectant une machine sur laquelle est déployé un ancêtre d'au moins un SED de S_s.
3. la qualité de service. Celle-ci est évaluée au regard de l'équilibre des charges sur les nœuds de même profondeur. Notons $LA(d)$ *l'ensemble des LAs de profondeur* d, et *M(c) le nombre d'entités gérées par le composant c*. Le critère présent est relatif à la satisfaction de la condition dite d'équilibrage exigeant que, pour toute profondeur d, l'écart type de l'ensemble $\bigcup_{la \in LA(d)} M(la)$ ne dépasse pas un seuil noté $max\sigma$.

Il est intéressant de noter que la robustesse et la consommation énergétique sont des critères concurrents ; déployer plus de composants logiciels ou utiliser plus de machines va à la fois améliorer le premier et dégrader le second.

6.2.1.3 Configuration de l'application

En plus de ces nouvelles informations devant être représentées et gérées, chaque agent doit être configuré lors de son déploiement. Pour ce faire, il est lié à un fichier de configuration contenant les informations suivantes :

1. Son type.
2. L'adresse à laquelle l'agent peut être contacté, formée de la machine sur laquelle il est déployé et de son nom (identifiant unique).
3. Pour un LA ou un SED, le nom de l'agent père.
4. Pour un SED, la taille du buffer d'envoi de message. Celui-ci dépend directement des services qu'il fournit.

Notation	Signification
$LA(d)$	l'ensemble des LAs de profondeur d
$M(c)$	le nombre d'entités gérées par le composant c
$maxSonsMA$ $maxSonsLA$	le nombre maximum d'entités gérées par le MA ou un LA, respectivement
$minSonsMA$ $minSonsLA$	le nombre minimum d'entités gérées par le MA ou un LA, respectivement
$max\sigma$	seuil de la condition d'équilibrage

TABLE 6.1 – Récapitulatif des notations introduites.

6.2.1.4 Informations nécessaires

Le modèle représentant DIET doit être adapté afin de permettre sa configuration et l'évaluation des trois critères précédemment introduits de manière simple et efficace durant l'exécution. En particulier, les informations relatives à ces critères doivent être facilement accessibles. Il est par conséquent nécessaire de considérer les attributs suivants :

1. Un identifiant unique pour chaque composant.
2. La nature de chaque composant.
3. La profondeur de chaque LA.
4. Le nombre d'entités gérées par chaque composant c de type LA ou MA : $M(c)$,
5. L'ensemble des services fournis par chaque SED et chaque LA. L'ensemble des services fournis par un LA est égal à l'ensemble des services fournis par au moins un SED descendant du dit LA.
6. La machine sur laquelle chaque entité est déployée.

Les notations introduites dans cette première sous-section sont résumées dans le Tableau 6.1.

6.2.2 Énoncé du problème

Par la suite, nous décrivons les limitations et problèmes des approches classiques de réécriture de graphe au regard de la modélisation de DIET et de son auto-configuration interne.

6.2.2.1 Interdépendance d'attributs

Un attribut d'un nœud v peut dépendre d'attributs relatifs à un ensemble S d'autres entités. Cette problématique est particulièrement liée à la phase de déploiement/génération : les éléments de S peuvent être déployés avant ou après v.

Similitude avec un problème classique. Une distinction similaire se retrouve dans les grammaires (attribuées) de chaînes de caractères classiques. Les attributs y sont classifiés en deux catégories. Ils sont qualifiés d'hérités si les éléments de S sont des parents ou des frères de v dans l'arbre syntaxique abstrait découlant de l'analyse de la phrase. Dans le cas contraire, ils sont dits synthétisés. Les valeurs d'attributs synthétisés ne peuvent être connues dans le contexte où ils apparaissent pour la première fois : elles dépendent de futures applications de règles de production. Dans les grammaires de graphes, ces règles symbolisent traditionnellement l'addition d'un nouveau composant logiciel.

De même, les attributs de graphes ne peuvent être gérés de la même manière suivant qu'ils dépendent d'attributs d'entités existantes ou non.

Attributs hérités. Le premier cas peut facilement être traité par l'héritage d'attribut dans les grammaires de graphes. Par exemple, au moment du déploiement, la profondeur d'un LA peut aisément être déduite de la profondeur de son père.

Attributs synthétisés. L'héritage ne peut s'appliquer au second cas. L'ensemble des services fournis par un LA, par exemple, ne peut être connu au moment de son déploiement puisqu'il dépend de nœuds descendants qui seront ajoutés ou supprimés par la suite. Pour répondre à cette problématique, les attributs peuvent être représentés de deux manières différentes.

Premièrement, les attributs peuvent être définis par, et associés à, leurs expressions analytiques dont la valeur est calculée soit à la demande soit systématiquement lors de chaque transformation. Le processus d'évaluation peut cependant être chronophage, voire inutile, lorsque la valeur de l'attribut n'a pas changé depuis la dernière évaluation, par exemple. En outre, contrairement à ce qui se passe dans les grammaires classiques, les transformations ne sont pas appliquées uniquement lors d'un phase de construction/déploiement. Ici, elles décrivent aussi les évolutions et reconfigurations du système. L'évaluation systématique lors de chaque transformation ne peut ainsi pas être prise à la légère.

La seconde solution consiste à associer directement un attribut à sa valeur actuelle. Elle est alors mise à jour lorsque nécessaire. Celle solution est immédiatement liée à la modification de la valeur d'un attribut existant, concept discuté dans la suite.

6.2.2.2 Modification de la valeur d'un attribut existant

Comme discuté dans l'état de l'art 2.3.3, les approches classiques de réécriture de graphe permettent à une règle de modifier uniquement les attributs y apparaissant.

En présence d'inter-dépendances, une telle modification doit être propagée aux attributs dépendants. Ceci peut entraîner un grand nombre d'applications de règle(s). Par exemple, lors du déploiement d'un SED sous un LA, l'ensemble des services fournis par ce dernier doit être mis à jour en conséquence. En fait, cette mise à jour doit être récursivement impactée sur chaque ancêtre du LA, jusqu'à atteindre le MA ou un LA dont l'ensemble de services reste inchangé (car il fournissait déjà tout service proposé par le nouveau SED).

Dans le scénario précédemment décrit, et puisqu'une modification ne peut être menée à bien que sur un attribut apparaissant explicitement dans une règle, il y a autant d'applications que de LAs modifiés. Ce phénomène "domino" implique une grande perte d'efficacité et d'élasticité.

Dans les grammaires de chaînes de caractères classiques, ce type de modification peut être effectuée grâce à l'adjonction, dans les règles de réécritures, d'algorithmes permettant la mise à jour d'attributs appelés mutateurs [135].

6.2.2.3 Évaluation d'une configuration : intégration de contraintes

Des contraintes analytiques et heuristiques peuvent refléter les exigences fonctionnelles et non fonctionnelles du système, respectivement. Elles permettent en particulier la description de seuils. Leur satisfaction ou violation permet alors d'évaluer rapidement une configuration. Leur violation peut être détectée et gérée par un gestionnaire autonome sans nécessiter de prise de décision complexe et sans analyser l'intégralité du système.

Les contraintes, directement liées au système et à ses composants, et les attributs présentent des similitudes ; elles dépendent d'attributs, évoluent au cours du temps, et les composants de même types ont, en plus des mêmes attributs/propriétés, les mêmes contraintes. Leur intégration soulève trois problématiques discutées ci-après.

Style intégrant des contraintes ou style contraint. Il est crucial de faire la distinction entre intégration des contraintes au modèle, au style architectural, et contraindre un style. Les approches existantes se basant sur des graphes se restreignent souvent à ce second cas, contraignant un style architectural dans le seul but de réduire le spectre des états corrects. Les contraintes sont alors intégrées au modèle du style, par exemple au graphe type dans [62], et non aux configurations elles-mêmes.

Cette différence est lié à la considération de contraintes "fortes" ou heuristiques. Dans le premier cas, il s'agit de chercher une configuration efficace parmi celles respectant toutes les contraintes (e.g., [136]). Dans le second, leur satisfaction/non-satisfaction est intégrée au sein de la fonction d'évaluation caractérisant la notion d'efficacité : la violation d'une contrainte par une configuration ne la rend alors pas théoriquement inacceptable, mais plutôt moins efficace.

Inconnues dans les contraintes. Dans l'écrasante majorité des approches de la littérature, des attributs variables dont la valeur est inconnue ne peuvent intervenir que dans des règles de transformations. Ils sont exclus du graphe représentant le système. Lorsque considérées dans de telles approches, les contraintes ne prennent par conséquent pas en compte ce type d'attribut.

Évaluation des contraintes. Les contraintes sont généralement soit gérées comme des post-conditions à une transformation, soit exprimées via leurs expressions analytiques et évaluées à la demande ou systématiquement après chaque transformation. La première solution écarte d'emblée l'existence de contraintes heuristiques. La seconde est exactement celle proposée classiquement pour la gestion des attributs dit "synthétisés". De manière similaire, nous proposons d'introduire une troisième alternative consistant à mettre à jour la valeur d'une contrainte lorsqu'elle est modifiée.

Objectifs et solutions. Eu égard aux similitudes entres attributs et contraintes, il est pertinent d'intégrer ces dernières d'un manière analogue aux attributs.

Cela permet premièrement la construction d'un ensemble de contraintes lors de la génération, du déploiement ou de la transformation d'une configuration.

Deuxièmement, les contraintes bénéficient ainsi des fonctionnalités introduites dans ce chapitre pour la gestion des attributs. Elles sont alors faciles à manipuler et évaluer.

6.2.2.4 Bilan

Ces trois points (attributs inter-dépendants, modification d'un attribut existant et intégration de contraintes) soulignent les limitations des approches classiques de modélisation basées sur les graphes vis à vis de leur propre configuration. Ils justifient la nécessité de l'extension proposée dans ce chapitre.

L'expressivité du modèle de la base de connaissance, et en particulier la flexibilité avec laquelle il peut être configuré et modifié, contraint à son tour l'évaluation et la configuration du système lui-même. Cette dernière peut typiquement être effectuée via l'appel de scripts et méthodes dont les arguments sont déduits des attributs du modèle. Le dernier objectif de ce chapitre est l'introduction, au sein de la réécriture de graphe, d'un cadre pour la représentation explicite de ces invocations.

6.3 Introduction de contraintes et de mutateurs dans les systèmes de réécriture de graphes

6.3.1 Attributs, contraintes et réécriture d'attributs

6.3.1.1 Attributs

Avant de s'intéresser à la manipulation des attributs du système, il convient tout d'abord de mieux les définir en cohérence avec les propositions du chapitre 3 adoptées jusqu'ici. Il s'agit de les adapter, en conservant la simplicité et l'efficacité de "lister" les attributs comme des labels, afin de permettre l'application flexible d'opérateurs algébriques.

Conventions et notations.
La notation canonique où, pour tout couple d'ensemble (X, Y), Y^X désigne l'ensemble des applications de X vers Y est adoptée.

Définition 6.1. (Attribut) Un attribut est un couple $\mathtt{Att} = (A, D)$ où :
- A est sa valeur qui est soit :
  - une variable.
  - une constante dans D.
  - une expression d'une (S, OP)-algèbre [87], où (S, OP) est une signature algébrique infinie avec S un ensemble incluant D et $OP = (D)^{S^+}$.
- et D est son domaine de définition.

6.3.1.2 Contraintes

Les attributs sont un moyen d'exprimer des informations sur une structure algébrique. Ceux-ci admettant des interdépendances, les contraintes peuvent elles-mêmes être vues comme des attributs particuliers.

Définition 6.2. (Contraintes) Une contrainte Cons est un attribut (A_{Cons}, D_{Cons}) où D_{Cons} = {"vrai", "faux", "inconnu"}.

Remarque 6.1. Une contrainte est une expression d'une logique ternaire dans le sens classique du terme. Considérer une logique ternaire plutôt que binaire implique qu'au contraire des attributs, les contraintes peuvent toujours être évaluées. Cette évaluation peut avoir lieu en affectant à toute expression logique minimale ne pouvant être évaluée, du fait par exemple de la présence d'un attribut dont la valeur est inconnue, la valeur "inconnu".

Conventions et notations.
Comme toutes les contraintes partagent le même domaine de définition, il sera implicite dans la suite : une contrainte Cons = (A_{Cons}, D_{Cons}) sera simplement désignée par A_{Cons}.
Nous adoptons ici les principes de la logique forte de Kleene [137, 138], en particulier ses opérations logiques de bases ($\vee, \wedge, \neg, \Rightarrow$) et le fait que la seule valeur de vérité soit "vrai". L'unicité de cette dernière implique que les évaluations sont pessimistes, i.e. "inconnu" est supposé faux.

Contraintes et attributs sont représentés dans deux ensembles distincts. Afin d'alléger les notations, un objet partiellement attribué et contraint, i.e., un triplet composé d'un objet et de deux familles d'attributs et de contraintes est appelé un AC-objet. Lors de la définition d'un AC-object constitué d'AC-éléments, deux seules familles indexées par les éléments attribués et contraints seront considérées (plutôt que de manipuler de multiples ensembles de contraintes et attributs).

Définition 6.3. (AC-objet) Un AC-objet est un triplet (obj, ATT, CONS) où obj désigne l'objet lui-même tandis que ATT et CONS sont deux familles d'attributs et de contraintes indexées par obj et/ou tout constituant de obj étant attribué ou contraint, respectivement.

6.3.1.3 Réécriture d'attributs

Il s'agit ici de transposer les mutateurs, une solution légère pour la modification d'attributs présentée dans le paragraphe 6.2.2.2, aux grammaires de graphes.

Ce mécanisme peut également être exploité afin d'expliciter l'appel à des méthodes et scripts externes lors d'une transformation. Ces derniers peuvent avoir un vaste champ d'action : le système réel (e.g., dans le cadre de sa configuration), sa représentation (e.g., les communications inter-couches abordées dans la section 5.3.3 du chapitre précédent)... Ils doivent néanmoins laisser l'objet transformé (dans notre cas, l' AC-graphe) inchangé.

Définition 6.4. (Mutateur sur un AC-objet) Un mutateur sur un AC-objet est un algorithme arbitraire :

- mettant à jour la valeur d'aucun, un ou plusieurs attributs et contraintes. Un tel mutateur est qualifié d'**interne**.
- n'ayant aucun impact sur l'objet lui-même. Un tel mutateur est qualifié d'**externe**.

Un mutateur interne est limité à la modification de valeurs ; il ne peut modifier ni le nombre d'attributs et de contraintes, ni leurs domaines de définition. Un mutateur externe correspond typiquement à des appels de méthodes externes.

6.3.2 Représentation des configurations : AC-graphes

6.3.2.1 Définition

Un AC-graphe est défini à partir d'un graphe dirigé partiellement attribué présenté dans la définition 3.1. Un ensemble de contraintes y est simplement ajouté en adoptant les conventions de notation relatives aux attributs (Con^{i}_{el} représente la i-ème contrainte de l'élément el). En outre, la possibilité est donnée aux attributs et contraintes de référer au graphe lui-même en plus de ses éléments.

Définition 6.5. (AC-graphe) Un AC-graphe G est défini comme un système $(V, E, Att, Cons)$ où :

1. V et $E \subseteq V^2$ correspondent respectivement aux ensembles des nœuds et arcs du graphe, de sorte que (v_1, v_2) désigne un arc de v_1 vers v_2.
2. Att et $Cons$ sont des familles d'attributs et de contraintes indexées par deux sous-ensembles de $V \cup E \cup \{G\}$.

6.3.2.2 Représentation d'une configuration contrainte de DIET

Avant de nous intéresser aux styles architecturaux, cette sous-section offre un premier aperçu de la représentation visée pour une configuration de DIET. Une partie des préoccupations exposées dans la section 6.2 est exprimée à l'aide des concepts introduits dans la section 6.3. La figure 6.1 illustre un AC-graphe représentant une configuration de DIET. Afin d'en améliorer la lisibilité, l'OMNI n'y est pas représenté.

Notations. Les notations présentes dans la figure 6.1 sont récapitulées dans le tableau 6.2.

Notation	Signification
Mach	l'ensemble des machines disponibles
ID	l'ensemble des identifiants possibles
Nat	l'ensemble des natures possibles d'un composant logiciel
Link	l'ensemble des relations possibles entre composants logiciels
S	l'ensemble des services pouvant être assurés
Serv	l'ensemble des parties de S
Red	la contrainte de redondance
Loc	la contrainte de dispersion physique

TABLE 6.2 – Notations relatives à la caractérisation d'une configuration de DIET

Dans le cas de l'architecture étudiée, Nat, l'ensemble des natures possibles d'un composant logiciel, est égal à {"OMNI", "MA", "LA", "SED"}. Lien, référant à l'ensemble des relations possibles entre composants, est ici {"ma2o", "ma2la", "ma2sed", "la2o", "la2la", "la2sed", "sed2o"}, où "o" fait référence à l'OMNI.

Red et Loc, les contraintes de redondance et de dispersion physique, sont détaillées dans le paragraphe dédié aux contraintes.

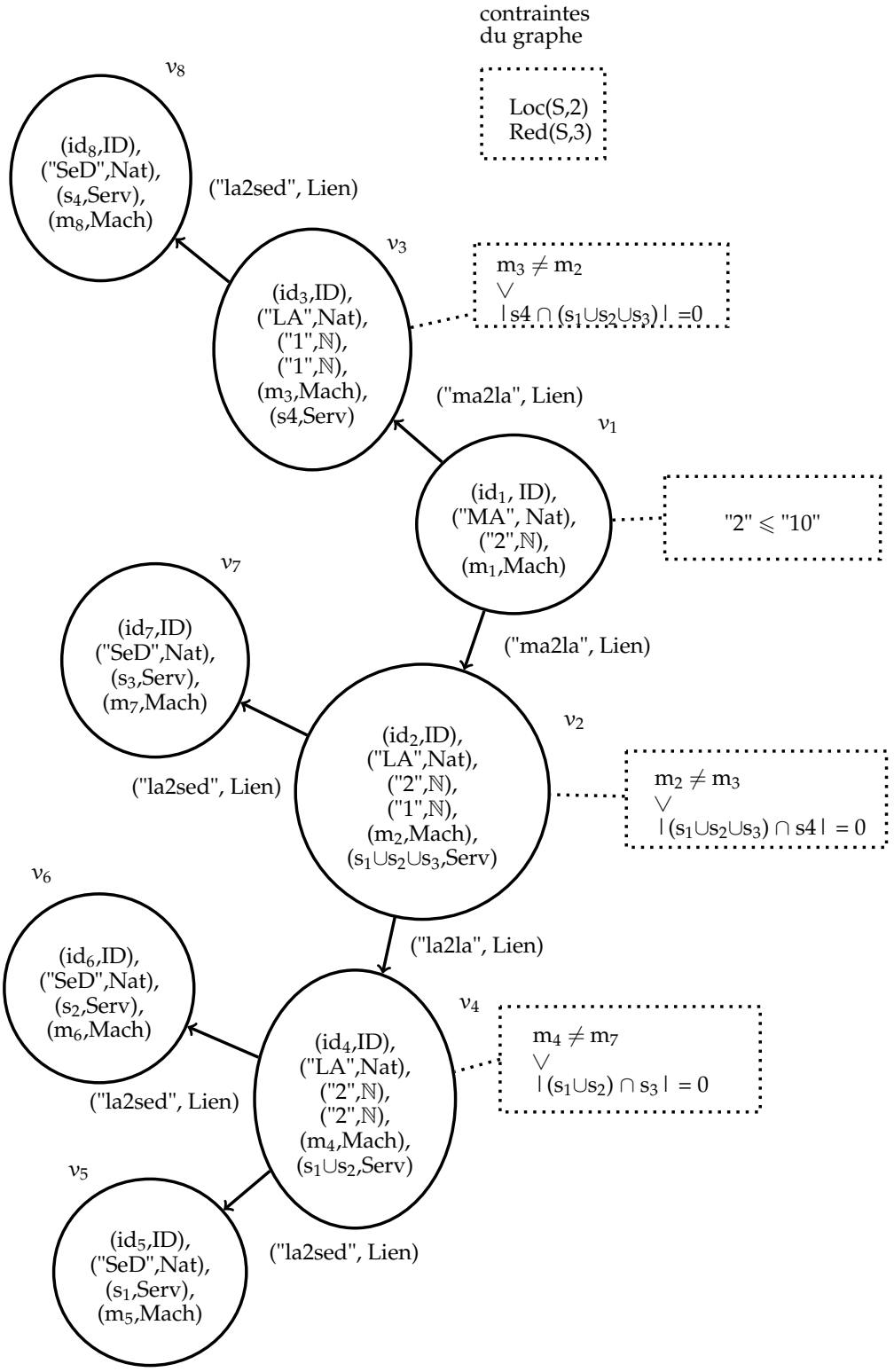


FIGURE 6.1 – Un AC-graphe représentant une configuration de DIET

Description de la configuration. L'application est ici composée de huit composants symbolisés par huit nœuds et leurs relations modélisées par des arcs, tous attribués afin de refléter leurs diverses propriétés. Les composants de même nature ont les mêmes types d'attributs, ceux identifiés dans la section 6.2. Certains composants, ainsi que le graphe lui-même, sont contraints afin de refléter les préoccupations décrites dans cette même section.

Les contraintes sont représentées à l'intérieur d'un cadre en pointillé, et liées à l'objet qu'elles contraignent par une ligne en pointillé, exceptées celles relatives au graphe lui-même.

Attributs. Le premier attribut de chaque nœud représente l'identifiant unique, ou nom, du composant qu'il représente. Le second, dont le domaine de définition est Nat, reflète sa nature. La configuration comprend un MA nommé id_1. Il gère deux entités et est déployé sur la machine m_1, comme décrit par ses troisième et quatrième attributs, respectivement.

Chaque LA possède deux attributs de plus, liés à sa profondeur et l'ensemble de services qu'il fournit. Dans l'exemple, trois LAs sont déployés. Ils sont respectivement :
- représentés par v_2, v_3 et v_4.
- nommés id_2, id_3 et id_4.
- de profondeur 1, 1 et 2.
- pères de 2, 1 et 2 entités.
- placés sur les machines m_2, m_3 et m_4.
- capables d'accéder aux services $s_1 \cup s_2 \cup s_3$, s_4 et $s_1 \cup s_2$.

Finalement, quatre SEDs déployés sur m_5, m_6, m_7 et m_8 fournissent les services s_1, s_2, s_3 et s_4, respectivement.

Contraintes. Le graphe est contraint par deux clauses Loc(S,2) et Red(S,3), reflétant le besoin de redondance et de dispersion physique des services fournis pour améliorer la robustesse du système.

Notation	Arguments	Signification
Red	$S_S \subseteq S$ $n \in \mathbb{N}$	Tout service s de S_S est fourni par au moins n SEDs.
Loc	$S_S \subseteq S$ $n \in \mathbb{N}$	Tout service s de S_S est fourni par des SEDs déployés sur au moins n machines .

TABLE 6.3 – Contraintes du graphe

Pour tout sous ensemble S_S de S et tout entier naturel n, la contrainte de redondance, notée Red(S_s, n), affirme que "Tout service s de S_S est fourni par au moins n SEDs". Pour un graphe $G = (V, E, ATT, CONS)$, elle est formellement définie par :
$Red(S_S,n) = \forall s \in S_S, \exists (v_i)_{i\in[1,n]} \in V^n, (\forall (i,j) \in [1,n]^2, i \neq j \Rightarrow v_i \neq v_j) \wedge (\forall k \in [1,n], A^2_{v_k}$ = "SED" $\wedge\, s \in A^3_{v_k})$.

Pour tout sous ensemble S_S de S et tout entier naturel n, la contrainte de dispersion physique, notée Loc(S_S,n), affirme que "Tout service s de S_S est fourni par des SEDs déployés sur au moins n machines". Pour un graphe $G = (V, E, ATT, CONS)$, elle est formellement définie par :
$Loc(S_S,n) = \forall s \in S_S, \exists (v_i)_{i\in[1,n]} \in V^n, (\forall (i,j) \in [1,n]^2, i \neq j \Rightarrow (v_i \neq v_j \wedge A^4_{v_i} \neq A^4_{v_j})) \wedge$ $(\forall k \in [1,n], A^2_{v_k}$ = "SED" $\wedge s \in A^3_{v_k})$.

Le tableau 6.3 propose un récapitulatif de ces deux contraintes. Eu égard à la configuration considérée, cela signifie que tout service de S devrait être fourni par au moins 3 SEDs déployés sur au moins 2 machines différentes.

Le troisième sous-critère de robustesse, la dispersion au sein de l'architecture logique, se traduit par la présence d'une contrainte sur chaque LA. Elle spécifie qu'un LA et ses nœuds frères ne doivent pas être déployés sur la même machine, à moins de ne pas donner accès à des services similaires. Cette contrainte assure que, au sein d'un sous arbre, les SEDs fournissant des services similaires (et déployés sur différentes machines grâce à la clause Loc) ne sont pas gérés par des entités elles-mêmes déployées sur la même machine. Ainsi, un plus grand nombre de machines doivent tomber en panne pour qu'un service ne puisse plus être proposé.

Finalement, le MA ne doit pas gérer plus de 10 entités, soulignant une des propriétés fondamentales du style architectural.

La cardinalité maximale d'entités gérées par les LAs pourrait être prise en compte de la même manière. Elle sera toutefois implémentée grâce à une contrainte de déploiement conditionnel. Cette technique sera également utilisée pour garantir l'équilibrage des charges relatif au critère de qualité de service.

Définition formelle. Le graphe de la figure 6.1 est formellement défini comme suit :

G = (V, E, ATT, CONS) où
V =$\{v_i | i \in [1,8]\}$,
E =$\{ e_1 = (v_1, v_2), e_2 = (v_1, v_3), e_3 = (v_2, v_4), e_4 = (v_2, v_7), e_5 = (v_3, v_8), e_6 = (v_4, v_5), e_7 = (v_4, v_6)\}$,
ATT =$\{ATT_{el} | el \in E \cup V\}$, avec :

ATT_{v_1} ={(id_1, ID), ("MA", Nat), ("2", $\mathbb{N}$), (m_1,Mach)}.
ATT_{v_2} ={(id_2, ID), ("LA", Nat), ("1", $\mathbb{N}$), ("2", $\mathbb{N}$), (m_2, Mach), ($s_1 \cup s_2 \cup s_3$, Serv)},
ATT_{v_3} ={(id_3, ID), ("LA", Nat), ("1", $\mathbb{N}$), ("1", $\mathbb{N}$), (m_3, Mach), (s_4, Serv)},
ATT_{v_4} ={(id_4, ID), ("LA", Nat), ("2", $\mathbb{N}$), ("2", $\mathbb{N}$), (m_4, Mach), ($s_1 \cup s_2$, Serv)}.
ATT_{v_5} ={(id_5, ID), ("SED",Nat), (s_1, Serv), (m_5, Mach)},
ATT_{v_6} ={(id_6, ID), ("SED",Nat), (s_2, Serv), (m_6, Mach)},
ATT_{v_7} ={(id_7, ID), ("SED",Nat), (s_3, Serv), (m_7, Mach)} et
ATT_{v_8} ={(id_8, ID), ("SED",Nat), (s_4, Serv), (m_8, Mach)}.

CONS =$\{CONS_G, CONS_{v_1}, CONS_{v_2}, CONS_{v_3}, CONS_{v_4}\}$, où :
$CONS_G$ ={Loc(S,2), Red(S,3)},
$CONS_{v_1}$ ={$A^3_{v_1} \leqslant maxSonsMA$},
$CONS_{v_2}$ ={$A^5_{v_2} \neq A^5_{v_3} \vee (A^6_{v_2} \cap A^6_{v_3} = \emptyset)$},
$CONS_{v_3}$ ={$A^5_{v_3} \neq A^5_{v_2} \vee (A^6_{v_3} \cap A^6_{v_2} = \emptyset)$} et
$CONS_{v_4}$ ={$A^5_{v_4} \neq A^4_{v_7} \vee (A^5_{v_4} \cap A^3_{v_7} = \emptyset)$}.

À présent, les notions relatives à la caractérisation du style architectural sont adaptées afin d'assurer la gestion des contraintes, des attributs interdépendants et de leur modification.

6.3.3 Recherche de motifs, transformations et grammaires.

Les ensembles d'identification ainsi que l'unification d'attributs et d'éléments ne sont pas impactés par l'extension proposée. Ils conservent donc leurs définitions initiales, nommément les définitions 3.2, 3.3 et 3.5, respectivement. Toutefois, du fait de l'introduction d'opérateurs et de relations entre attributs, les ensembles d'identification sont maintenant assimilables à des systèmes d'équations plutôt qu'à de simples relations d'équivalence. Ils sont ainsi consistants si le système d'équation correspondant admet au moins une solution.

Affectations, morphismes et compatibilités ne sont pas non plus touchés par la modification décrite dans ce chapitre, et conservent leurs définitions (3.6, 3.7 et 3.8, respectivement).

Remarque 6.2.

Les contraintes n'affectent pas la recherche de motifs et relations entre graphes. Elles interviennent différemment dans le processus de transformations. De même, seuls les attributs des éléments d'un graphe sont considérés dans ces recherches, indépendamment de la présence ou de l'absence d'attributs portant sur le graphe lui-même.

L'existence d'un morphisme cohérent est conditionné par la solvabilité d'un système d'équations portant sur des attributs. Précédemment, seul un ensemble d'égalités entre attributs était considéré. C'est également le cas dans la majorité des approches portant sur les graphes attribués [87, 92], bien que ces égalités puissent n'apparaître qu'implicitement (e.g., dans un morphisme des espaces d'attributs). C'est souvent la seule clause liée à l'applicabilité d'une règle de réécriture et ne reposant que sur des attributs.

Expansions et restrictions se basent toujours sur une compatibilité dont la définition n'est pas changée par l'intégration de contraintes. Elles gèrent les contraintes similairement aux attributs, à l'exception notable que les contraintes de graphes ou d'éléments intervenant dans la compatibilité (c'est à dire, présents dans le premier graphe et ayant une image dans le second) sont concaténées plutôt que "fusionnées".

Soit deux AC-graphes C-compatibles G_1 et G_2 avec $C = (f, I)$, et deux de leurs sous-graphes, $G_1^S = (V_1^S, E_1^S, Att_1^S, CONS_1^S)$ et $G_2^S = (V_2^S, E_2^S, Att_2^S, CONS_2^S)$ dont les éléments contraints sont respectivement EL_1^{cons} et EL_2^{cons}. (V, E, Att), le résultat de l'expansion $G_1^S \uparrow_C G_2^S$ ou de la restriction $G_1^S \downarrow_C G_2^S$, vérifie alors :

— $CONS^G = (CONS_1^S)^G \cup (CONS_2^S)^G$.

— $\forall el_2 \in EL_2^{cons}, \exists el_1 \in EL_1^{cons}, f(el_1) = el_2 \implies CONS^{el_2} = (CONS_1)^{el_2} \cup (CONS_2)^{el_2}$.

Inspiré par la théorie des langages classique [135], contraintes et mutateurs sont intégrés aux règles de réécriture de graphe.
Les conditions d'applicabilités présentées dans la définition 3.11 sont étendues à la considération d'un ensemble de contraintes, celui de la règle elle-même. Cet ensemble (potentiellement vide) peut être vu comme un ensemble de prédicats sémantiques ou de pré-conditions.
Les mutateurs complètent les effets de l'application d'une règle. Ils s'exécutent à la fin de cette application.

Théoriquement, tout formalisme de réécriture de graphes peut être augmenté afin d'inclure prédicats sémantiques et mutateurs. Cette extension est décrite dans la définition suivante.

Définition 6.6. (AC-règle de réécriture de graphes) Une AC-règle de réécriture de graphe r_{ac} (incluant des mutateurs) est un quadruplet (r, ATT, CONS, ACT), où r est une règle de réécriture, ATT

et CONS les ensembles d'attributs et contraintes de la règle, respectivement, et ACT un ensemble de mutateurs.

Applicabilité. Une AC-règle de réécriture r_{ac} est applicable à un graphe G selon un morphisme m si

- r est applicable à G selon m.
- tout contrainte $c \in$ CONS est évaluable à "$vrai$" en intégrant l'affectation obtenue via m et en affectant à chaque expression logique élémentaire (i.e., atomique) ne pouvant être évaluée (du fait par exemple d'un attribut variable dont la valeur est inconnue) la valeur "$inconnu$", conformément à la remarque 6.1.

Application. L'application de r_{ac} à G selon m consiste à :

- appliquer r à G selon m.
- appliquer chaque $\mu \in$ ACT.

Avec les définitions modifiées de l'expansion et la restriction, les AC-règles de réécriture peuvent s'appliquer à des AC-graphes. Les contraintes d'éléments sont traitées similairement aux attributs ; elles sont ajoutées et supprimées avec l'élément qu'elles visent.

Les définitions des grammaires de graphes 4.1 et de leurs instances 4.2 sont quasi invariantes vis à vis de l'extension élaborée ici. La seule nouveauté réside dans le fait que les éléments de la grammaire (axiome, termes et productions) sont à présent attribués et contraintes, se conformant aux définitions AC-nœuds/règles.

Accès efficace aux valeurs d'attributs : évaluation à la demande ou mise à jour lors d'une modification.

Grâce aux mutateurs, ce formalisme autorise plusieurs moyens de considérer et d'évaluer attributs et contraintes. Ils peuvent premièrement être explicitement caractérisés par leurs expressions analytiques. Cette expression doit alors être évaluée soit à chaque fois que sa valeur doit être connue, soit après chaque transformation. Afin d'éviter de trop fréquents calculs, un attribut peut directement être associé à sa valeur. Cette dernière est alors mise à jour lorsque nécessaire par le biais de mutateurs.

La pertinence relative de ces deux options dépend des complexités et de la fréquence des mises à jour et des évaluations. L'adoption d'une solution pour la manipulation d'un attribut ou d'une contrainte n'est pas nécessairement définitive ; il est possible de passer de l'une à l'autre suivant l'évolution des paramètres précédemment évoqués.

6.3.4 Impact de l'extension sur la correction par construction

L'extension présentée dans ce chapitre n'est pas sans effet sur la méthodologie élaborée dans la section 4.3 et permettant de construire des transformations correctes par construction. Dans la suite, nous discutons ces effets et proposons des perspectives de résolution.

6.3.4.1 Inversion

Toute contrainte sur une règle pouvant être vue comme une pré-condition, elle doit devenir une post-condition de sa règle inverse. Par simple construction et sous réserve uniquement d'applicabilité d'une règle, le formalisme présent ne permet pas *a priori* de statuer sur la validation d'une post-condition quelconque.

Ensuite, il faut tout d'abord que tous mutateurs d'une règle soit eux-même inversibles. De plus, la validité de l'inversion se base sur un mécanisme de comptage régissant des modifications et basé sur l'apparition d'un élément dans une règle. Or, les mutateurs peuvent avoir des effets de bords en modifiant des attributs extérieurs à la règle. En l'état, les mutateurs invalident la propriété de l'inversion relative à sa conservation de la correction. Une solution immédiate serait d'étendre le mécanisme de comptage pour englober et restreindre la modification d'attributs par le biais de mutateurs. Ceci réduirait néanmoins grandement la souplesse de l'inversion et l'usage qui peut en être fait.

6.3.4.2 Composition

Dans la composition, les contraintes de la seconde règle ainsi que les mutateurs de la première posent problème.
Ces derniers peuvent impacter le graphe de manière à modifier l'applicabilité de la seconde règle. Cet impact est difficile à prévoir dans le cas d'un mutateur quelconque.
Deuxièmement, la seconde règle peut contenir un certain nombre de contraintes. Il ne suffit pas de les intégrer à la règle composée. Il s'agit de trouver un ensemble de contraintes devant être satisfaites avant application de la première règle pour que le résultat satisfasse les contraintes d'applicabilité de la seconde.

6.3.4.3 Spécialisation

Cette dernière opération est peu touchée par l'extension proposée. Rappelons qu'elle consiste à restreindre l'applicabilité d'une règle et/ou de réduire le panel de ses résultats possibles.

Une spécialisation doit donc simplement posséder les mêmes mutateurs que la règle originelle, ainsi que des contraintes équivalentes ou plus fortes.

6.4 Exploitation et illustration du nouveau formalisme : caractérisation, évaluation et gestion de DIET

Cette section illustre dans un premier temps le potentiel du formalisme élaboré en proposant une modélisation de DIET prenant en compte toutes les considérations introduites dans la section 6.2. L'adéquation de cette description vis à vis de l'évaluation et de la gestion est par la suite démontrée en s'appuyant sur des exemples concrets.

6.4.1 Caractérisation de DIET

6.4.1.1 Axiome

Vu la définition des règles de réécritures et grammaires de graphes, leurs instances sont des graphes héritant des attributs et contraintes du graphe axiomatique. Dans le cas de DIET et d'après les considérations de la section 6.2, les attributs et contraintes communs à toutes les configurations sont :

1. la cardinalité maximum d'entités qu'un LA peut gérer (la cardinalité minimum étant directement mise en œuvre par les règles de production).
2. la cardinalité maximum d'entités qu'un MA peut gérer (idem).
3. la valeur du seuil intervenant dans la condition d'équilibrage discutée dans la section 6.2.
4. le nombre maximum d'agents.
5. le nombre courant d'agents, noté courAgents.

Les contraintes communes sont relatives aux conditions de redondance et de dispersion physiques détaillées dans l'exemple développé au sein de la section 6.3.2.2.

Par conséquent, soit AX_{DIET} le nœud type ν_{AX}, attribué par ATT_{AX} = ((maxFilsLA, $\mathbb{N}$), (maxFilsMA, $\mathbb{N}$), (maxσ, $\mathbb{R}^+$), (maxAgents, $\mathbb{N}$), (courAgents, $\mathbb{N}$)) et contraint par $CONS_{AX}$ = (Loc(S,2), Red(S,3))).

La valeur de courAgents est initialisée à 0. Arbitrairement, maxFilsMA = maxFilsLA = 10 et maxAgents = 100.

Dans cette section, le graphe cible (i.e., celui sur lequel une règle de production est appliquée), est noté G = (V, E, ATT, CONS). L'héritage d'attributs et de contraintes garantit que si G est une instance du style architectural défini ici, $ATT_G = ATT_{AX}$ et $CONS_G \subseteq CONS_{AX}$, aux modifications induites par les mutateurs près. Ici, la valeur de courAgents est susceptible de varier.

6.4.1.2 Termes Terminaux

Encore une fois, ces termes représentent des archétypes de nœuds représentant des types de composant. Du fait de l'introduction de contraintes, ils caractérisent des archétypes d'AC-nœuds, définissant non seulement un pattern d'attributs mais également de contraintes. Ce dernier est partagé par tous les nœuds symbolisant un même type de composant ; les termes terminaux constituent ainsi l'outil principal pour la gestion de contraintes intégrées.

Les archétypes des termes terminaux de la grammaire sont représentés dans le tableau 6.4. Ils sont attribués conformément à la discussion de la section 6.2.

Le service de nommage n'est pas contraint et ses attributs sont limités à son identifiant, sa nature et la machine sur laquelle il est déployé. De même, les SEDs ne sont pas contraints.

En plus des attributs spécifiés précédemment, T_{MA}, l'archétype des MAs, est contraint, un MA ne devant pas gérer plus de maxFilsMA = A^2_{AX}.

Finalement, un LA et ses nœuds frères ne doivent pas être déployés sur la même machine, à moins de ne pas donner accès à des services similaires. Pour tout nœud ν représentant un LA ou un SED,

Type de composant	**Archétype**	**Attributs**	**Contraintes**
Omni	T_{Omni}	(id, ID) ("Omni", Nat) (m, Mach)	
MA	T_{MA}	(id, ID) ("MA", Nat) (nFils, $\mathbb{N}$) (m, Mach)	(nFils $\leqslant$ maxFilsMA) soit ($A^3_{MA} \leqslant A^2_{AX}$)
LA	T_{LA}	(id, ID) ("LA", Nat) (nFils, $\mathbb{N}$) (profondeur, $\mathbb{N}$) (m, Mach) (s, Serv)	$(CONS^i_{LA})_{i\in[1,\lvert(v_{LA})_f\rvert]}$ où $(v_{LA})_f$ est l'ensemble des frères du LA et $CONS^i_{LA}$ = $A^j_{v_{LA}} \neq A^j_{(v_{LA})^i_f} \vee A^k_{v_{LA}} \cap A^k_{(v_{LA})^i_f} = \emptyset$ Si $A^1_{(v_{LA})^i_f}$ = "LA", $j = 5$ et $k = 6$. Si $A^1_{(v_{LA})^i_f}$ = "SED" , $j = 4$ et $k = 3$.
SED	T_{SED}	(id, ID) ("SED", Nat) (s, Serv) (m, Mach)	

TABLE 6.4 – Termes terminaux de la grammaire

notons v_p son père, i.e. l'unique nœud $v_p \in V, (v_p, v) \in E$, et v_f l'ensemble des frères de v, i.e. $v_f = \{v^k_f \in V | (v_p, v^k_f) \in E \wedge v_f \neq v\}$. L'archétype des LAs, T_{LA} est un noeud v_{LA} contraint par :

$CONS_{LA} = (CONS^i_{LA})_{i\in[1,\lvert(v_{LA})_f\rvert]}$, où
$CONS^i_{LA} = A^5_{v_{LA}} \neq A^j_{(v_{LA})^i_f} \vee A^6_{v_{LA}} \cap A^k_{(v_{LA})^i_f} = \emptyset$ avec :
$j = 5$ et $k = 6$ si $A^2_{(v_{LA})^i_f}$ = "LA".
$j = 4$ et $k = 3$ si $A^2_{(v_{LA})^i_f}$ = "SED".

6.4.1.3 Productions

Les productions de la grammaire formalisent, comme précédemment, la construction de ses instances en définissant quand et comment une entité peut être déployée, ainsi que les conséquences d'un tel déploiement. Ces conséquences incluent à présent, grâce aux mutateurs, la modification d'attributs existants, ainsi qu'un ensemble explicite d'actions externes. Ces dernières peuvent être liées à la configuration de l'application modélisée, à des procédures de déclaration, à des mises à jour externes (comme dans le cas d'une représentation multi-modèles)...

La première règle (p_1) est similaire à celle introduite dans la section 4.2.2.2 ; elle ne diffère que par l'introduction de contraintes et de mutateurs (ainsi que certains attributs supplémentaires). Elle formalise l'initialisation remplaçant l'axiome par une configuration minimale. Le MA est configuré et s'enregistre au service de nommage. Finalement, le nombre d'agents courant est mis à jour en conséquence.

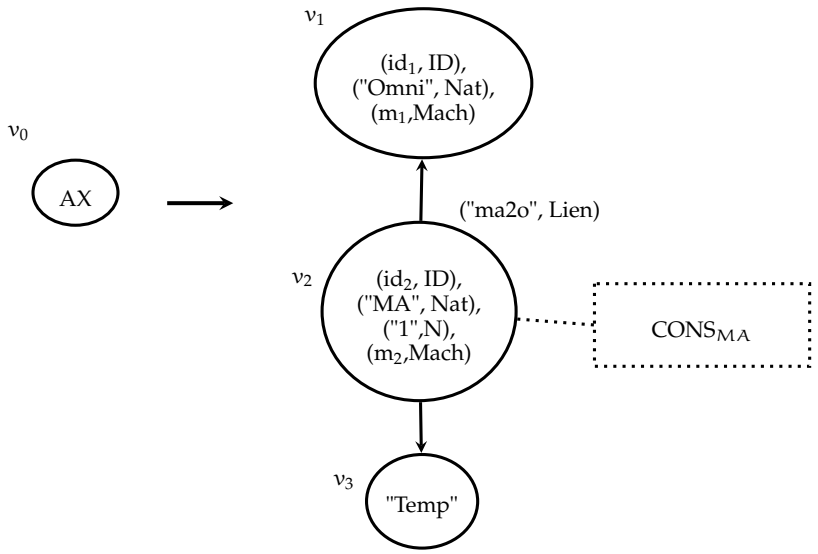


FIGURE 6.2 – Initialisation (p_1)

La production p_1 est la règle illustrée dans la figure 6.2 avec les mutateurs $\mu_{reg}(id_1, m_1, id_2, m_2)$, $\mu_{inc}(G, 5, 3)$, $\mu_{config,type}(v_2, \text{"MA"})$ et $\mu_{config,addr}(v_2, id_2, m_2)$, où

- $\mu_{reg}(id_o, m_o, id, m)$ est un mutateur externe décrivant la procédure d'enregistrement d'un composant au service de nommage. La connaissance d'une machine hôte et d'un identifiant permet de contacter le composant correspondant. Ainsi, la procédure prend contact avec l'OMNI (id_o, m_o) et lui transmet les informations nécessaires à l'enregistrement du nouveau composant (id, m).
- $\mu_{inc}(el, i, n)$ est un mutateur interne défini par l'algorithme 6.1. Il décrit l'incrémentation du i-ème attribut de el par n. Ici, il s'agit d'augmenter de 3 le nombre courant d'agents.
- $\mu_{config,type}(v, nat)$ et $\mu_{config,addr}(v, id, mach)$ sont des mutateurs externes de configuration. Ils mettent à jour le fichier de configuration du composant représenté par v en spécifiant son type et son adresse, respectivement.

Données : el, l'élément attribué à mettre à jour.
i, le numéro de l'attribut à modifier.
n, le nombre à ajouter.
début
  $A^i_{el} \leftarrow A^i_{el} + n$
fin

ALGORITHME 6.1 - $\mu_{inc}(el, i, n)$, n incrémentations du i-ème attribut de l'élément el

Les productions p_2 et p_3 sont illustrées dans la figure 6.3. Elles modélisent l'addition d'un nœud non-terminal, géré par un MA et un LA, respectivement. Ceci constitue la première étape pour l'ajout d'un composant, ce nœud temporaire étant plus tard instancié en un LA ou un SED. Pour déployer une nouvelle entité, trois conditions doivent être remplies ; il faut respecter la condition d'équilibrage (LA uniquement), le nombre maximum d'éléments gérables par un agent (les MAs possèdent déjà une contrainte reflétant cette considération) et le nombre maximum d'agents. L'application d'une de ces productions entraîne de plus l'incrémentation du nombre actuel d'agents et du nombre de fils du nœud père.

La production p_2 est contrainte par $A^4_G > A^5_G$ et possède les mutateurs $\mu_{inc}(v_2, 3, 1)$ (incrémentation du nombre d'agents gérés par v_2) et $\mu_{inc}(G, 5, 1)$ (incrémentation du nombre total d'agents).

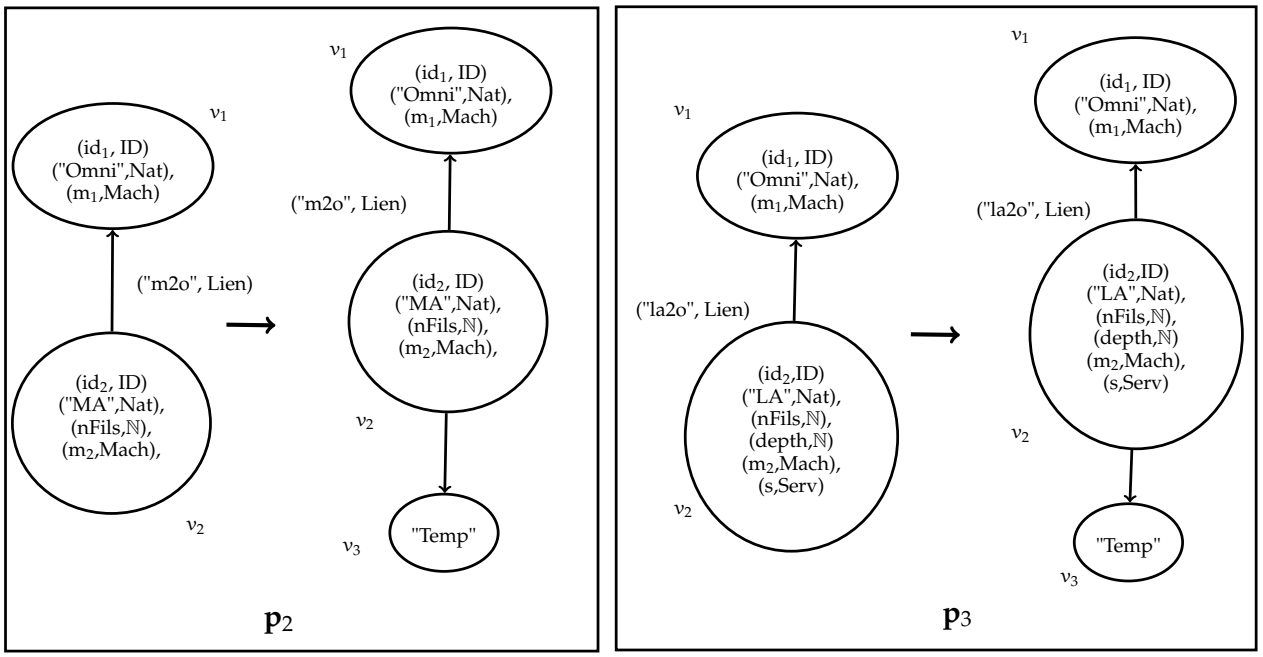


FIGURE 6.3 – Ajout d'un nœud temporaire sur un MA (p_2) ou un LA (p_3)

La production p_3 est contrainte par eq(v_2), $A^3_{v_2} < A^1_G$ et $A^4_G > A^5_G$. Ses mutateurs sont $\mu_{inc}(v_2, 3, 1)$ et $\mu_{inc}(G, 5, 1)$.

La condition eq(v) est égale à $\sigma((A^3_{la})_{la \in LA(A^4_v)\setminus\{v\}}, A^3_v{+}1) < A^3_G$, $\sigma(s)$ étant l'écart type de l'ensemble de données s.

Notons que les limitations sur le nombre d'entités gérées par un MA ou un LA ne sont pas assurées de la même façon. Une contrainte reflétant cette restriction est ajoutée au MA. Pour un LA, ce seuil est géré par un prédicat sémantique. Le dit prédicat réduit l'applicabilité de p_3 en imposant le non dépassement du seuil. Comme cette production est la seule permettant d'augmenter le nombre d'entités gérées par un LA, ce seuil ne peut donc jamais être dépassé.

L'instanciation d'un nœud terminal géré par un MA ou un LA en un SED est décrite par les productions p_4 et p_5, respectivement. Ces règles sont illustrées dans la figure 6.4. Après ajout, le SED est configuré et s'enregistre au service de nommage. S'il est géré par un LA, l'ensemble des services accessibles depuis ce dernier est mis à jour.

Les productions p_4 et p_5, décrites respectivement par $L_{p_4} \to R_{p_4}$ et $L_{p_5} \to R_{p_5}$, sont munies des mutateurs externes $\mu_{reg}(v_4)$, $\mu_{config,type}(v_4, \text{"SED"})$, $\mu_{config,addr}(v_4, id_3, m_3)$, $\mu_{config,pere}(v_4, id_2)$ et $\mu_{config,buffer}(v_4, serv)$. La règle p_5 possède en plus le mutateur $\mu_{majServ}(v_2,v_4,3)$, où

- $\mu_{config,buffer}(v, serv)$ est un mutateur externe configurant la taille du buffer du SED représenté par v en fonction des services qu'il fournit. La taille de ce buffer est déduite des services proposés.
- $\mu_{config,pere}(v, id)$ est un mutateur externe de configuration spécifiant que le père du nœud représenté par v_a a pour nom id.
- $\mu_{majServ}(v_a, v_b, i)$ est un mutateur interne correspondant à l'algorithme 6.2. Ce processus impacte l'ajout de nouveaux services à $A^i_{v_b}$, l'ensemble des services fournis par (ou accessibles depuis) v_b. L'ensemble des services accessibles depuis son père v_a est alors mis à jour si v_a est un LA (et non un MA). Si une modification est effectivement apportée, elle est récursivement propagée au père de v_a.

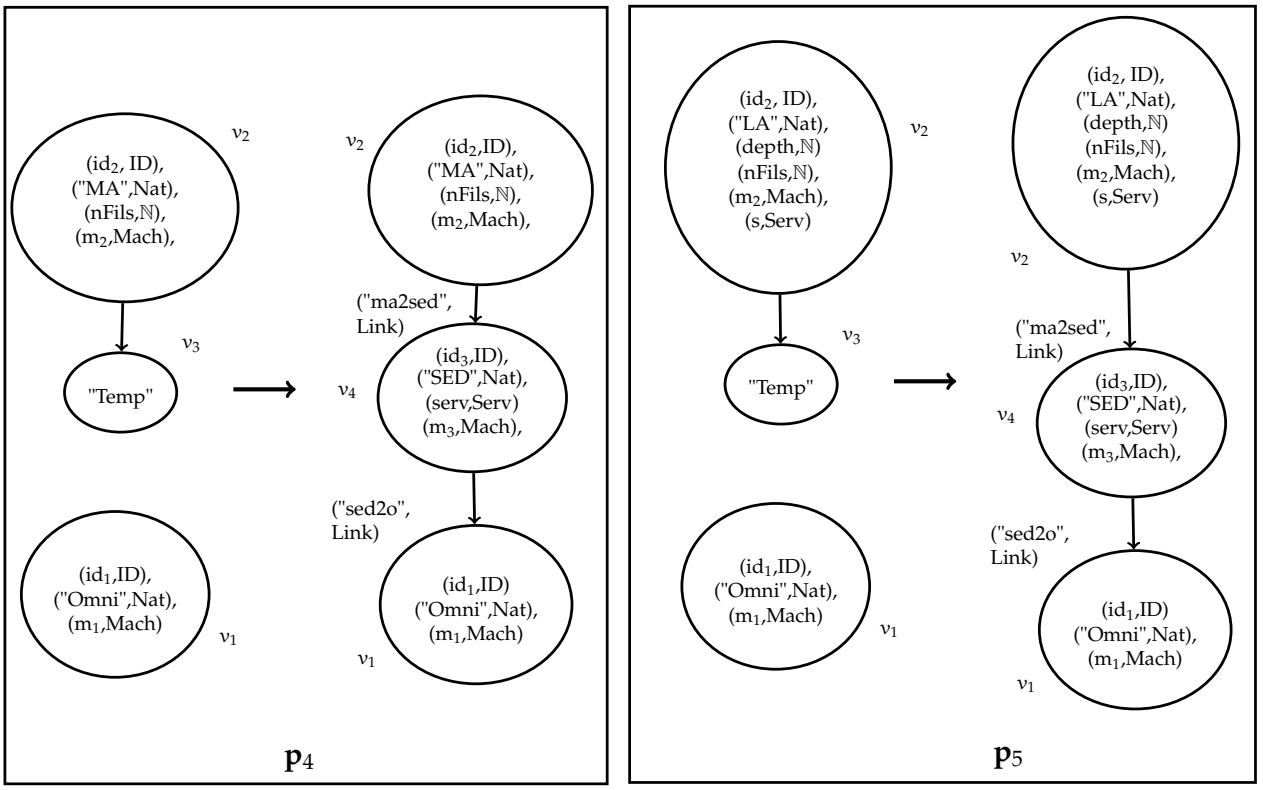


FIGURE 6.4 – Instanciation d'un nœud temporaire géré par un MA (p_4) ou un LA (p_5) en un SED

Données : v_a, le nœud attribué à mettre à jour.
v_b, le fils de v_a précédemment modifié.
i, le numéro de l'attribut modifié.
début
  si $A^2_{v_a}$ = *"LA"* **alors**
    oldServ $\leftarrow A^6_{v_a}$
    $A^6_{v_a} \leftarrow$ oldServ $\cup A^i_{v_b}$
    si $A^6_{v_a} \neq$ *oldServ* **alors**
      $v_c \leftarrow v \in V, (v_c, v_a) \in E$
      $\mu_{majServ}(v_c, v_a, 6)$
    fin
  fin
fin

ALGORITHME 6.2 - $\mu_{majServ}(v_a, v_b, i)$, Une extension de $A^i_{v_b}$, l'ensemble des services accessibles à partir de v_b, est impacté sur son père v_a.

Les deux dernières productions de la grammaire, p_6 et p_7, sont représentées dans la figure 6.5. Elles décrivent l'instanciation d'un terme non terminal en un LA géré par un MA et un LA, respectivement. Comme un LA doit gérer au moins une entité, une telle instanciation entraîne l'ajout d'un nœud temporaire. Ces transformations ne peuvent donc pas être conduites si un tel ajout n'est pas possible du fait du nombre limité d'agents total. À la fin de leur application, le LA est configuré et enregistré au service de nommage.

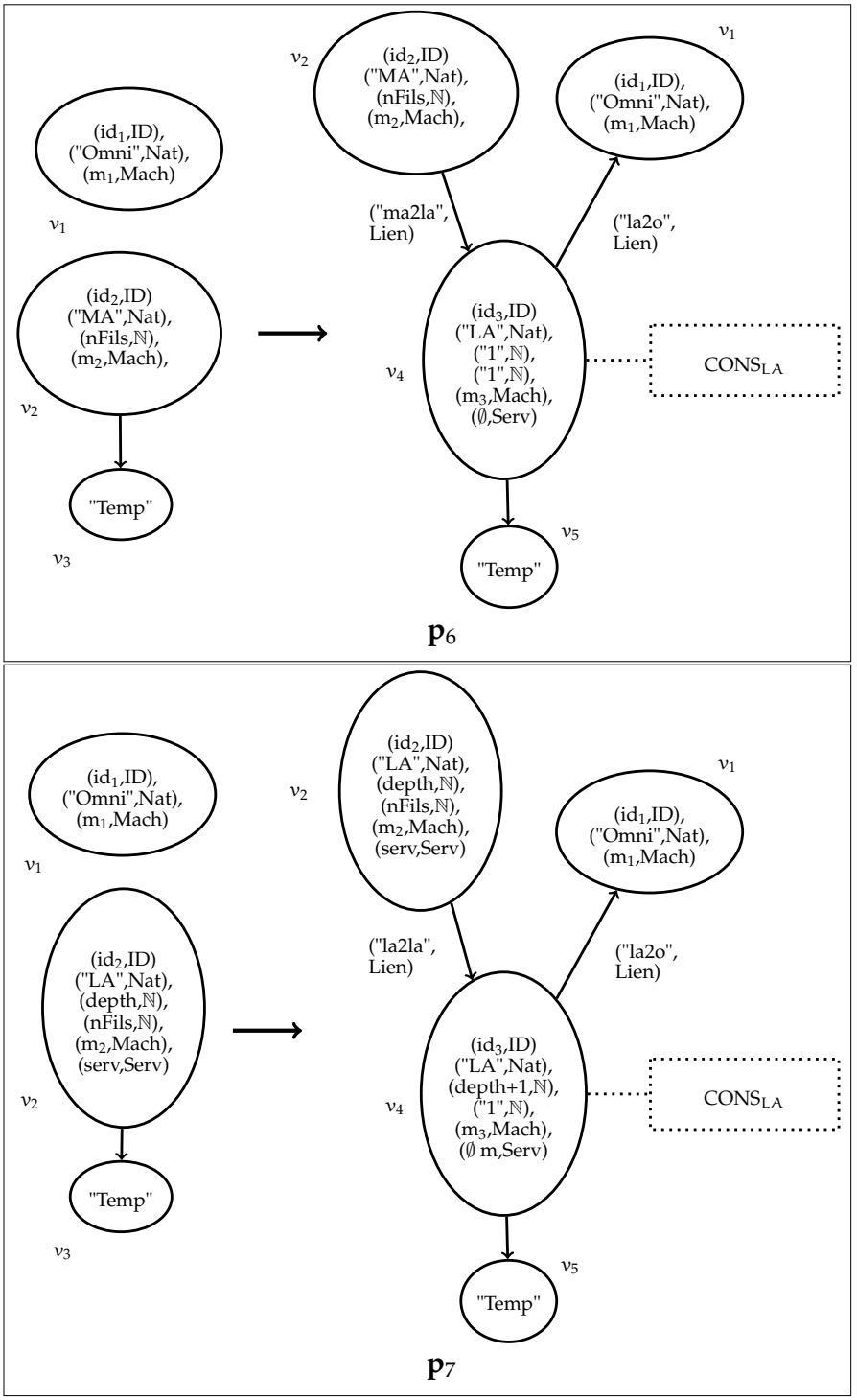


FIGURE 6.5 – Instanciation d'un nœud temporaire géré par un MA (p_6) ou un LA (p_7) en un LA

Notation	Expression formelle	Description
	Nœuds temporaires dans R_{p_1}, R_{p_6} et R_{p_7}	MA et LAs déployés avec $minFilsMA$ et $minFilsLA$ nœuds temporaires
$CONS_{MA}$	$A^3_{MA} < A^2_G$ $nFils < maxFilsMA$	Le MA est contraint afin de ne pas excéder son nombre max d'entités gérées
$CONS^2_{p_3}$	$A^4_{LA} < A^1_G$ $nFils < maxFilsLA$	Une nouvelle entité ne peut être gérée par un LA ayant atteint sa limite d'entités gérées
$CONS^1_{p_2}$ $CONS^3_{p_3}$ $CONS^1_{p_6}$ $CONS^1_{p_7}$	$A^4_G > A^5_G$	Gestion du nombre d'entités max : Déploiement d'un nouveau composant si le système n'a pas atteint sa limite d'entités gérables
$CONS^1_{p_3}$ $= eq(v_{LA})$	$\sigma((A^3_{la})_{la \in LA(A^2_v)\setminus\{v_{LA}\}}$, $A^3_{v_{LA}}+1) < A^3_G$	Respect de la condition d'équilibrage
$CONS^i_{LA}$	$A^j_{v_{LA}} \neq A^j_{(v_{LA})^i_f}$ $\vee A^k_{v_{LA}} \cap A^k_{(v_{LA})^i_f} = \emptyset$	Contrainte de dispersion logique
Red(S,3)	cf. 6.3.2.2	Contrainte de redondance. Tout service est fourni par au moins 3 SEDs.
Loc(S,2)	cf. 6.3.2.2	Contrainte de dispersion physique. Tout service est conductible sur au moins 2 machines .

TABLE 6.5 – Considérations informelles et leurs expressions formelles

Les règles p_6 et p_7 sont toutes deux contraintes par $A^4_G > A^5_G$ et possèdent les mutateurs $\mu_{reg}(v_4)$, $\mu_{inc}(G, 5, 1)$, $\mu_{config,type}(v_4, \text{"LA"})$, $\mu_{config,addr}(v_4, id_3, m_3)$ et $\mu_{config,pere}(v_4, id_2)$.

6.4.1.4 La grammaire contrainte et attribuée caractérisant DIET

Considérant les ensembles introduits dans cette section, GG_{DIET}, la grammaire caractérisant l'application DIET telle qu'introduite dans la section 6.2, est définie par GG_{DIET} =(AX_{DIET}, NT_{DIET}, T_{DIET}, P_{DIET}), où
NT_{DIET} = (v_{temp}, ATT_{temp} = ("temp",{"temp"}), $CONS_{temp} = \emptyset$),
T_{DIET} ={ T_{Omni}, T_{MA}, T_{LA}, T_{SED}}, et
$P_{DIET} = (p_i)_{i\in[1,7]}$.

Un bref récapitulatif de la façon dont chaque considération de la section 6.2 a été formellement prise en compte est proposé dans le tableau 6.5.

Garantir des propriétés théoriques de la grammaire : terminaison. Le formalisme proposé facilite également le raisonnement sur la grammaire elle-même et sur ses propriétés théoriques. Ce paragraphe propose d'analyser à titre d'exemple la terminaison de GG_{DIET}. Une grammaire générative possède la propriété de terminaison, et est dite finie, s'il ne peut exister de séquence infinie de ses règles de production.

Théoriquement, cette propriété assure que l'ensemble des instances de la grammaire est fini, sa construction et son exploration pouvant alors être conduites par un algorithme fini. De même, elle assure que la construction d'une instance se fait nécessairement en temps fini.

Dans la pratique, cette propriété est consistante avec le caractère fini des ressources disponibles.

Théorème 6.1. *GG_{DIET} est finie.*

Démonstration. Soit S une sequence (non vide) d'éléments de P_{DIET}. Prouvons que $\exists\, N \in \mathbb{N}$, $|S| \leqslant N$.

Pour tout $p \in P_{DIET}$, soit Occ(p) le nombre d'occurrences de p dans S. Ainsi,

$$|S| = \Sigma_{p \in P_{DIET}} Occ(p). \tag{6.1}$$

Considérons le système de jetons suivant :
- Jeton A : G_4 - G_5, le nombre d'agents pouvant encore être déployés.
- Jeton B : le nombre de nœuds non-terminaux du graphe.

Appliquer p_2, p_3, p_6 ou p_7 diminue le nombre de jetons A de 1, trois jetons A étant consommés par p_1. Par conséquent,

$$\begin{aligned} &3 * Occ(p_1) + Occ(p_2) + Occ(p_3) + \\ &Occ(p_6) + Occ(p_7) \leqslant maxAgents. \end{aligned} \tag{6.2}$$

L'application de p_4 ou p_5 consomme 1 jeton B, tandis que l'application de p_1, p_2 ou p_3 produit un jeton B. De fait,

$$Occ(p_4) + Occ(p_5) \leqslant Occ(p_1) + Occ(p_2) + Occ(p_3). \tag{6.3}$$

Comme p_1 consomme l'axiome, il est évident que

$$Occ(p_1) = 1. \tag{6.4}$$

L'équation (6.2) devient alors :

$$Occ(p_2) + Occ(p_3) + Occ(p_6) + Occ(p_7) \leqslant maxAgents - 3. \tag{6.5}$$

Par définition, $\forall\, p \in P_{DIET}$, $Occ(p) \geqslant 0$. Ainsi, les équations (6.3), (6.4) et (6.5) donnent

$$Occ(p_4) + Occ(p_5) \leqslant maxAgents - 2. \tag{6.6}$$

Selon l'équation (6.1), |S| = $Occ(p_1)$ + ($Occ(p_2)$ + $Occ(p_3)$ + $Occ(p_6)$ + $Occ(p_7)$) + ($Occ(p_4)$ + $Occ(p_5)$). Du fait des équations (6.4), (6.5) et (6.6), ceci devient $|S| \leqslant 1$ + (maxAgents - 3) + (maxAgents - 2).

Finalement, $|S| \leqslant$ 2*maxAgents-4. Avec maxAgents = 100, on obtient $|S| \leqslant 196$.

□

6.4.2 Évaluation de l'adéquation des configurations

Cette représentation enrichie du système facilite sa gestion et en particulier son évaluation nécessaire aux processus d'(auto-)optimisation.

Afin de permettre l'évaluation des différentes configurations, nous proposons d'assigner un poids (potentiellement infini) aux contraintes. Ce poids permet de définir une fonction de coût reflétant l'inadéquation d'une configuration. La violation d'une contrainte par une configuration implique l'augmentation de son coût reflétant du non respect ou de l'affaiblissement d'un critère d'évaluation. Plus le poids associé à une contrainte est important, plus elle est critique et plus élevé est le surcoût lié à sa violation. La violation d'une contrainte de poids infini implique une configuration de poids infini assimilable à un état incorrect. Une telle contrainte est alors analytique, une contrainte de poids finie étant qualifiée d'heuristique.

Le coût total d'une configuration est ici calculé comme étant la somme des coûts liés à la consommation énergétique et aux violations de contraintes.

Notations. Soit ξ la fonction d'évaluation logique ; $\forall$ cons $\in$ CONS, $\forall$ c $\in$ cons, ξ(c) = 1 si c est égale à "vrai" et 0 sinon.

Consommation énergétique. Dans la section 6.2, il a été supposé que la consommation énergétique dépend uniquement des machines utilisées et du nombre de composants logiciels déployés. Conformément aux travaux de Borgetto et al. [134], nous supposons cette relation linéaire et notons les coefficients correspondants λ_{mach} et λ_{entite}.

Notons que le nombre de composants logiciels déployés est un attribut du graphe. Pour faciliter l'évaluation, le nombre de machines utilisée peut également être représenté par un attribut dont la valeur est mise à jour quand nécessaire, i.e. lors de l'application des règles productions p_1, p_4, p_5, p_6 ou p_7. Comme ce nombre est facilement obtenable, supposons le connu et noté n_{mach}.

L'énergie consommée par une configuration est alors :

$$\lambda_{entite} \cdot A^5_G + \lambda_{mach} \cdot n_{mach} \tag{6.7}$$

Violation de contraintes. Il est clair que la contrainte reflétant la limite du nombre d'entités gérées par un MA doit absolument être respectée et par conséquent avoir un poids infini.

Les contraintes reflétant la robustesse du système sont, au contraire, heuristiques. Des poids finis choisis arbitrairement leurs sont assignés.
Le coût de la violation d'une contrainte de dispersion logique spécifiant qu' "un LA et un de ses nœuds frères ne doivent pas être déployés sur la même machine, à moins de ne pas donner accès à des services similaires" est égal à la profondeur du dit LA.
Les contraintes de redondance et de dispersion physique ont pour poids λ_R et λ_L, respectivement.

Le coût lié à la violation de contraintes est ainsi :

$$\begin{aligned} &\lambda_L \xi(CONS^1_G) + \lambda_R \xi(CONS^2_G) + \\ &\infty \cdot \Sigma_{ma \in V, A^2_{ma} = \text{"MA"}}\ \xi(CONS^1_{ma}) + \\ &\Sigma_{la \in V, A^2_{la} = \text{"LA"}}\ (A^3_{la} \Sigma_{i \in [1, |CONS_{la}|}\ \xi(CONS^i_{la})) \end{aligned} \tag{6.8}$$

Une partie de la première configuration illustrative de la figure 6.1 est arbitrairement instanciée afin d'être évaluable. Le résultat de cette instanciation est présenté dans la figure 6.6. S, l'ensemble des services pouvant être assuré, est {"s_1","s_2","s_3"}.

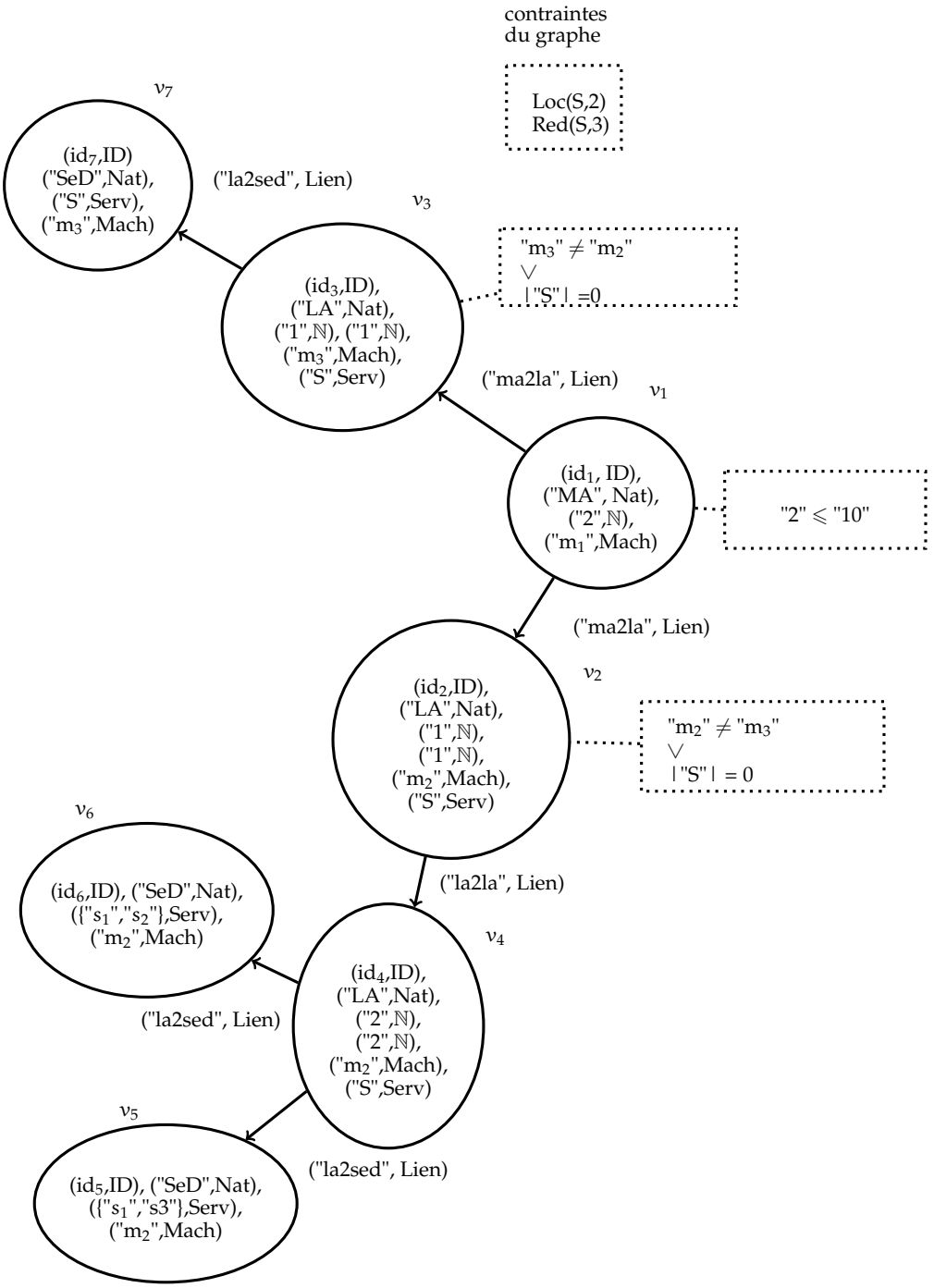


FIGURE 6.6 – AC-graphe instancié représentant une configuration de DIET

Cette configuration ne remplit pas la contrainte de redondance, comme seuls deux SEDs proposent les services "s_2" et "s_3". Son coût total est égal au coût de consommation énergétique, $3\lambda_{mach} + 7\lambda_{entite}$, plus celui lié à la contrainte violée, λ_R.

Cette manière naturelle et immédiate de considérer les exigences heuristiques et analytiques de l'application dérive du formalisme proposé ici. En fait, ce dernier a été explicitement conçu pour permettre l'intégration de contraintes et pour faciliter leur manipulation ainsi que celle des ses attributs. Ainsi, une fois un système correctement défini, son adéquation peut facilement être évaluée dynamiquement.

6.4.3 Contraintes non-fonctionnelles et gestion du système

La manipulation aisée des attributs et contraintes du système renforce la pertinence du modèle pour la gestion logicielle. En outre, les prédicats sémantiques et restrictions sur l'applicabilité des transformations permettent de construire facilement des règles spécialisées et ainsi de traiter dynamiquement des objectifs particuliers.

Remarque 6.3. Rappelons que l'utilisation d'une règle spécialisée garantit la conservation du style architectural. Néanmoins, les propriétés du système de réécriture lui-même (sa terminaison, sa confluence...), ne sont pas nécessairement invariantes vis à vis de l'ajout d'une telle règle.

Considérons la configuration de DIET représentée dans la figure 6.6 et évaluée précédemment. Déployer un SED peut permettre de satisfaire la contrainte de redondance et ainsi augmenter la qualité de la configuration en réduisant son coût.

Il s'agit tout d'abord d'appliquer p_2 et, ce faisant, de choisir le composant gérant le nouveau SED. Pour trouver une solution optimale, il est possible d'explorer toutes les possibilités (i.e., ν_2, ν_3 et ν_4), de trouver une solution idéale pour chaque cas et de comparer les coûts de chacune de ces solutions. En choisissant arbitrairement d'appliquer p_2, et donc déployer le nouveau SED, sur ν_2, une solution optimale peut être trouvée comme suit.

Ensuite, le nœud temporaire sera instancié en un SED à l'aide de la production p_5. Comme la motivation de cette reconfiguration est la satisfaction de la contrainte de redondance, une spécialisation de p_5 doit être envisagée.

Premièrement, le SED déployé doit fournir les services "s_2" et "s_3". En notant ν_p^4 le nœud représentant le SED et noté ν_4 lors de la définition de p_5 (voir figure 6.4), $\{\text{"}s_2\text{"}, \text{"}s_3\text{"}\} \subseteq A^3_{\nu_p^4}$.
Deuxièmement, la contrainte $(\text{"}m_2\text{"} \neq A^3_{\nu_p^4}) \vee \text{"S"} = \{\text{"}s_1\text{"}, \text{"}s_2\text{"}, \text{"}s_3\text{"}\} \cap A^2_{\nu_p^4} = \emptyset)$ apparaîtra sur le nœud ν_4. Afin de la satisfaire, la transformation doit vérifier $A^3_{\nu_p^4} \neq \text{"}m_2\text{"}$.
Finalement, eût égard à la consommation énergétique et pour ne pas utiliser une nouvelle machine, il est imposé que $A^3_{\nu_p^4} \in \{\text{"}m_1\text{"}, \text{"}m_2\text{"}, m_3\text{"}\}$.

D'après les méthodes d'évaluation adoptées pour ce style, le graphe présenté dans la figure 6.7 est un des résultats optimums de la reconfiguration envisagée. Toutes les contraintes sont satisfaites et le coût de la configuration est à présent limité à sa composante liée à la consommation énergétique, $3\lambda_{mach} + 8\lambda_{entite}$. Ainsi, cette transformation est pertinente si et seulement si $\lambda_{entite} < \lambda_R$.

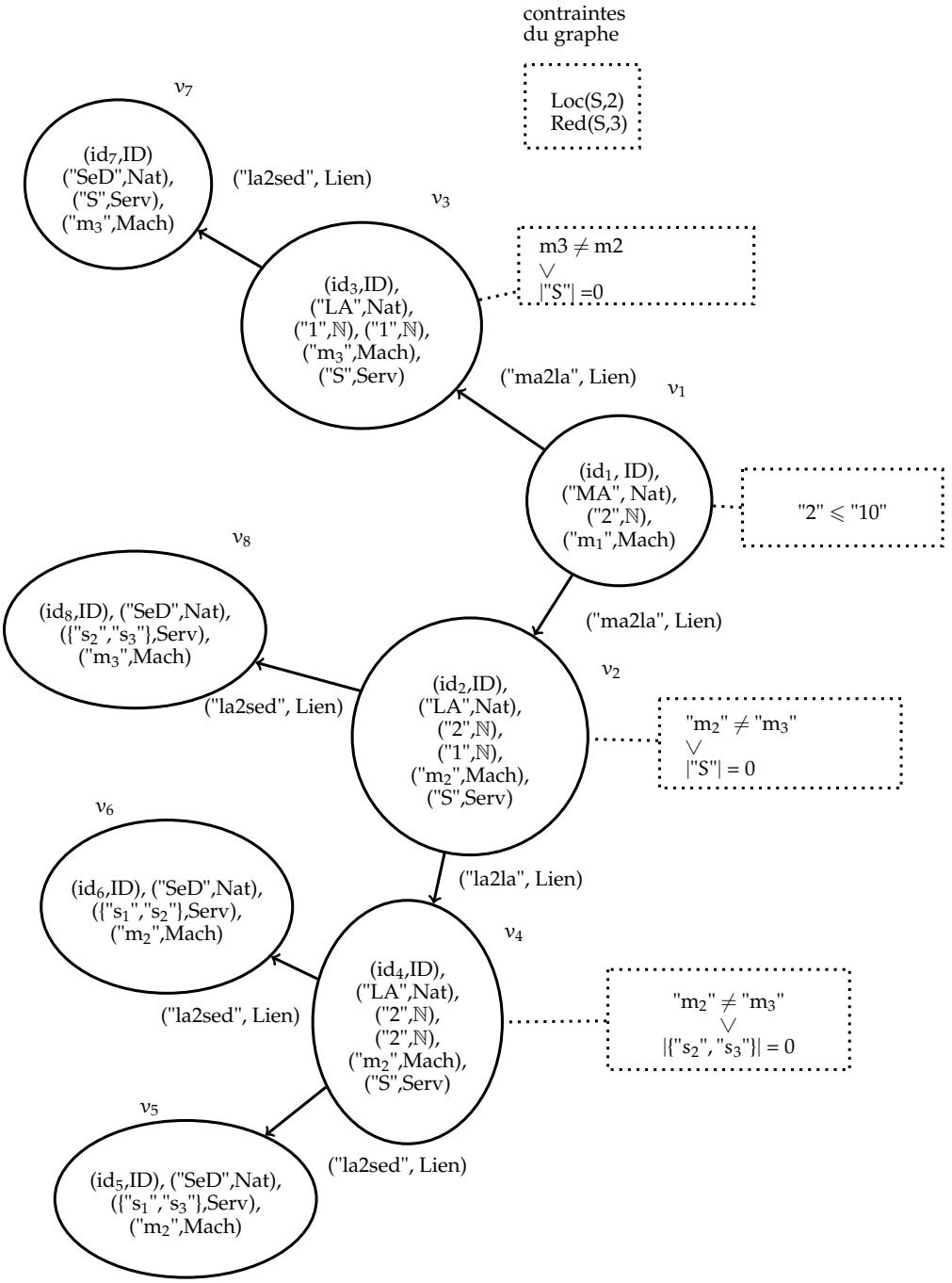


FIGURE 6.7 – Une configuration de DIET satisfaisant toutes les contraintes définies par le style

6.5 Expérimentations : évaluation et comparaison

6.5.1 Contexte expérimental

6.5.1.1 Scénario de reconfiguration

Les expériences présentées ici sont construites autour de l'ajout d'un SED sur un LA de profondeur maximale. Cette reconfiguration implique diverses mises à jour d'attributs :

Premièrement, les attributs représentant le nombre d'agents du système et le nombre d'entités gérées par le LA doivent être incrémentés.
Deuxièmement, l'introduction d'un SED peut étendre l'ensemble des services atteignables depuis un ou plusieurs de ses ancêtres. Dans le scénario exécuté, le SED propose un service inédit ; l'ensemble en question doit donc effectivement être modifié pour chacun de ses ancêtres.

Suivant la méthode introduite dans ce chapitre, cette transformation est une composition des règles p_3 et p_5 présentées dans la section 6.4.1.3. Les mises à jours d'attributs sont effectuées par des mutateurs de ces productions, μ_{inc} et $\mu_{majServ}$.

6.5.1.2 Configurations manipulées

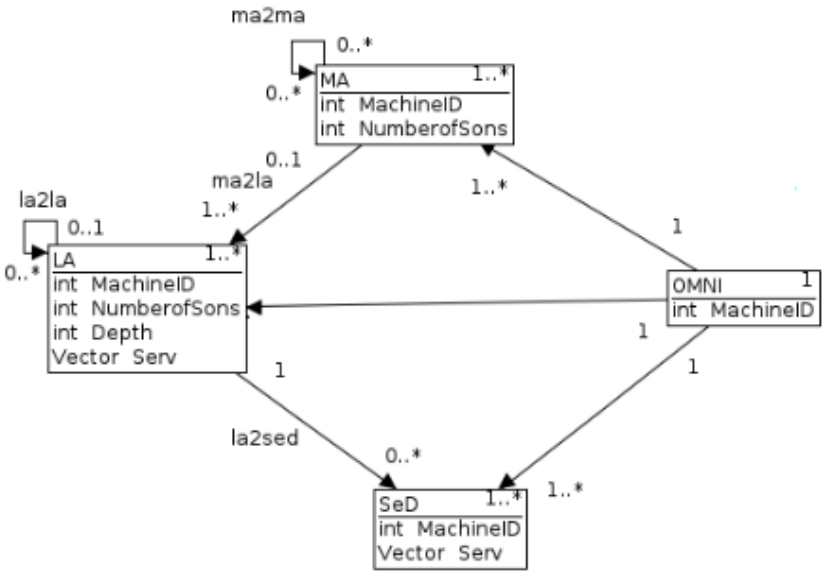


FIGURE 6.8 – Style expérimental de DIET

Dans cette partie expérimentale, une architecture non-simplifiée est considérée. Le graphe type correspondant est représenté dans la figure 6.8. Une configuration peut à présent comporter plusieurs MAs connectés et par conséquent présenter des cycles. Pour correspondre à des cas réalistes et non-simplistes, les graphes manipulés possèdent les caractéristiques suivantes :

- Chacun comporte plusieurs MAs et au moins un cycle.
- Les arbres composés des sous-graphes induits par chacun des MAs avec les LAs et les SEDs qu'il gère n'ont pas la même hauteur. À l'intérieur de ces arbres, les SEDs n'ont pas nécessairement la même profondeur.
- Il n'existe pas de n tel qu'un de ces arbres est n-aire.
- Chaque LA non-intermédiaire gère au moins 100 SEDs.

Les configurations étudiées varient par leurs tailles et leurs hauteurs. La taille d'une configuration est égale au nombre de composants la constituant. Sa hauteur est égale au plus grand nombre de LAs entre un MA et un SED. Le plus petit graphe transformé est illustré dans la figure 6.9, dans laquelle les cercles bleus et rouges représentent des LAs et des SEDs, respectivement. L'OMNI est représenté sous la forme d'un rectangle rouge arrondi.

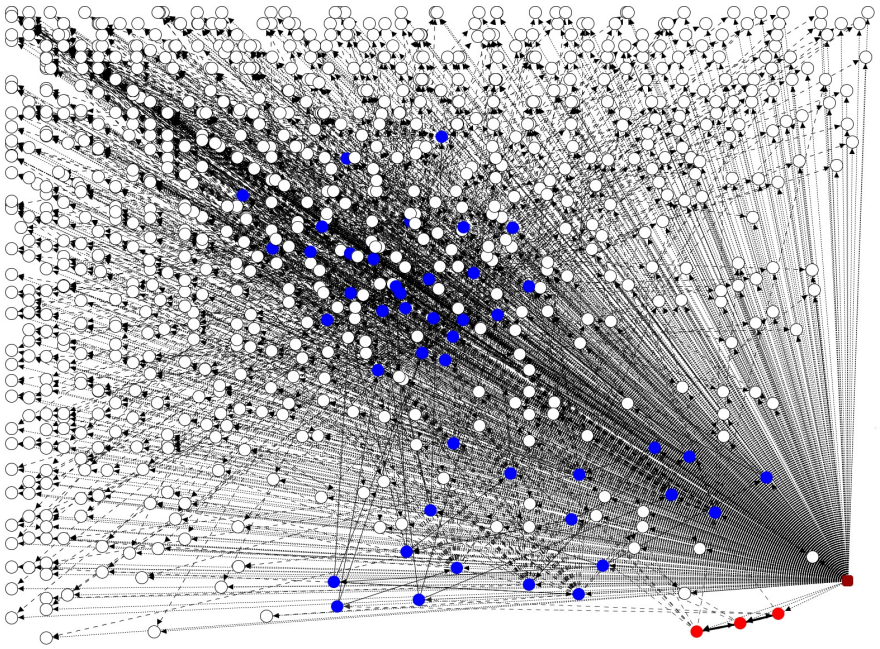


FIGURE 6.9 – Plus petit graphe expérimental : une configuration de DIET de taille 1000 et hauteur 5

La taille et la hauteur d'un graphe influence significativement l'exécution du scénario expérimental. Pour les méthodes natives, la hauteur du graphe coïncide avec le nombre d'applications de règles nécessaires à la conduite de la transformation.
La complexité et la durée d'exécution d'une application dépendent principalement de la taille du graphe cible. La topologie du graphe et, donc, son nombre de MAs, constitue un autre facteur, bien que moins impactant. Son influence n'étant pas pertinente dans la présente étude, les configurations expérimentales possèdent un nombre fixe de MAs (3).

6.5.1.3 Outils et méthodes testés

Les expériences reposent sur deux outils différents : AGG [27] et GMTE [6].
AGG [27] est l'un des logiciels de transformation de graphe les plus utilisés actuellement [99] et dont l'efficacité n'est plus à démontrer [100]. C'est donc tout naturellement qu'il a été utilisé dans ces expériences.
Les évaluations expérimentales présentées par son créateur [6] suggèrent que GMTE est significativement plus efficace qu'AGG. Ce gain a motivé l'inclusion de ce second moteur dans la présente étude, afin d'investiguer les bénéfices apportés par la méthode élaborée précédemment en terme d'efficacité et d'extensibilité.

Ces deux moteurs permettent à n'importe quelle règle de modifier tout attribut apparaissant dans sa partie droite. Pour chaque outil, le scénario de transformation est conduit suivant trois méthodes différentes :

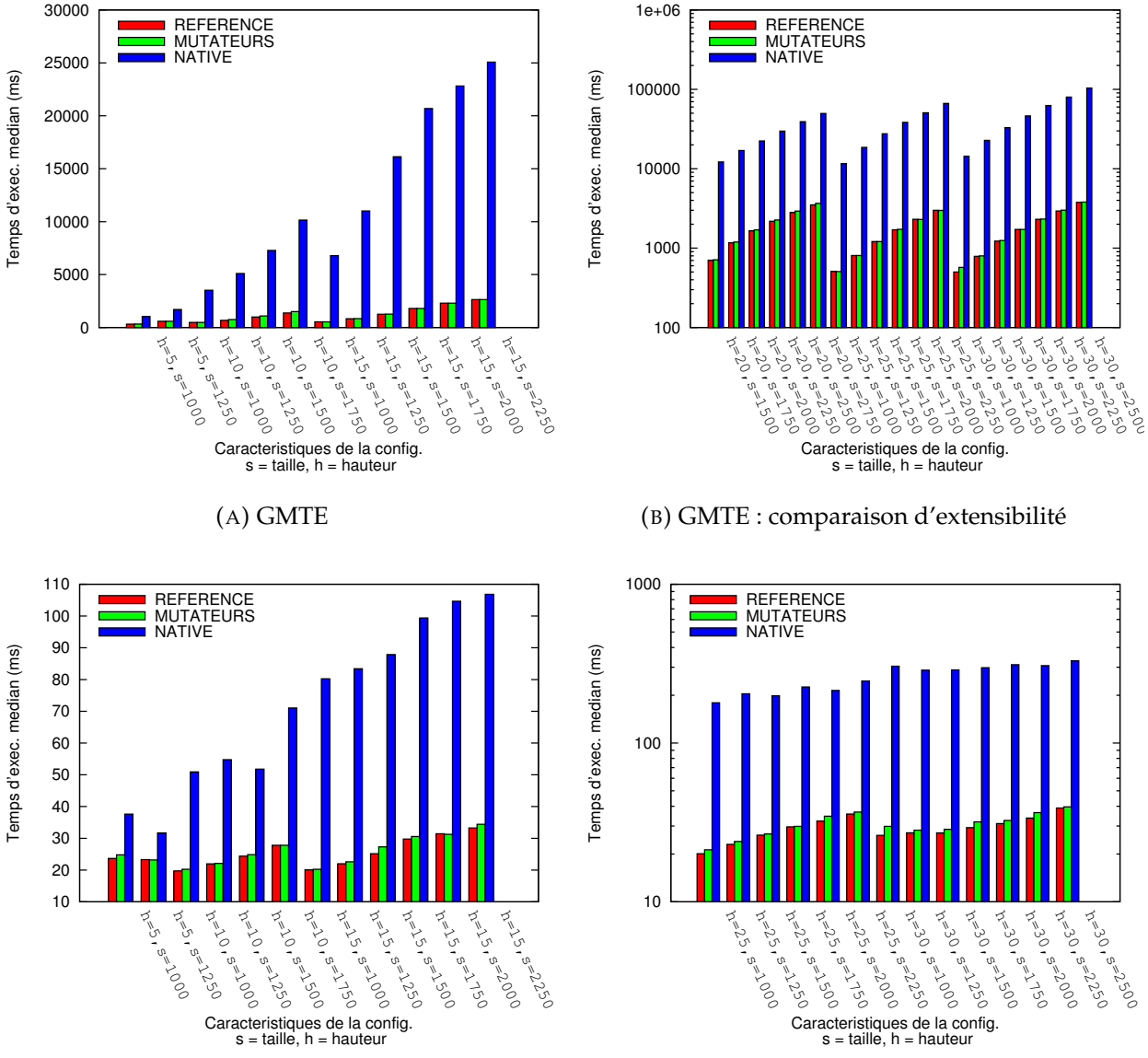
— "REFERENCE", un référentiel n'incluant pas de modification d'attribut. Il permet d'estimer le surcoût temporel induit par cette modification.
— "NATIVE", la méthode native du moteur, soit une séquence d'application de règles. Premièrement, le SED est ajouté sur un LA dont l'ensemble de services est mis à jour. Ensuite, une

règle est séquentiellement appliquée afin d'impacter cette modification sur chacun des ancêtres du LA. Seuls les attributs apparaissant dans la règle pouvant être modifiés, chaque application touche un unique LA.

— "MUTATEURS", La méthode proposée ici. Pour ce faire, deux sur-couches ont été implémentées sur AGG et GMTE. Celles-ci mènent à bien l'exécution des règles p_3 et p_5 incluant leurs mutateurs.

Considérer deux outils différents démontre l'indépendance de la méthode proposée et des conclusions tirées des expérimentations vis à vis du moteur utilisé.

6.5.2 Résultats expérimentaux



(A) GMTE

(B) GMTE : comparaison d'extensibilité

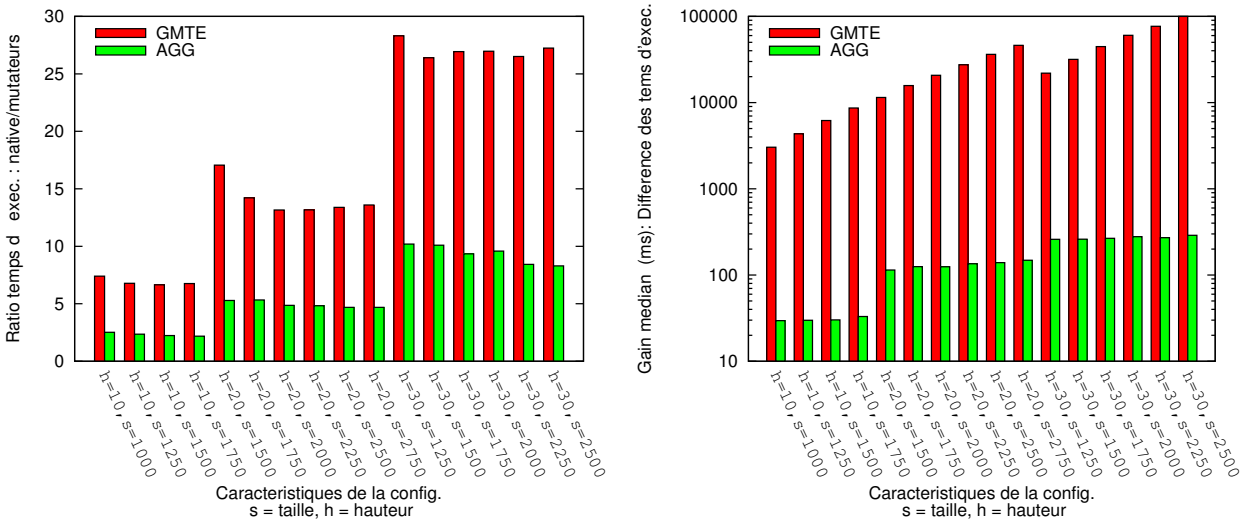
(C) AGG

(D) AGG : comparaison d'extensibilité

FIGURE 6.10 – Temps d'exécution du scénario de reconfiguration

Les figures 6.10a et 6.10c illustrent, pour GMTE et AGG respectivement, les temps d'exécution du scénario de transformation selon les trois méthodes précédemment décrites. Les figures 6.10b et 6.10d montrent l'évolution de ces temps pour de plus grand graphes afin d'apprécier l'extensibilité des méthodes étudiées. Les temps d'exécution rapportés sont les temps médians sur 100 exécutions. Les expériences ont été conduites sur un ordinateur possédant un processeur quadricœurs (4M Cache, 2.66 GHz, 1333 MHz FSB) et 8 Go de RAM. Chaque configuration est caractérisée par sa taille et sa hauteur.

Premièrement, les résultats expérimentaux montrent que le sur-coût temporel induit par les modifications d'attributs à l'aide de mutateurs (i.e., mutateurs − reference) est petit devant le temps de transformation total si elles sont effectuées à l'aide de mutateurs. Par exemple, pour une configuration de taille 1000 et hauteur 5, ces modifications constituent 7 des 325 ms et 1,2 des 24,8 ms nécessaires à la conduite du scénario à l'aide de GMTE et AGG, respectivement. Ce sur-coût augmente linéairement en hauteur et reste globalement invariant vis à vis de la taille du graphe transformé. Pour une configuration de taille 2250 et hauteur 30, il équivaut à 58 ms sur 3010 avec GMTE et 2,7 ms sur 36,6 avec AGG.



(A) Ratio des temps d'exécution (B) Gain net

FIGURE 6.11 – Méthodes natives vs. mutateurs

Deuxièmement, les mutateurs améliorent significativement l'efficacité et l'extensibilité des modification du système par rapport aux méthodologies classiques. La figure 6.11a montre l'évolution du ratio des temps d'exécution des méthodes natives sur les mutateurs. Eût égard à une configuration de taille 1750 et hauteur 20, par exemple, ce ratio est d'environ 14 pour GMTE (de 16925 à 1192 ms) et 5,3 pour AGG (de 154 à 28,9 ms). Il augmente avec la hauteur et décroît logarithmiquement en fonction de la taille du graphe considéré. Par exemple, le ratio atteint 27 pour GMTE (de 103796 à 3807 ms) et 8,3 pour AGG (de 330 à 39,6 ms) sur une configuration de hauteur 30 et taille 2500. Contrairement au ratio étudié, le gain net, i.e. la différence des temps d'exécution, est strictement croissante à la fois en taille et en hauteur, comme illustré dans la figure 6.11b.

Ces résultats peuvent être expliqués par le fait que la méthode proposée ne nécessite qu'une seule application de règle suivie de l'exécution d'un mutateur dont la complexité temporelle est linéaire vis à vis de la hauteur du graphe. Les méthodes natives, quant à elles, requièrent h applications de règles pour mettre à jour les attributs d'un graphe de hauteur h. Une application de règle elle-même

a une complexité temporelle polynomiale en fonction de la taille du graphe pour AGG, tandis qu'elle est exponentielle pour GMTE. Le gain net augmente avec le temps nécessaire à l'application d'une règle.

Les résultats ne sont néanmoins pas indépendants du scénario étudié. On assiste ici à une propagation de modifications constituant un exemple typique de l'effet domino évoqué dans la section 6.2.2.2. Toutefois, ce scénario comprend également le pire cas de comparaison, l'incrémentation du nombre de composants gérés. Comme cette modification n'impacte que les attributs apparaissant dans la première règle, mutateurs et méthodes natives sont alors équivalentes.

6.6 Conclusion du chapitre

Ce chapitre décrit une **extension des systèmes de réécriture de graphe permettant de représenter, mettre à jour, évaluer et paramétrer les caractéristiques du système modélisé aisément et efficacement**. Ces prétentions sont soutenues par des exemples illustratifs concrets ainsi que par une étude expérimentale extensive. Le contenu de ce chapitre a fait l'objet d'une publication [25].

En référence à une application réelle, nous soulignons dans un premier temps les **limitations des approches de réécritures existantes**. Celles-ci portent en particulier sur les interdépendances d'attributs, la modification d'attributs existants ainsi que l'intégration de contraintes.

En conséquence, nous proposons une extension des systèmes de réécritures et grammaires de graphes levant ces restrictions. Cette **extension** est **construite autour de la spécification de relations admissibles ainsi que de l'intégration de contraintes et de mutateurs**.
Les interdépendances entre attributs sont ainsi exprimées grâce à une signature algébrique qualifiant toutes leurs relations admissibles plutôt que de les restreindre à des opérateurs prédéfinis.
Les mutateurs sont introduits au sein des règles de réécriture et constituent une méthode légère et flexibles pour la modification d'attributs et de contraintes existants. Ils permettent également d'expliciter des opérations (de configuration) ayant trait au système lui-même plutôt qu'à sa représentation.
Finalement, afin de faciliter les opérations de management et les décisions de reconfiguration, la qualité d'une reconfiguration au regard de critères fonctionnels et non fonctionnels est reflétée par des contraintes. Ces dernières sont introduites comme des attributs particuliers, bénéficiant ainsi de tous les mécanismes permettant de les manipuler. Afin de prendre en compte les attributs dont la valeur est inconnue, elles sont définies comme des éléments d'un système logique ternaire .

L'application du formalisme résultant à la spécification de l'**application illustrative démontre son adéquation pour la gestion de systèmes sujets à des contraintes et critères de qualité**, qu'ils soient fonctionnels ou non. Des exemples et scénarios concrets sont ainsi fournis pour démontrer les possibilités de la proposition en terme d'évaluation et de gestion.

L'approche proposée est alors expérimentalement évaluée et comparée à diverses méthodes existantes à l'aide de deux outils différents, AGG et GMTE. Reconfigurer un graphe de taille 2500 avec des mutateurs plutôt qu'en usant de méthodes classiques est, **d'après étude expérimentale, jusqu'à 27 fois plus rapide sur GMTE et 8,3 fois sur AGG**. Ce gain significatif permet d'efficacement apprécier les caractéristiques du système en combinant évaluation à la demande et mise à jour lors d'une modification. À son tour, cette évaluation améliorée supporte une identification et une appréciation rapide de la qualité d'une configuration, des objectifs pouvant être améliorés ainsi que des composants entraînant une violation de contrainte.

Chapitre 7

Bilan des contributions et perspectives

Dans cette thèse, nous nous sommes intéressés aux architectures logicielles dynamiques et à leur représentation à l'aide de graphes, de règles de réécriture et de grammaires. Cette modélisation a pour vocation d'être intégrée au sein d'une base de connaissances dans le cadre de la gestion autonome de systèmes et applications adaptables.

Par conséquent, elle doit répondre à tous les différents aspects et exigences de l'informatique autonome, afin de permettre en particulier la mise en œuvre des quatre propriétés majeures de l'autogestion : auto-guérison, auto-configuration, auto-optimisation et auto-protection.

Deux aspects centraux ont particulièrement capté notre attention.
Le premier est la conservation de la correction d'un système lors de ses évolutions. Cette considération est sous-jacente à toute adaptation, quel qu'en soit le but. Sa vérification en phase d'exécution doit être la plus rapide possible, et, si possible, apporter des garanties *a priori* afin d'éviter des procédures de retour en arrière.
Le second réside dans la représentation des propriétés et contraintes non fonctionnelles du système ainsi que la prise en compte de leur propre dynamique. Cet aspect conditionne directement l'expressivité du modèle vis à vis des éléments permettant la prise de décision au sein des politiques autonomes.

7.1 Récapitulatif et bilan des contributions

7.1.1 Contributions théoriques

Dans un premier temps, **nous avons conçu les bases de la représentation utilisée** tout au long de ce manuscrit. Ce méta-modèle, basé sur les graphes et leur réécriture, a été élaboré dans la continuité des tentatives de traductions d'approches catégorielles d'un point de vue ensembliste et opérationnel plus classique et plus proche de l'implémentation en formalisant et étendant une approche existante [6].

Pour répondre à la *problématique de validation* et, plus précisément, à celle de l'**auto-protection interne**, nous avons élaboré une **méthodologie correcte par construction pour la préservation de la correction d'un système adaptable.** Ce type de méthodologie vise à construire des systèmes corrects pour des spécifications données.

Dans notre approche, ces spécifications sont caractérisées par un style architectural fournissant ainsi la base axiomatique de la notion de correction. Une configuration est alors considérée correcte vis à vis d'un style si elle s'y conforme (i.e, si elle en est une instance). Une transformation est à son tour dite correcte vis à vis d'un style si elle le préserve, c'est à dire que son application à une configuration correcte quelconque produit *nécessairement* une (autre) configuration correcte.

L'adoption de grammaires de graphe pour la spécification de styles architecturaux permet, grâce à leur caractère génératif, de raisonner sur un style quelconque (i.e., dont les propriétés et contraintes ne sont pas connues). En effet, la définition d'un style par une grammaire fournit un ensemble initial de transformations *correctes par définitions* (i.e., ses productions).

Sur la base de cet ensemble initial, nous avons proposé une approche générative pour la caractérisation et la construction de règles correctes. Dans ce but, nous nous sommes intéressés à trois opérations sur les règles de réécriture et nous avons montré qu'elles préservent leur correction sous certaines conditions :

— l'inversion, opération consistant à définir une transformation inverse annulant les effets d'une autre.
— la composition dans son sens classique, qui conserve toujours la correction des transformations.
— la spécialisation, opération conservant la correction et permettant de restreindre les conditions d'applicabilité d'une règle de réécriture et/ou de ses résultats possibles.

Pour améliorer la *représentation des propriétés et de leur dynamisme*, nous avons proposé une **extension des systèmes de réécriture permettant de paramétrer les caractéristiques d'un système aisément et efficacement**. Ceci facilite, comme détaillé par des exemples et scénarios concrets, l'évaluation de la qualité d'un système, sa gestion et son auto-configuration.

La nécessité de l'extension élaborée a été motivée par l'identification de restrictions au sein des approches classiques. Elle a alors été construite autour de la spécification de relations admissibles ainsi que de l'intégration de contraintes et de mutateurs, ces derniers constituant entre autre une méthode légère et flexible pour la modification d'attributs et de contraintes existants.

L'approche proposée est alors expérimentalement évaluée et comparée à diverses méthodes existantes à l'aide de deux outils différents, AGG et GMTE. Reconfigurer un graphe de taille 2500 avec des mutateurs plutôt qu'en usant de méthodes classiques est, **d'après l'étude expérimentale, jusqu'à 27 fois plus rapide sur GMTE et 8,3 fois sur AGG**. Ce gain significatif permet d'efficacement apprécier les caractéristiques du système en combinant évaluation à la demande et mise à jour lors d'une modification. À son tour, cette évaluation améliorée supporte une identification et une appréciation rapide de la qualité d'une configuration, des objectifs pouvant être améliorés ainsi que des composants entraînant une violation de contrainte.

7.1.2 Validation et contributions applicatives

Le modèle proposé a été plus particulièrement appliqué aux contextes du traitement d'évènements complexes (CEP) et des systèmes Machine-à-Machine (M2M).

Pour le CEP, **nous avons décrit les premiers efforts de conception du module d'analyse et de gestion de requêtes (QAM) jouant le rôle de gestionnaire autonome** dans un moteur "CEPaaS", une solution innovante de CEP en tant que service dans des environnements multi-clouds.

QAM repose sur une représentation haut niveau abstrayant opérateurs et requêtes et n'exhibant que les propriétés nécessaires à leur gestion, lui permettant ainsi de palier à la fragmentation des solutions techniques de CEP. Pour ce faire, nous avons examiné les étapes de vie d'une requête et avons identifié les caractéristiques pertinentes en élaborant une **classification innovante des opérateurs de CEP**. Cette conception modulaire et abstraite facilite l'adoption de QAM et étend son champ d'application.

Dans le cadre des systèmes M2M, nous avons présenté un **gestionnaire autonome apte à piloter tout système se conformant au standard ETSI M2M** que nous avons conçu dans le cadre du projet OM2M. Pour ce faire, nous avons élaboré une **vue architecturale multi-modèles couplant la représentation sous-jacente au standard ETSI M2M à notre modèle basé sur les graphes**. Un mécanisme de communications et mises à jour bi-directionnelles assure la cohérence interne de la vue proposée et des deux représentations la composant.

Cette représentation a permis l'élaboration de diverses politiques d'auto-gestion dont un aperçu est fourni sous la forme de deux politiques permettant de prévenir des saturations du réseau ou des pertes de données. Elles rentrent ainsi dans le cadre de l'auto-guérison pro-active du système en incluant des éléments d'auto-optimisation. Élaborées conformément à la méthodologie pour la construction de transformations correctes, elles respectent en sus la propriété d'auto-protection interne du système.

Une première expérimentation reposant sur un système de compteur intelligent démontre la faisabilité de l'approche.

7.1.3 Bilan

Les graphes et leurs règles de réécriture ont un intérêt certain pour la représentation de systèmes dans le cadre de leur gestion autonome. D'un côté, leur caractère visuel facilite une compréhension intuitive des situations mais aussi des actions et transformations envisageables. De l'autre, leur inhérente formalité permet de raisonner rigoureusement, appuyé en cela par un nombre massif de travaux et résultats liés à la théorie des graphes.

La complexité du problème de recherche d'homomorphismes peut rebuter *a priori*. L'étude expérimentale menée dans le cadre de l'extension des systèmes de réécriture offre, outre des valeurs relatives pour comparaison, des résultats absolus en terme de temps de transformations de modèle. Pour AGG, rechercher les sous-motifs pouvant faire l'objet d'une transformation et l'appliquer sur l'un d'eux prends ainsi environ 30 ms pour un système comportant 2500 composants. Ceci répond favorablement aux interrogations quant à la faisabilité et l'extensibilité d'une approche basée sur la réécriture de graphe pour la reconfiguration de systèmes dynamiques.

En pratique, ce type de modèle haut-niveau s'éloigne du système réel. Une solution immédiate et illustrée dans ce manuscrit consiste à intégrer les graphes au sein d'approches multi-modèles, les associant à des représentations plus proches du réel. En particulier, on peut envisager de coupler les graphes à Fractal en les substituant à UML dans des approches UML/Fractal.
Si l'on souhaite conserver une représentation mono-modèle, un couplage peut être atteint en s'inspirant du raisonnement sous-jacent à la définition des mises à jour et communications dans l'approche multi-modèles proposée : raisonner sur les suppressions/additions au niveau modèle et les lier à des actions vers le réel en fonction des attributs des éléments touchés.

7.2 Perspectives

7.2.1 Perspectives directes des travaux

Les contributions proposées ouvrent des perspectives immédiates discutées dans ce manuscrit (maturation et implémentation de QAM, intégration de concurrence et parallélisation dans le M2M, résolution des conflits dus à l'impact de l'extension des systèmes de réécriture sur la méthodologie pour la correction par construction). Outre celles-ci, trois points de développement à court terme nous semble particulièrement intéressants.

Théoriquement, notre modèle peut être étendu pour lui permettre d'englober les multi-graphes (i.e., la possibilité d'avoir plusieurs arcs ayant la même source et la même destination). Ceci permettra de relaxer les contraintes d'applicabilité d'une règle de réécriture et, indirectement, de l'opération de composition. Avec la définition actuelle, l'applicabilité d'une règle composée implique l'existence d'une séquence des règles ayant été composées produisant le même résultat. Grâce à la relaxation des pré-conditions de la composition, une équivalence pourrait être atteinte plutôt qu'une simple implication.

D'un point de vue applicatif, il semble crucial de cacher la complexité inhérente au formalisme au sein d'un outil graphique, le caractère visuel du modèle et des opérations étudiées permettant sa réalisation. Ce dernier devrait implémenter les algorithmes et méthodes opérationnelles proposées tout en guidant l'utilisateur dans leur manipulation. Ceci faciliterait significativement l'adoption et la dissémination de l'approche proposée. Un tel outil pourrait se reposer sur l'existant, en particulier sur les solutions de réécriture et de vérification de modèle proposées par AGG (plus mature, plus efficace et plus répandu que GMTE ou GROOVE).

Finalement, la méthodologie proposée permet de caractériser des règles correctes vis à vis d'un style, mais peut-il en exister d'autres ? La réponse est oui. Pour s'en convaincre, imaginons une grammaire munie des productions $p_1 = AX \to aB$ et $p_2 = aB \to ab$. La règle $\tilde{p}_2 = B \to b$ est correcte vis à vis du langage spécifié, mais ne peut être obtenue à partir d'une composition, d'une inversion ou d'une restriction. En fait, p_2 est une restriction de $\tilde{p}_2$ ajoutant des contraintes d'applicabilité inutiles car toujours satisfaites. Notre approche ne permet pas de caractériser l'ensemble des règles correctes pour une grammaire comportant une telle règle de production. Une deuxième piste théorique intéressante consiste ainsi à étudier la spécification de l'ensemble des règles correctes vis à vis d'un style, et notamment toutes les conditions dans lesquelles notre approche ne permet pas de le construire dans son intégralité.

7.2.2 Aide à la spécification d'une architecture

Dans nos travaux comme dans bien d'autres, il nous faut accepter comme base axiomatique les spécifications d'un utilisateur, sans pouvoir certifier leur validité et en particulier leur adéquation avec le système qu'il tente de caractériser. Cet inconvénient semble incontournable. Néanmoins, la formalisation de ces spécifications dans ce manuscrit, les grammaires de graphe, nécessite un certain niveau d'expertise. Pour faciliter le travail des utilisateurs/architectes, cette difficulté pourrait être atténuée.

Premièrement, il est possible de valider une grammaire en la confrontant à un niveau intermédiaire de spécification reposant sur un modèle plus simple. En ce sens, AGG, par exemple, permet de vérifier la cohérence d'une grammaire vis à vis d'un graphe type et d'un ensemble de contraintes.

Précisons que plusieurs grammaires peuvent satisfaire ces spécifications et que ceci ne permet pas d'assurer qu'une grammaire corresponde effectivement au système considéré par l'utilisateur : ce n'est qu'un niveau de sécurité supplémentaire.

Deuxièmement, il peut être intéressant de vérifier qu'une propriété souhaitée est bel et bien remplie par tous les états considérés comme corrects. En s'appuyant encore une fois sur le caractère génératif des grammaires, on peut envisager un couplage avec des approches basées sur des preuves par théorème démontrant la préservation de certaines propriétés : si une propriété est satisfaite par l'axiome d'une grammaire et si elle est conservée par chacune de ses productions, alors elle est vérifiée par toute instance de la grammaire.

7.2.3 Évaluation et optimisation des reconfigurations

Le coût d'une configuration et son évaluation multi-critères ont été largement étudiés dans la littérature. Une perspective particulièrement intéressante, qui n'est non pas introduite par nos travaux mais plutôt laissée ouverte, est celle de l'évaluation et l'optimisation des reconfigurations elles-mêmes. Lors de processus de décision, il s'agirait alors de confronter le coût de reconfiguration au gain induit par la différence d'évaluation des configurations initiale et finale.

Dans nos expériences, nous investiguons principalement le coût temporel des reconfigurations. D'autres types de coût peuvent néanmoins être considérés. Une reconfiguration peut entraîner une surconsommation énergétique ou monétaire, lorsqu'un cloud publique est utilisé par exemple...

Un coût en terme de robustesse peut également être envisagé si la reconfiguration fait temporairement passer le système par un état intermédiaire vulnérable.

La stabilité du système dans l'état cible peut aussi être un critère pertinent. Par exemple, considérons une fonction d'évaluation f et deux états A et B tels que $f(A) < f(B)$. Imaginons que, du fait du contexte du système, l'état B est instable et implique un retour vers A au bout d'un temps t. Comment le gain $f(B) - f(A)$ pendant un temps t se compare-t-il au coût des deux reconfigurations ?

L'évaluation des transformations ouvre la voie à l'optimisation du processus d'adaptation. En particulier, il peut exister, entre deux états, plusieurs moyens ou chemins de reconfigurations différents. L'étude des moyens de sélection du meilleur d'entre eux constituent une perspective intéressante.

7.2.4 Gestion des états incorrects.

Nos travaux se situent largement dans le contexte de la préservation de la cohérence. Même dans le cadre de la guérison, les états considérés sont toujours corrects et nous ne proposons que des techniques pro-actives permettant de prévenir de futures pertes ou inconstances.

Comment faire lorsque l'état actuel est incorrect ? Nous avons abordé une première piste de résolution en nous intéressant à la génération de points de sauvegarde [1] [139]. Une des stratégies possibles consiste ainsi à rétablir le système dans son état au dernier point de sauvegarde.

Plutôt que de restaurer un état précédent, une méthode plus proche des travaux présentés dans ce manuscrit réside dans la transformation du système afin d'atteindre un état correct. Il faut alors distinguer deux possibilités : soit l'historique récent du système est connu, soit non. Dans ce premier

1. en anglais : *checkpoint*

cas, nous connaissons son dernier état correct ainsi que ce qui a provoqué la perte de la cohérence, typiquement la défaillance d'un composant logiciel ou matériel. Un certain nombre de questions restent ouvertes :

Peut-on s'inspirer de la méthodologie proposée pour la construction de règles préservant la correction pour construire des règles la rétablissant ?

Vaut-il mieux tenter de corriger l'évènement problématique par des suppressions/déploiements/redéploiements pour revenir au dernier état correct ?
En théorie, le système s'optimisant lui même, son dernier état devrait être optimal ou quasi-optimal. Il n'est toutefois pas exclu qu'il soit un optimum local, conservé car le coût de la reconfiguration vers un meilleur état était alors trop élevé au regard des bénéfices attendus. Le retour vers l'état précédent pourrait toutefois constituer une heuristique acceptable. Quoi qu'il en soit, la réponse à cette question doit également intégrer les problématiques de coût de la reconfiguration permettant de retourner vers un état correct ainsi que celles soulevées par la question suivante.

Faut-il se diriger directement vers un nouvel état optimal ou minimiser le temps passé dans des états incorrects (i.e., privilégier un retour rapide vers le correct quitte à instaurer temporairement un état coûteux et rallonger le retour à l'optimal) ?
Dans les faits, il est probable qu'aucune des deux solutions ne soit toujours la meilleure. L'adoption de l'une ou l'autre doit au moins prendre en considération la criticité du système, une criticité élevée encourageant un retour rapide dans le correct.
En outre, toutes les incorrections ne se valent pas et n'handicapent pas le système de la même façon. Dans le cas de DIET, par exemple, une panne du service de nommage est plus critique qu'une panne d'un LA, même si cette dernière entraîne une incorrection au sens strict du terme via la non-gestion d'un ensemble de SED devenant ainsi inatteignables. Une approche intermédiaire pourrait ainsi être envisagée, en corrigeant au plus vite les incohérences critiques, tout en acceptant une correction plus lente des autres.

Annexe A

penser Services globaux pour les Ordinateurs Personnels

Le projet ANR SOP "penser Services globaux pour les Ordinateurs Personnels"[1] répond à la nécessité d'une informatique opérationnelle et efficace avec un minimum d'administrateurs système. Il vise à construire un modèle de fonctionnement hybride pour l'informatique des particuliers et du monde professionnel mélangeant les avantages des modèles d'installation et de gestion locale autonomes avec l'utilisation de ressources distantes.

Le projet SOP a été financé par l'Agence Nationale de la Recherche. Il s'est déroulé de décembre 2011 à février 2015 et a comporté 5 partenaires : le LAAS-CNRS, l'IRIT, SysFera, Degetel et QoSdesign.

A.1 Contexte et objectif du projet

Nous sommes encore dans une phase de révolution dans l'accès et l'usage des technologies de la communication et de l'information. Deux univers de plus en plus convergeant s'y côtoient : le monde des télécommunications, et, plus particulièrement, de la téléphonie mobile d'une part et le monde des ordinateurs personnels de l'autre. Entre ces deux milieux s'insèrent des objets à la connectivité croissante telles que les télévisions et consoles de jeux vidéos.

Au fur et à mesure des générations, objets connectés et téléphones intègrent de plus en plus de services liés à internet tandis que les ordinateurs personnels, via la voix sur IP et les clés 3G, proposent des services de téléphonie et de mobilité. Cette convergence s'est encore accentuée avec le franc succès des smart-phones et des tablettes numériques, conçus pour faciliter leur prise en main par des utilisateurs novices. Cette aisance d'accès est soutenue en particulier par leur modèle d'accession aux services et applications automatisant leur installation, leur configuration et leur mise à jour.

Pour les ordinateurs, nous sommes encore très largement sur un modèle de fonctionnement obsolète où l'utilisateur doit gérer sa machine et son système d'exploitation, acheter ses logiciels, les installer, les mettre à jour... Beaucoup se retrouvent avec un ordinateur mal configuré, rarement sauvegardé, souvent mal protégé contre les agressions ou autres virus, subissant au fil du temps

1. ANR-11-INFR-001, homepage : sop-project.org/

une perte de performances par l'ajout de « verrues » logicielles. S'ensuit l'apparition d'une certaine lassitude vis à vis des problèmes informatiques né de leur mécompréhension. Dans le milieu professionnel, l'appel à une équipe de spécialistes permet d'éviter cette situation au prix de moyens humains conséquents.

Cet environnement est propice à l'introduction d'un nouveau modèle de fonctionnement de l'informatique. Si nous ne pouvons démystifier la gestion d'un système informatique telle qu'elle se présente actuellement afin de la rendre abordable à tous, c'est à elle d'évoluer.

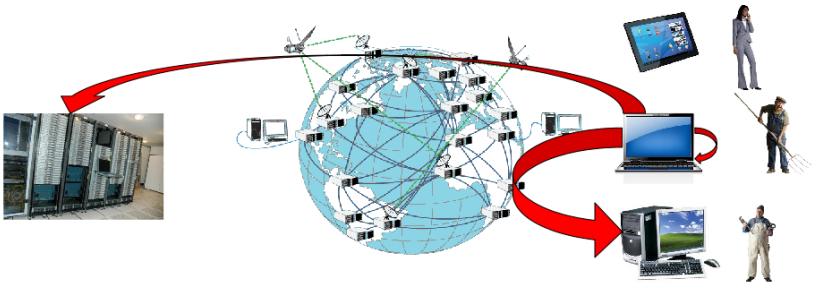


FIGURE A.1 – Modèles d'utilisation des ressources SOP

Le modèle hybride proposé par SOP intègre trois paradigmes d'utilisation de ressources illustrés dans la figure A.1 : *local* avec les ressources de la machine utilisateur, *en nuage* avec des ressources distantes situées dans des data centers dédiés et *communautaire* en usant des ressources disponibles sur les machines d'autres utilisateurs. Il s'appuie pour cela sur les récents acquis de divers domaines :

- la téléphonie mobile en terme d'usage et d'accès aux service
- l'informatique autonome pour la réactivité et l'auto-gestion réduisant les intervenants humains
- l'informatique en nuage pour ses modèles d'accès distant aux ressources
- le développement de la virtualisation dans les centres de ressources informatiques et sur les machines des utilisateurs

Positionnement au sein du projet SOP. La thèse dont les travaux sont présentées dans ce manuscrit a été effectuée et financée dans le cadre du projet SOP. Dans ce contexte, et outre la co-coordination du projet aux cotés de son porteur initial, il s'agissait principalement d'effectuer les tâches suivantes en partenariat avec les membres du consortium :

- concevoir l'architecture logique et logicielle de la solution visée
- mettre en place et maintenir une plate-forme de démonstration
- concevoir et implémenter le gestionnaire mettant en œuvre la gestion autonome du système résultant à l'aide du framework FrameSelf

Un aperçu des travaux menés à bien et des résultats obtenus dans le cadre du projet SOP est présenté par la suite.

A.2 Le modèle de fonctionnement SOP

A.2.1 Introduction, vue utilisateur

Du point de vue de l'utilisateur, celui-ci pourra tout simplement souscrire à un abonnement de type ADSL, lui permettant de recevoir une machine de type PC « léger » ou de déployer le système sur ses propres appareils, que ce soit un ordinateur fixe, portable ou une tablette. Le particulier connecte par la suite son terminal et choisit les applications auxquelles il veut accéder, payantes car liées à des licences, ou libres. Dès lors, ces logiciels sont accessibles de façon transparente, sans que l'utilisateur intervienne dans les politiques de configuration, d'installation ou d'exécution. Une fois qu'une application a été sélectionnée par un utilisateur, elle est installée, configurée puis mise à jour quand nécessaire de sorte à se conformer à sa machine (environnements, ressources mémoire et CPU...).

Pour la partie service, une application logicielle est vue par ses utilisateurs comme un service réseau fourni à travers internet. Ces services sont fournis grâce à des machines virtuelles (VMs [2]) déployées sur les ressources matérielles de centres de calcul, de l'utilisateur ou même d'autres particuliers, inactifs ou collectivisant leur puissance de calcul dans un esprit d'économie d'énergie. Les machines virtuelles nécessaires à la bonne utilisation d'un logiciel sont déployées et redéployées de manière transparente pour l'utilisateur en prenant en compte de multiples critères (caractéristiques du logiciel, qualité de service souhaitée, consommation énergétique...).

Du point de vue administrateur, l'infrastructure logicielle permettant la mise en œuvre de ce modèle de fonctionnement est décrite dans la section suivante.

A.2.2 Architecture logique et logicielle

Dans les faits, les objectifs fixés peuvent être atteint en intégrant les machines d'utilisateurs au sein du nuage afin de les englober dans ses fonctionnalités du nuage en terme à la fois de gestion autonome et d'utilisation des ressources.

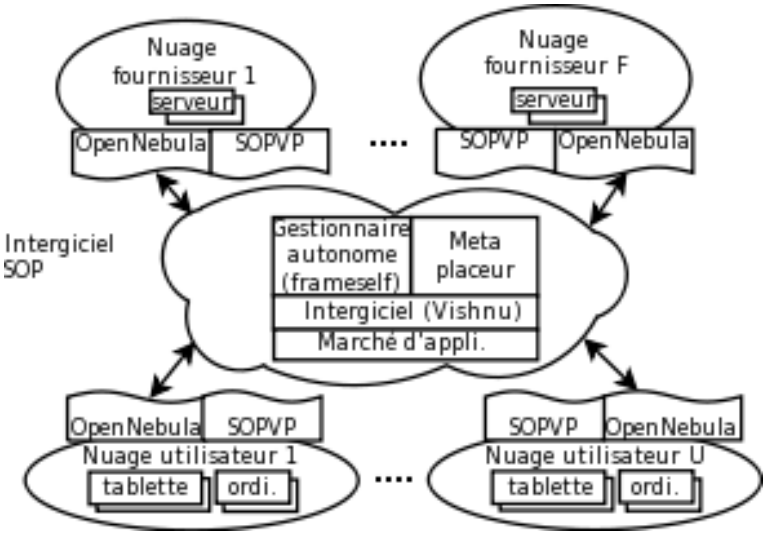


FIGURE A.2 – Architecture logicielle et logique simplifiée de l'intergiciel SOP

2. de l'anglais : *Virtual Machine(s)*

Une vue simplifiée de l'architecture logicielle et logique proposée est illustrée dans la figure A.2. Elle implique divers éléments :

- *OpenNebula*, une plate-forme d'orchestration de nuages informatiques permettant le déploiement de deux types de nuages :
  - des nuages et data-center classiques utilisés selon le paradigme "infrastructure en tant que service". Ils sont possédés et mis à disposition par le fournisseur d'accès et appelés par conséquent "Nuages fournisseurs".
  - des nuages constitués des machines utilisateurs appelés "Nuages utilisateurs".
- *SOPVP*[140], une extension du placeur interne d'OpenNebula. SOPVP met en œuvre les politiques d'utilisation des ressources propres au projet SOP au sein de chacun des nuages. Il considère des contraintes de consommation énergétique et de qualité de service.
- *Vishnu* [3], un logiciel permettant la fédération de nuages et l'exécution de tâches en leur sein. Vishnu intègre un méta-placeur développé pour le projet SOP. Ce dernier se base sur des métriques agrégées pour choisir le nuage le plus approprié pour l'exécution de chaque tâche.
- Un marché d'applications guidant les utilisateurs dans leur accès aux applications et récompensant le prêt de ressources.
- Un gestionnaire autonome guidant les évolutions de cette architecture.

Le gestionnaire autonome de l'intergiciel SOP a été conçu et implémenté dans le cadre de la thèse présentée dans ce manuscrit. Il a été réalisé à l'aide de FrameSelf[141], un framework permettant d'implémenter des gestionnaire autonomes se basant sur l'architecture proposée par IBM et présentée dans la section 2.1.3.

Un description plus détaillée de ce gestionnaire se focalisant sur ses interactions avec les autres éléments de l'architecture SOP est proposée dans la section suivante.

A.2.3 Contribution technique : le gestionnaire autonome SOP

La figure A.3 précise la position du gestionnaire autonome au sein de l'intergiciel SOP et ses interactions avec les autres briques logicielles.

Le gestionnaire peut agir :
- sur chaque instance d'*OpenNebula* à l'aide de requêtes dédiées. Il peut ainsi récupérer des informations et agir sur les hôtes, les VMs, les réseaux virtuels...
- sur *Vishnu* à l'aide de requêtes dédiées. Il peut ainsi agir sur les tâches en les démarrant, les arrêtant, les suspendant et les re-soumettant. Il peut également solliciter une décision du méta-placeur pour trouver un nuage utilisable.
- directement sur les machines. Il peut ainsi éteindre une machine d'un utilisateur ou du fournisseur et allumer une machine d'un nuage fournisseur.

Il peut recevoir des événements :
- de chaque utilisateur, le notifiant du démarrage ou de l'arrêt d'une application, de la mise à disposition ou non de ses ressources, de sa volonté d'arrêter sa machine...
- du placeur SOPVP, le notifiant d'un échec de placement, de la nécessité d'éteindre ou d'allumer une machine d'un nuage fournisseur...
- d'*OpenNebula*, le notifiant de l'arrêt d'une VM.

3. Vishnu Distributed Resource Management Middleware, homepage : sysfera.github.io/Vishnu.html

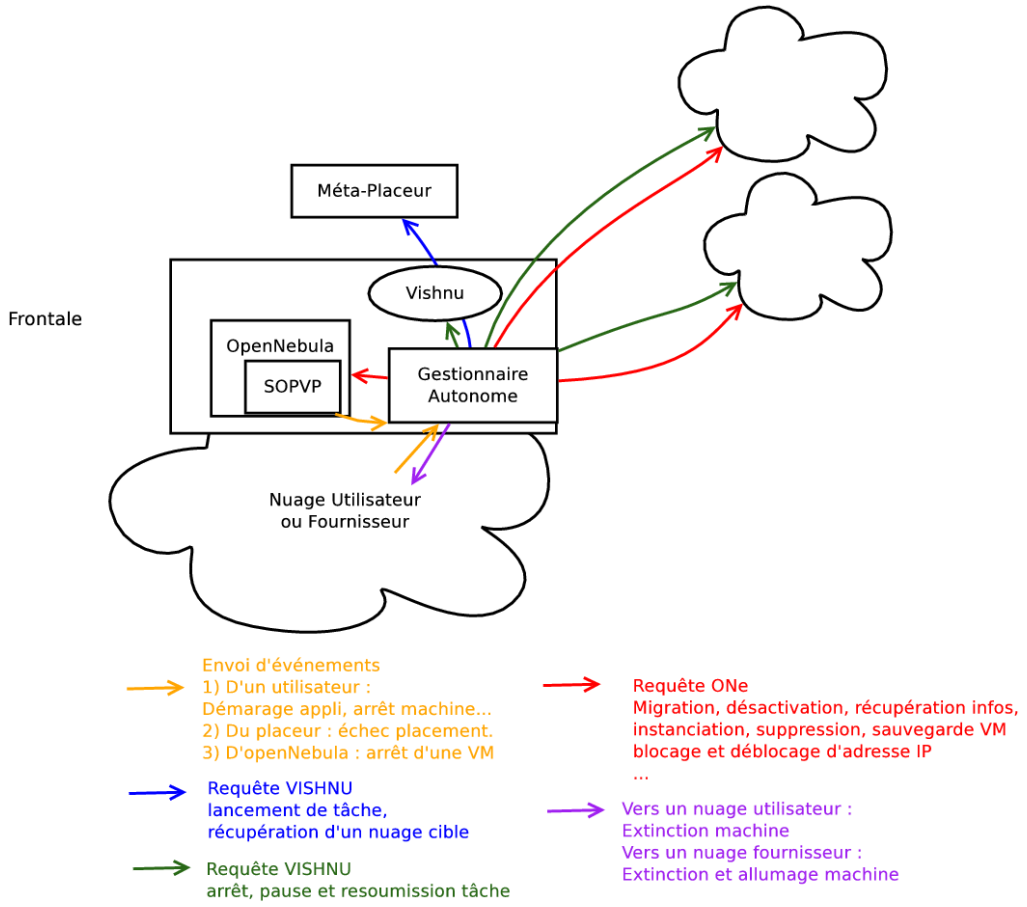


FIGURE A.3 – Gestionnaire autonome et ses interactions au sein de l'intergiciel SOP

Des scénarios d'utilisation précisant certaines de ces interactions et actions du gestionnaire autonome sont détaillés dans la section suivante.

A.3 Démonstrateur et validation

Les fonctionnalités de la plate-forme résultante ont été testées et validées au travers de scénarios d'utilisation exécutés sur d'une plateforme expérimentale de démonstration composée d'un nuage utilisateur et d'un nuage fournisseur de 3 machines chacun.

Les trois scénarios les plus pertinents sont détaillés ci-après. Ils sont contigües : l'état final d'un scénario est l'état initial du suivant.

Le premier scénario, illustré dans la partie gauche de la figure A.4, décrit l'installation d'une nouvelle application. Il se déroule comme suit.

1. Pendant qu'il parcourt le marché, l'utilisateur 1 découvre *scilab*, une application qu'il souhaite utiliser. Il actionne par conséquent le bouton "télécharger & installer".
2. Une fois toutes les actions de facturation pertinentes effectuées, l'application est :

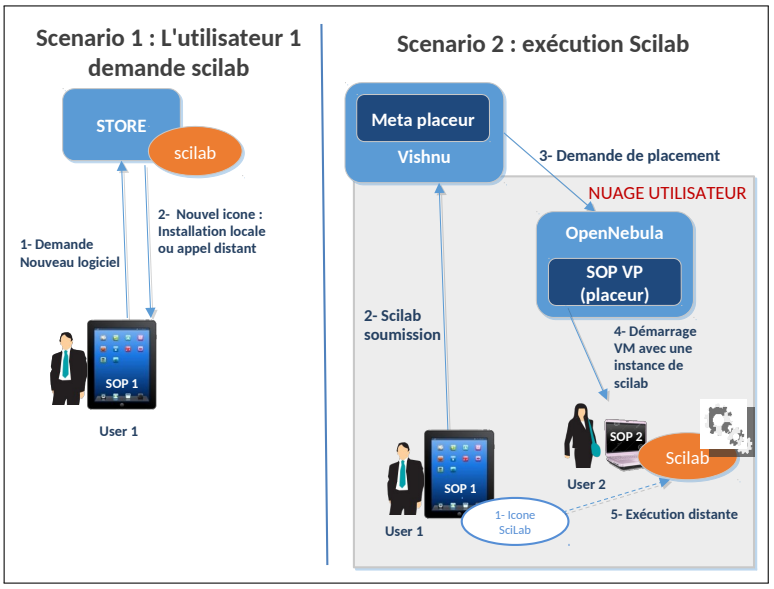


FIGURE A.4 – Utilisation du marché et exécution communautaire

— soit téléchargée et installée sur sa machine. Ce processus est lancé si l'application peut être exécutée localement. Pendant son installation, l'utilisateur peut également se connecter à une VM distante lui permettant d'accéder sans attendre à l'application.
— soit rendue accessible à distance pour l'utilisateur. La machine SOP1 étant une tablette comportant peu de ressources, cette option est ici choisie.

Les deux options résultent finalement en la création d'un bouton de lancement sur le bureau de l'utilisateur.

Le second scénario, illustré dans la partie droite de la même figure, décrit le processus permettant le lancement d'une application, i.e. le déploiement d'une VM l'exécutant.

1. L'utilisateur 1 veut lancer l'application qu'il a fraîchement acquis. Il actionne le bouton "*scilab*" sur son bureau et sélectionne un script qu'il souhaite exécuter.
2. En fonction des données saisies par l'utilisateur, une commande *Vishnu* est générée avec les paramètres appropriés.
3. Le meta-placeur intégré à *Vishnu* sélectionne le nuage le plus adéquat pour l'exécution de cette tâche, et envoie une requête demandant l'instanciation d'une VM appropriée.
4. Cette VM est instanciée et déployée sur un hôte selectionné par SOPVP. La machine SOP1 étant une tablette ne possédant pas assez de ressources de calcul, SOPVP choisit ici la machine SOP2 de l'utilisateur 2.
5. Le script *scilab* soumis par l'utilisateur 1 à partir de sa machine SOP1 est exécuté à distance dans cette nouvelle VM. Son résultat est mis à disposition de l'utilisateur 1 dès son obtention.

Le troisième scénario, dont l'acteur principal est le gestionnaire autonome, est illustré dans la figure A.5. Il consiste en l'extinction de la machine SOP2, ce qui induit une migration d'une VM du nuage utilisateur vers le nuage fournisseur.

Une attention particulière est donnée à la gestion des adresses IP de chaque VM. Pour éviter le déploiement de solutions plus complexe de type "réseau définis logiciellement"[4], les deux nuages

4. en anglais : *software defined network*

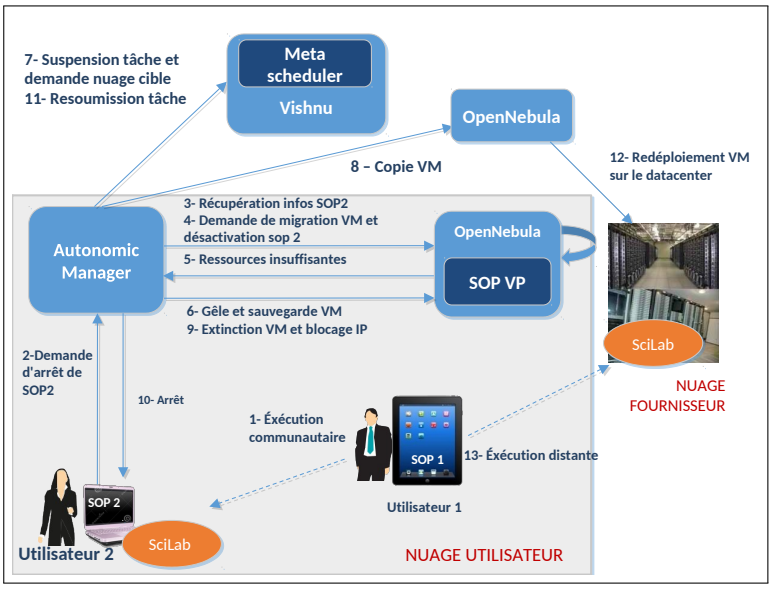


FIGURE A.5 – Extinction machine et migration inter-nuages

du démonstrateur utilisent un plan d'adressage cohérent. Chaque nuage possède sa propre plage d'adresse attribuable à une nouvelle VM.

La migration inter-nuage s'effectue alors comme suit.

1. Une VM exécutant un script *scilab* pour l'utilisateur 1 est toujours en cours sur la machine de l'utilisateur 2.
2. L'utilisateur 2 veut arrêter sa machine et active le bouton correspondant, ce qui envoie un événement au gestionnaire autonome.
3. Le gestionnaire autonome récupère la liste des VMs exécutées sur SOP2 et vérifie leurs propriétaires. Si elles sont toutes possédées par l'utilisateur 2, la machine est effectivement arrêtée (cf. étape 10). Si un autre utilisateur utilise les ressources de SOP2, ce qui est ici le cas, son arrêt est suspendu pour laisser place à une migration des VMs correspondantes.
4. Différents appels à *OpenNebula* sont alors effectués par le gestionnaire autonome :
   — la machine SOP2 est passée dans l'état "disable". Cet état n'impacte pas l'exécution des VMs courantes, mais interdit toute nouvelle instanciation.
   — une requête de migration est envoyée pour chaque VM possédée par un utilisateur autre que le propriétaire de la machine. Ces VMs sont mises dans l'état "reschedule" et traitées par SOPVP lors de sa prochaine boucle de placement. Ici, la seule VM concernée est celle contenant *scilab* et possédée par l'utilisateur 1.
5. Comme la seule autre machine du nuage, SOP1, ne peut supporter l'exécution de la VM, celle-ci ne peut être migrée à l'intérieur du nuage utilisateur. SOPVP notifie le gestionnaire autonome afin qu'il initie le processus de migration inter-nuages.
6. Le gestionnaire autonome envoie une requête à *OpenNebula* afin de geler et sauvegarder la VM.
7. Le gestionnaire informe *Vishnu* que la tâche correspondante est suspendue et demande au meta-placeur de sélectionner un nuage cible.
8. Le gestionnaire autonome copie le point de sauvegarde de la VM sur le nuage cible et y créé un template approprié. Ce template contient l'adresse IP de la VM d'origine, adresse qui est libérée sur le nuage cible afin d'être rendue utilisable.

9. Le gestionnaire autonome supprime la VM initiale et rend son adresse IP indisponible sur le nuage utilisateur.
10. Comme aucune VM n'est exécutée sur SOP2, le gestionnaire autonome l'arrête.
11. Le gestionnaire autonome informe *Vishnu* que la tâche est prête à être soumise dans le nuage cible.
12. La VM est instanciée au sein du nuage cible sur un hôte sélectionné par SOPVP.
13. La VM exécutant le script *scilab* pour l'utilisateur 1 est à présent instanciée sur le nuage fournisseur.

Lors de l'arrêt d'une VM, si son template indique qu'elle est le produit d'une migration inter-nuage, son adresse IP est bloquée sur le nuage courant et libérée sur le nuage d'origine.

Dissémination du projet

Un site web [5] a été réalisé pour le projet. Il en présente les partenaires, le contexte et les objectifs. Une vidéo [6] présentant le projet, son architecture et son fonctionnement au travers de scénarios d'utilisation y est également disponible.

Plusieurs articles portant chacun sur un aspect spécifique du projet ont été publiées dans son cadre. Un article détaillant l'intégralité du projet, de son architecture et de son fonctionnement a été accepté et est à paraître dans le "Journal of Grid Computing" [142].

5. sop-project.org/
6. la vidéo sans commentaires peut être directement consultée sur : https ://youtu.be/PZwgUu-XLqY

Annexe B

Preuves des théorèmes du chapitre 4 : auto protection

B.1 Inversion : preuve du théorème 4.2

Démonstration. Soit une grammaire augmentée GG, une règle $r = L_r \xrightarrow{h_r} R_r$ avec $h_r : L_r^I \to R_r$ correcte pour GG et $r^{-1} = L_{r^{-1}} \xrightarrow{h_{r^{-1}}} R_{r^{-1}}$ avec $h_{r^{-1}} : L_{r^{-1}}^I \to R_{r^{-1}}$ l'inverse de r.

Supposons les hypothèses du théorème 4.2 vérifiées et montrons que r^{-1} est correcte pour GG. Pour cela, supposons que r^{-1} est applicable à une configuration G correcte pour GG et montrons que le résultat est une autre configuration correcte.

Soit $G = (V, E)$ une instance de GG telle qu'il existe un homomorphisme $h = (f_h, I_H)$ selon lequel r^{-1} est applicable à G.

Par définition, $h : L_{r^{-1}} \to G$ et $L_{r^{-1}} = R_r$. D'après la troisième hypothèse du théorème :
$(\exists n \in \mathbb{N}, \exists((p_i, h_i))_{i\in[1,n]}, G = p_n\_h_n.(\dots).p_1\_h_1(AX_{GG}))\wedge$
$(\exists k \in [1,n], p_k = r \wedge \forall j \in [1,n], j \neq k \implies p_j \in P_{GG})\wedge$
$(\forall v \in V_{R_r}, (v \in Dom(f_r) \wedge f_h(v) = f_{h_k}(f_r^{-1}(v))) \vee f_h(v) = v)$. (i)

Pour tout $i \in [0,n]$, notons :
- $G_i = p_{(n-i)}\_h_{(n-i)}.(\dots).p_1\_h_1(AX_{GG}))$, de sorte que $G_0 = G$
- $P(i)$ la propriété "r^{-1} est applicable à G_i selon h et $r^{-1}\_h(G)$ peut être obtenu à partir de $r^{-1}\_h(G_i)$ en appliquant successivement, s'il en existe, les règles (p_j) avec j vérifiant $n-i+1 \geqslant j \geqslant n$" suivant l'homomorphisme h_j.
  C'est à dire que :
  - si $i = 0$, $r^{-1}\_h(G) = r^{-1}\_h(G_0)$,
  - si $i = 1$, $r^{-1}\_h(G) = p_n\_h_n.r^{-1}\_h(G_1)$,
  - sinon $r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(n-i+1)}\_h_{(n-i+1)}.r^{-1}\_h(G_i)$.

Par récurrence, prouvons que $P(i)$ est vraie pour tout $i \in \mathbb{N}$ tel que $n-k > i \geqslant 0$. D'après (i), $P(0)$ est vraie. Pour $n-k > j \geqslant 0$, supposons $P(j)$ et montrons que $P(j+1)$.

Pour cela, montrons que r^{-1} est applicable à G_{j+1} selon h et que
$r^{-1}\_h(G_j) = p_{n-j}\_h_{n-j}.r^{-1}\_h(G_{j+1})$.

B.1.1 Applicabilité de r^{-1}

h est un homomorphisme vers G_{j+1}. Comme $P(j)$, r^{-1} est applicable à $G_j = p_{n-j}\_h_{n-j}(G_{j+1})$ selon h.

Du fait de la simplification relative aux parties invariante décrite dans la remarque 4.3, il est immédiat que h est un homomorphisme de $L_{r^{-1}}$ vers G_{j+1} si aucun nœud ou arc matché par h dans G_j est ajouté lors de l'application de p_{n-j} à G_{j+1}. Par définition de h ,c'est un homomorphisme de $L_{r^{-1}}$ vers G_{n-k}.

Tout nœud de $f_h(V_{L_{r^{-1}}})$ est donc présent dans G après l'application de r à G_{n-k} selon h_k et ne peut donc être ajouté par une règle p_m tel que $n \geqslant m > k$. Comme $n-k > j \geqslant 0$, $n-j > k$ et aucun nœud de $f_h(V_{L_{r^{-1}}})$ n'est ajouté par l'application de p_{n-j}.

Supposons qu'un arc $e \in f_h(E_{L_{r^{-1}}})$ de G_j soit ajouté par p_{n-j}. Comme h est un homomorphisme de $L_{r^{-1}}$ vers G_{n-k} par définition, cet arc est présent dans G_{n-k}. Par conséquent, s'il est ajouté par p_{n-j}, il existe une transformation (p_m, h_m) avec m vérifiant $n-j > m > k$ et supprimant l'arc e. D'après (i), $p_m \in P_{GG}$ et ceci est donc impossible selon l'hypothèse 1 du théorème.

Ainsi h est un homomorphisme de $L_{r^{-1}}$ vers G_{j+1}. Sa cohérence est immédiate du fait de celle de h de $L_{r^{-1}}$ vers G_{j+1} (G_{j+1} intégrant les mêmes identifications que G_j plus celles induites par I_{n-j}).

Satisfaction de la seconde condition d'applicabilité. Supposons que h ne vérifie pas la seconde condition d'applicabilité pour G_{j+1}. Cela signifie qu'il existe $(v_1, v_2) \in (V_{L^I_{r^{-1}}})^2 \backslash E_{L_{r^{-1}}}$ tel que $f_h((v_1, v_2)) \in E_G$. Comme la seconde condition d'applicabilité est vérifié par h pour r^{-1} et G_j, $f_h((v_1, v_2)) \notin E_{G_j}$. L'arc $f_h((v_1, v_2))$ est donc supprimé lors de l'application de p_{n-j}. D'après (i), $p_{n-j} \in P_{GG}$. Comme $f_h(v_1)$ et $f_h(v_2)$ sont invariants pour p_{n-j}, ceci est impossible selon l'hypothèse 1 du théorème.

L'homomorphisme h remplit dans la seconde condition d'applicabilité pour r^{-1} et G_{j+1}. Les morphismes considérés étant des homomorphismes, la troisième condition est automatiquement remplie. Ainsi, r^{-1} est applicable à G_{j+1} selon h.

B.1.2 Applicabilité de p_{n-j}

h_{n-j} est un homomorphisme vers $r^{-1}\_h(G_{j+1})$. D'après (i), p_{n-j} est applicable selon h_{n-j} à G_{j+1}.

Du fait de la simplification relative aux parties invariante décrite dans la remarque 4.3, il est immédiat que h_{n-j} est un homomorphisme de $L_{p_{n-j}}$ vers $r^{-1}\_h(G_{j+1})$ si aucun nœud ou arc matché par h_{n-j} dans G_{j+1} n'est supprimé lors de l'application de r^{-1} à G_j selon h.

Supposons qu'un élément $el \in f_{h_{j-1}}(V_{L_{pn-j}}) \cup f_{h_{j-1}}(E_{L_{p_{n-j}}})$ soit explicitement supprimé par r^{-1}. Il existe donc un élément $el_r \in V_{L_{r^{-1}}} \cup E_{L_{r^{-1}}}$ tel que $f_h(el) = el$.

Comme r^{-1} est applicable à G_j selon h, el est nécessairement invariant vis à vis de p_{n-j}. Ainsi, comme GG est une grammaire augmentée, $(Att_{G_j})^{el}_0 = (Att_{G_{j+1}})^{el}_0 + 1$.

Par définition de l'augmentation, $Att_0^{el_r}$, $(Att_{G_j})_0^{el}$ et $(Att_{G_{j+1}})_0^{el}$ sont des constantes. Comme r^{-1} est applicable à la fois à G_j et G_{j+1} selon h, $Att_0^{el_r}$ est identifiable à la fois à $(Att_{G_j})_0^{el}$ et $(Att_{G_{j+1}})_0^{el}$. D'où $(Att_{G_j})_0^{el} = (Att_{G_{j+1}})_0^{el}$.

On obtient une contradiction et donc aucun nœud ou arc matché par h_{n-j} dans G_{j+1} n'est explicitement supprimé lors de l'application de r^{-1} à G_j selon h. Pour qu'un arc soit implicitement supprimé, il faut qu'une de ses extrémités le soit également, ce qui n'est donc pas possible.

Par conséquent, h_{n-j} est un homomorphisme de $L_{p_{n-j}}$ vers $r^{-1}\_h(G_{j+1})$. Comme r^{-1} est applicable selon h à $p_{n-j}\_h_{n-j}(G_{j+1})$, un graphe contenant les identifications induites par I_{n-j}, $I_{n-j} \cup I_h$ n'est pas incohérent et h_{n-j} est un homomorphisme cohérent de $L_{p_{n-j}}$ vers $r^{-1}\_h(G_{j+1})$.

Satisfaction de la seconde condition d'applicabilité. Supposons que h_{n-j} ne remplisse pas la seconde condition d'applicabilité pour p_{n-j} et $r^{-1}\_h(G_{j+1})$.

Il existe donc un arc $e \in f_{n-j}(V_{L_{p_{j+1}}})^2$ ajouté lors de l'application de p_{n-j} selon h_{n-j} et présent dans $r^{-1}\_h(G_{j+1})$. Comme p_{n-j} est applicable selon h_{n-j} à G_{j+1}), e n'est pas présent dans G_{j+1}) : il a donc été ajouté par r^{-1} lors de son application selon h.

Or, comme e est ajouté lors de l'application de p_{n-j} selon h_{n-j}, e est présent dans G_j, graphe auquel r^{-1} est applicable selon h, ce qui est impossible si l'application de r^{-1} selon h ajoute e.

h_{n-j} remplit donc la seconde condition d'applicabilité pour p_{n-j} et $r^{-1}\_h(G_{j+1})$. Encore une fois, la satisfaction de la troisième condition d'applicabilité est triviale comme les morphismes considérés sont des homomorphismes.

Par conséquent, p_{n-j} est applicable à $r^{-1}\_h(G_{j+1})$ suivant h_{n-j}.

B.1.3 Égalité entre $r^{-1}\_h(G_j)$ et $p_{n-j}\_h_{n-j}.r^{-1}\_h(G_{j+1})$

Par définition de G_j, cette égalité se ramène à $r^{-1}\_h.p_{n-j}\_h_{n-j}(G_j + 1) = p_{n-j}\_h_{n-j}.r^{-1}\_h(G_{j+1})$.

Les deux règles restant applicables selon les mêmes homomorphismes lorsqu'interchangées, il est immédiat que les éléments explicitement supprimés ou ajoutés sont les mêmes.

Comme aucune règle ne supprime un élément matché par l'autre, elles ne peuvent pas supprimer implicitement d'arc ajouté par l'autre.

Trivialement, $r^{-1}\_h.p_{n-j}\_h_{n-j}(G_j + 1) = p_{n-j}\_h_{n-j}.r^{-1}\_h(G_{j+1})$ et donc
$r^{-1}\_h(G_j) = p_{n-j}\_h_{n-j}.r^{-1}\_h(G_{j+1})$.

Comme $P(j)$, $r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(n-j+1)}\_h_{(n-j+1)}.r^{-1}\_h(G_j)$.

D'où $r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(n-j+1)}\_h_{(n-j+1)}.p_{n-j}\_h_{n-j}.r^{-1}\_h(G_{j+1})$.

Ainsi, $P(j+1)$ est vraie.

Par conséquent, $P(i)$ est vraie pour tout $i \in \mathbb{N}$ tel que $n - k \geqslant i \geqslant 0$. En particulier, pour $i = n - k$, ceci donne :

r^{-1} est applicable à $r\_h_k(G_{n-k+1})$ selon h (ii) et
$r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(k+1)}\_h_{(k+1)}.r^{-1}\_h.r\_h_k(G_{n-k+1})$ (iii).

Considérons la propriété 4.1. Ici, il existe un homomorphisme h_k selon lequel r est applicable à G_{n-k+1} et, d'après l'hypothèse 2 du théorème, l'application de r ne peut donner lieu à l'apparition d'arc suspendu. Alors, d'après la propriété sus-référencée, r^{-1} est applicable à $r\_h_k(G_{n-k+1})$ selon un homomorphisme dont la caractérisation est celle de h (d'après la propriété (1)), selon lequel r^{-1} est applicable à $r\_h_k(G_{n-k+1})$ tel que G_{n-k+1} et $r^{-1}\_h.r\_h_k(G_{n-k+1})$ sont isomorphes. Ici, du fait de la remarque 4.3 et de la définition d'inversion, il est immédiat que l'isomorphisme en question est caractérisé par la fonction identité et l'ensemble d'identification $I_h \cup I_k$. Par conséquent,

$$r^{-1}\_h.r\_h_k(G_{n-k+1}) = Aff_{(I_h \cup I_k)}(G_{n-k+1}).$$

Ceci et (iii) donnent $r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(k+1)}\_h_{(k+1)}(Aff_{(I_h \cup I_k)}(G_{n-k+1}))$. En intégrant $I_h \cup I_k$ dans I_{k+1} de sorte que $\tilde{h}_{k+1} = (f_{k+1}, I_{k+1} \cup I_h \cup I_k)$, on obtient donc :

$$r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(k+1)}\_\tilde{h}_{(k+1)}(G_{n-k+1}).$$

En remplaçant G_i par sa définition explicite, on a donc
$r^{-1}\_h(G) = p_n\_h_n.(\dots).p_{(k+1)}\_\tilde{h}_{(k+1)}.p_{(k-1)}\_h_{(k-1)}.(\dots).p_1\_h_1(AX_{GG})$.

D'après (i) (qui provient de l'hypothèse 3 du théorème), $\forall j \in [1, n], j \neq k \implies p_j \in P_{GG}$. Par définition d'une instance d'une grammaire, $r^{-1}\_h(G)$ est donc une instance de GG, i.e., une configuration correcte pour GG.

Par conséquent, r^{-1} est correcte pour GG.

□

B.2 Composition : preuve du théorème 4.4

Une fois encore, la preuve est donnée dans le cas le plus général où la recherche de motif est formalisée par la recherche d'un morphisme. La démonstration peut aisément être transposée aux autres approches : en particulier, les conditions 5 et 6 ne seront utilisées que lorsque strictement nécessaires.

Démonstration. Soient deux règles de réécriture p et q ainsi qu'une compatibilité C vérifiant les conditions d'applicabilité de l'algorithme 4.2. Notons r la règle $p \circ_C q$.

Montrons que r est bien une règle composée de p et q. Pour cela, supposons qu'il existe un morphisme $m_r^L = (f_r^L, I_r^L)$ selon lequel r est applicable à un graphe G quelconque.

En reprenant les notations introduites dans l'algorithme 4.2, montrons qu'il existe un couple de morphismes (m_p^L, m_q^L) de sorte que q est applicable à G selon m_q^L, que p est applicable à $q\_m_q^L(G)$ selon m_p^L et $p\_m_p^L.q\_m_q^L(G) = r\_m_r^L(G)$.

B.2.1 Il existe un morphisme m_q^L selon lequel q est applicable à G

Existence d'un morphisme cohérent. Comme $r = p \circ_C q$, $L_r = L_q \uparrow_{(f_q, I)} L_p^S$. D'après la remarque 3.2, L_r est invariant par Aff_I et il existe un morphisme, caractérisée par :

- une certaine fonction f_q^+, coïncidant avec f_q sur $\{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}$ et l'identité sur le reste de V_{L_q},
- l'ensemble d'identification I,

tel que $L_q \xrightarrow{(f_q^+, I)} L_r$ et $L_r = Aff_I(L_r)$.

Par hypothèse, $L_r \xrightarrow{m_r^L} G$. D'après la remarque 3.1, $m_q^L = (f_r^L \circ f_q^+, I \cup I_r^L)$ est un morphisme cohérent de L_q vers G.

Satisfaction de la seconde condition d'applicabilité. Supposons que m_q^L ne remplisse pas la seconde condition d'applicabilité, soit $\exists (v_1^q, v_2^q) \in V^2_{L_q^I}, (v_1^q, v_2^q) \notin E_{L_q} \wedge f_q((v_1^q, v_2^q)) \in E_{R_q} \wedge f_q^L((v_1^q, v_2^q)) \in E_G$ (1). Par hypothèse, m_r^L remplit cette seconde condition et : $\forall (v_1^r, v_2^r) \in V^2_{L_r^I}, (v_1^r, v_2^r) \notin E_{L_r} \wedge f_r((v_1^r, v_2^r)) \in E_{R_r}) \implies f_r^L((v_1^r, v_2^r)) \notin E_G$. (2)

Par disjonction de cas :

Supposons que $v_1^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ et $v_2^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ (3.1).

D'après la définition de f_q^+ on a alors $f_q^+((v_1^q, v_2^q)) = (v_1^q, v_2^q)$ et donc $(v_1^q, v_2^q) \in V^2_{L_r}$.

Par construction de $V_{L_r^I}$, $\forall v \in V_{L_r} \cap V_{L_q}, v \in V_{L_q^I} \implies v \in V_{L_r^I}$. Ceci et (1) donnent $(v_1^q, v_2^q) \in V^2_{L_r^I}$ (3.2)

Par construction de L_r et par définition de l'expansion, $\forall e \in V^2_{L_q} \cap V^2_{L_r}, e \in E_{L_q} \Leftrightarrow e \in E_{L_r}$, comme (1) alors $(v_1^q, v_2^q) \notin E_{L_r}$ (3.3).

Supposons qu'il existe un nœud de $L_p^{C_q}$ dont l'image par f est $f_q(v_1^q)$ ou $f_q(v_1^q)$. Par définition de R_q^I, $\forall v \in V_{L_q^I}, f_q(v) \in V_{R_q^I}$, donc $f_q((v_1^q, v_2^q)) \in V^2_{R_q^I}$. Par définition de la restriction, tout nœud de

R_q^I étant l'image d'un nœud de $L_p^{C_q}$ par f est dans $L_p^{C_q} \downarrow_{(f,I)} R_q^I$. Par conséquent, $f_q(v_1^q)$ ou $f_q(v_2^q)$ est dans $L_p^{C_q} \downarrow_{(f,I)} R_q^I$, la partie droite de r_L. Par définition des règles de réécriture et de L_p^S, $f_q(v_1^q)$ ou $f_q(v_2^q)$ est également dans L_p^S. Ceci contredit (3.1), par conséquent il n'existe pas de nœud de L_p^C dont l'image par f est $f_q(v_1^q)$ ou $f_q(v_2^q)$ (3.4).

D'après (1), $f_q((v_1, v_2)) \in E_{R_q}$ et d'après (3.4), $f_q(v_1^q) \notin Im(f) \wedge f_q(v_2^q) \notin Im(f)$. Par construction, $R_r = r_{R\_}(f, I)(R_q)$. Par définition des règles de réécriture, un arc ne peut être supprimé ou modifié lors de l'application d'une règle si aucune de ses extrémités ne fait partie de l'image du morphisme selon laquelle la règle est appliquée. Ainsi, $f_q((v_1^q, v_2^q)) \in E_{R_r}$. Comme f_r coïncide avec f_q sur $V_{L_q^I}$, $f_r((v_1^q, v_2^q)) \in E_{R_r}$ (3.5).

D'après (3.2), la relation (2) est vrai en particulier pour $(v_1^r, v_2^r) = (v_1, v_2)$ et donc, avec (3.3) et (3.5), $f_r^L((v_1^q, v_2^q)) \notin E_G$. D'après (1) $f_q^L((v_1^q, v_2^q)) \in E_G$. Or $f_q^L((v_1^q, v_2^q)) = f_r^L \circ f_q^+((v_1^q, v_2^q) = f_r^L((v_1^q, v_2^q))$. On obtient donc une contradiction et m_q^L remplit la seconde condition d'applicabilité pour ce type d'arc.

Supposons que $(v_1^q, v_2^q) \in \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}^2$ **(4.1).**

Par définition de R_q^I, $f_q((v_1^q, v_2^q)) \in V^2_{R_q^I}$ (4.2) et d'après la définition de f_q^+ on a alors $f_q^+((v_1^q, v_2^q)) = f_q((v_1^q, v_2^q))$.

Par hypothèse, la pré-condition 4. de l'algorithme de composition est vérifiée et d'après (1), elle donne, pour v_1^q et v_2^q : $(\forall v_p \in V_{L_p}, (f(v_p) = f_q(v_1^q) \vee f(v_p) = f_q(v_2^q)) \implies v_p \in V_{L_p^I})$ (4.3) $\wedge$ $(\forall (v_1^p, v_2^p) \in V^2_{L_p}, f((v_1^p, v_2^p)) = f_q((v_1^q, v_2^q)) \implies ((v_1^p, v_2^p) \notin E_{L_p} \vee (v_1^p, v_2^p) \in E_{L_p^I})$ (4.4).

D'après (4.2) et (4.3), $f_q(v_1^q)$ et $f_q(v_2^q)$ remplissent la condition P_1 de l'algorithme Inv_V, et donc $f_q((v_1^q, v_2^q)) \in V^2_{L_r^I}$ (4.5).

Par définition des règles de réécriture, tout nœud du graphe résultant de l'application d'une règle n'étant pas dans le graphe initial est nécessairement dans la partie droite de la règle. Or, d'après (1) $f_q((v_1^q, v_2^q)) \in V^2_{R_q}$, donc $f_q((v_1^q, v_2^q)) \notin V^2_{L_p}$. Par conséquent, comme $f_q((v_1^q, v_2^q) \in V^2_{L_p^S}$, $f_q((v_1^q, v_2^q)$ est un couple de nœuds de la partie droite de r_L, soit $f_q((v_1^q, v_2^q)) \in V^2_{L_p^{C_q^I} \downarrow_{(f,I)} R_q^I}$ (4.6).

D'après (1), $(v_1^q, v_2^q) \notin E_{L_q}$ donc, par définition de R_q^I, $f_q((v_1^q, v_2^q)) \notin E_{R_q^I}$. Ceci implique, par définition de la restriction, que $f_q((v_1^q, v_2^q))$ n'est pas un arc de $L_p^{C_q^I} \downarrow_{(f,I)} R_q^I$, la partie droite de r_L. Par définition des règles de réécriture, pour que $f_q((v_1^q, v_2^q))$ soit un arc de L_p^S, il faut qu'il soit l'image d'un arc présent dans E_{L_p} (le graphe auquel la règle est appliquée) mais pas dans $E_{L_p^{C_q}}$ (la partie gauche de la règle). Soit : $\exists (v_1^p, v_2^p) \in V^2_{L_p} \cap V^2_{L_p^{C_q}}, (v_1^p, v_2^p) \in E_{L_p} \wedge (v_1^p, v_2^p) \notin E_{L_p^{C_q}} \wedge f((v_1^p, v_2^p)) = f_q((v_1^q, v_2^q))$. Or, selon (1), $f_q((v_1^q, v_2^q)) \in E_{R_q}$, par définition de $L_p^{C_q}$, $\exists (v_1^p, v_2^p) \in V^2_{L_p}, (v_1^p, v_2^p) \in E_{L_p} \wedge f((v_1^p, v_2^p)) \in E_{R_q} \implies (v_1^p, v_2^p) \in E_{L_p^{C_q}}$. On obtient donc une contradiction et par conséquent, $f_q((v_1^q, v_2^q)) \notin E_{L_p^S}$. Par définition de l'expansion, comme $(v_1, v_2) \notin E_{L_q} \wedge f_q((v_1^q, v_2^q)) \notin E_{L_p^S}$ on obtient $f_q((v_1^q, v_2^q)) \notin E_{L_q \uparrow_{(f_q,I)} L_p^S}$, soit $f_q((v_1^q, v_2^q)) \notin E_{L_r}$ (4.7).

D'après (1) $f_q((v_1^q, v_2^q)) \in V^2_{R_q} \cap E_{R_q}$. D'après (4.6), $\exists (v_1^p, v_2^p) \in V^2_{L_p}, f((v_1^p, v_2^p)) = f_q((v_1^q, v_2^q)$. La propriété (4.3) donne pour chacun de ces (v_1^p, v_2^p), $(v_1^p, v_2^p) \in V^2_{L_p^I}$. $f_q((v_1, v_2))$ est donc l'image par f de couple de nœuds de la partie invariante de r_R, $L_p^{C_q} \cap L_p^I$, uniquement. Notons (v_1^p, v_2^p) l'un de ses couples. Par définition, $f_p((v_1^p, v_2^p)) \in V^2_{R_r}$, l'unicité de $f_p((v_1^p, v_2^p))$ étant garantie par 3. (b). Selon

(4.4) tout arc entre (v_1^p, v_2^p) est invariant, et comme $f_q((v_1^q, v_2^q)) \in E_{R_q}$, $f_p((v_1^p, v_2^p)) \in E_{r_{R\_}(f,I)(R_q)} = E_{R_r}$. Par définition de f_r, $f_r((v_1^q, v_2^q)) = f_p((v_1^p, v_2^p))$, donc $f_r((v_1^q, v_2^q)) \in E_{R_r}$ (4.8).

D'après (4.2), la relation (2) est vrai en particulier pour $(v_1^r, v_2^r) = f_q((v_1^q, v_2^q))$ et donc, avec (4.7) et (4.8), $f_r^L(f_q((v_1^q, v_2^q))) \notin E_G$. Or, d'après (1) $f_q^L((v_1, v_2)) \in E_G$, par définition $f_q^L = f_r^L \circ f_q^+$, et d'après (4.2) $f_q^+((v_1, v_2)) = f_q((v_1, v_2))$. On obtient donc une contradiction et m_q^L remplit la seconde condition d'applicabilité pour ce type d'arc.

Si une extrémité est dans $\{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ **et l'autre non**, la démonstration étant similaire dans les deux cas, supposons que $v_1^q \in \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ et $v_2^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ (5.1). D'après la définition de f_q^+ on a alors $f_q^+((v_1^q, v_2^q)) = (f_q(v_1^q), v_2^q)$ (5.2). Comme précédemment (voir (3.2) et (4.5)), on a $f_q^1(v) \in V_{L_r^I}$ et $v_2^q \in V_{L_r^I}$ (5.3).

Par construction de L_r et par définition de l'expansion, $\forall v_r \in V_{L_q} \cap V_{L_r}$, $\forall f_q(v_q) \in V_{L_r}$, $(v_r, v_q) \notin E_{L_q} \Leftrightarrow (f_q(v_q), v_r) \notin E_{L_r}$. (1) donne donc $(f_q(v_1^q), v_2^q) \notin E_{L_r}$ (5.4).

D'après (1) $f_q((v_1^q, v_2^q)) \in V_{R_q}^2 \cap E_{R_q}$. Le raisonnement aboutissant à (3.4) tient toujours pour v_2^q, et $f_q(v_2^q) \notin Dom(f)$ et $f_q(v_2^q) \in V_{R_r}$. De même, la démonstration de (4.8) tient partiellement et en a $v_p \in V_{L_p^{C_q^I}} \cap V_{L_p^I}$, $f(v_p) = f_q(v_1^q)$. Par définition, $f_p(v_p) \in V_{R_r}$, l'unicité de $f_p(v_p)$ étant garantie par 3. (b).

Par définition des règles de réécriture, un arc ne peut être supprimé ou modifié lors de l'application d'une règle si une seule de ses extrémités fait partie de l'image du morphisme selon laquelle la règle est appliquée et si celle-ci n'est pas supprimé lors de l'application. Par conséquent, $(f_p(v_p), f_q(v_2^q) \in E_{R_r}$. Par définition de f_r, $f_r((v_1^q, v_2^q)) = (f_p(v_p), f_q(v_2^q))$, donc $f_r((v_1^q, v_2^q)) \in V_{E_r}$ (5.5).

D'après (5.3), la relation (2) est vrai en particulier pour $(v_1^r, v_2^r) = (f_q(v_1^q), v_2^q)$ et donc, avec (5.4) et (5.5), $f_r^L(f_q(v_1^q), v_2^q)) \notin E_G$. Or, $f_q^L((v_1^q, v_2^q)) \in E_G$ d'après (1), par définition $f_q^L = f_r^L \circ f_q^+$, et d'après (5.2) $f_q^+((v_1^q, v_2^q)) = (f_q(v_1^q), v_2^q)$. On obtient donc une contradiction et m_q^L remplit la seconde condition d'applicabilité pour ce type d'arc.

Ainsi, m_q^L remplit la seconde condition d'applicabilité.

Satisfaction de la troisième condition d'applicabilité. Supposons que m_q^L ne remplisse pas la troisième condition d'applicabilité, donc $\exists (v_1^q, v_2^q) \in Dom(f_r^L \circ f_q^+)^2, f_r^L(f_q^+(v_1^q)) = f_r^L(f_q^+(v_2^q)) \wedge ((v_1^q \in Dom(f_q) \wedge v_2^q \notin Dom(f_q)) \vee ((v_1^q \notin Dom(f_q) \wedge v_2^q \in Dom(f_q)) \vee ((v_1^q, v_2^q) \in Dom(f_q)^2 \wedge f_q(v_1^q) \neq f_q(v_2^q))$ (6).

Par hypothèse, m_r^L remplit cette troisième condition et : $\forall (v_1^r, v_2^r) \in Dom(f_r^L)^2, f_r^L(v_1^r) = f_r^L(v_2^r) \implies (v_1^r \notin Dom(f_r) \wedge v_2^r \notin Dom(f_r)) \vee ((v_1^r, v_2^r) \in Dom(f_r)^2 \wedge f_r(v_1^r) = f_r(v_2^r)))$ (7).

Supposons dans un premier temps que $(v_1^q \in Dom(f_q) \wedge v_q^2 \notin Dom(f_q)) \vee ((v_1^q \notin Dom(f_q) \wedge v_2^q \in Dom(f_q))$. La démonstration étant similaire dans les deux cas, supposons $(v_1^q \in Dom(f_q) \wedge v_2^q \notin Dom(f_q))$. Par conséquent, $f_q^+(v_2^q) = v_2^q$. Par construction de L_r^I, tout nœud appartenant à $V_{L_q} \cap V_{L_r}$ et pas à $V_{L_q^I}$ (= $Dom(f_q)$) n'appartient pas à $V_{L_r^I}$ (= $Dom(f_r)$). Donc $v_2^q \notin Dom(f_r)$. La relation (7) étant en particulier vraie pour $(f_q^+(v_1^q), f_q^+(v_2^q)) = (f_q^+(v_1^q), v_2^q)$ on obtient, $((f_q^+(v_1^q) \notin Dom(f_r) \wedge v_2^q \notin Dom(f_r)) \vee ((f_q^+(v_1^q), v_2^q) \in Dom(f_r)^2 \wedge f_r((f_q^+(v_1^q)) = f_r(v_2^q)))$. Comme $v_2^q \notin Dom(f_r)$, $f_q^+(v_1^q) \notin Dom(f_r)$.

Si $v_1^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$, $f_q^+(v_1^q) = v_1^q$ et $v_1^q \in Dom(f_q) \implies v_1^q \in Dom(f_r)$.
Sinon, $f_q^+(v_1^q) = f_q(v_1^q)$ et $f_q(v_1^q) \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$. La pré-condition 5 entraîne donc la validité de

$P_1(f_q(v_1^q))$ et $f_q(v_1^q) \in Dom(f_r)$. On obtient donc une contradiction, et la première clause de (6) ne peut être vraie.

Supposons donc que $(v_1^q, v_2^q) \in Dom(f_q)^2 \wedge f_q(v_1^q) \neq f_q(v_2^q)$.(8) Par disjonction des cas,

— Supposons que $v_1^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ et $v_2^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$. D'après la définition de f_q^+ on a alors $f_q^+((v_1^q, v_2^q)) = (v_1^q, v_2^q)$ et donc $(v_1^q, v_2^q) \in V^2_{L_r}$. Par construction de $V_{L_r^I}$, $\forall v \in V_{L_r} \cap V_{L_q}, v \in V_{L_q^I} \implies v \in V_{L_r^I}$. D'où $(v_1^q, v_2^q) \in V^2_{L_r^I}$ et $(v_1^q, v_2^q) \in Dom(f_r)^2$. D'après (6), $f_r^L(f_q^+(v_1^q)) = f_r^L(f_q^+(v_2^q))$ et donc $f_r^L(v_1^q) = f_r^L(v_2^q)$. La relation (7) donne donc pour v_1^q et v_2^q : $f_r(v_1^q) = f_r(v_2^q)$. Or, f_r coïncide avec f_q sur $V_{L_q^I}$, donc $f_r(v_1^q) = f_q(v_1^q)$ et $f_r(v_2^q) = f_q(v_2^q)$, et selon (8) $f_q(v_1^q) \neq f_q(v_2^q)$. On obtient donc une contradiction et m_q^L remplit donc la troisième condition d'applicabilité pour ce type de nœuds.

— Supposons que $(v_1^q, v_2^q) \in \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}^2$. D'après la définition de f_q^+ on a alors $f_q^+((v_1^q, v_2^q)) = f_q((v_1^q, v_2^q))$ et, par construction, $f_q((v_1^q, v_2^q)) \in V^2_{L_p^{C_q^I} \downarrow_C R_q^I}$. D'après la condition 5, $f_q(v_1^q)$ et $f_q(v_2^q)$ satisfont P_1, et $(f_q(v_1^q), f_q(v_2^q)) \in Dom(f_r)^2$. D'après (6), $f_r^L(f_q(v_1^q)) = f_r^L(f_q(v_2^q))$. La relation (7) est vraie en particulier pour $f_q(v_1^q)$ et $f_q(v_2^q)$, et donc $f_r^L(f_q(v_1^v)) = f_r^L(f_q(v_2^q)) \implies f_r(f_q(v_1^q)) = f_r(f_q(v_2^q))$. Par définition de f_r, $\forall i \in [1,2]$, $f_r(f_q(v_i^q)) = f_p(v_i^p)$ avec $v_i^p \in V_{L_p^{C_q^I}}$ tel que $f(v_i^p) = f_q(v_i^q)$ et donc $f_p(v_1^p) = f_p(v_2^p)$. D'après la pré-condition 6. on obtient donc $f(v_1^p) = f(v_2^p)$ et $f_q(v_1^q) = f_q(v_2^q)$.Ceci contredit la (8) et m_q^L remplit donc la troisième condition d'applicabilité pour ce type de nœuds.

— Finalement, supposons que $v_1^q \notin \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$ et $v_2^q \in \{v \in Dom(f_q) | f_q(v) \in V_{L_p^S}\}$. On a donc $f_q^+(v_1^q) = v_1^q, f_q^+(v_2^q) = f_q(v_2^q)$, et comme précédemment $(v_1^q, f_q(v_2^q)) \in V^2_{L_r^I}$. D'après (6), $f_r^L(f_q^+(v_1^q)) = f_r^L(f_q^+(v_2^q))$ et donc $f_r^L(v_1^q) = f_r^L(f_q(v_2^q))$. La relation (7) donne donc pour v_1^q et v_2^q : $f_r(v_1^q) = f_r(v_2^q)$. Par définition de f_r, $f_r(v_1^q) = f_q(v_1^q)$ et $f_r(v_2^q) = f_p(v_p)$ avec $f(v_p) = v_2^q$. Or, $Im(f_p)$ et $Im(f_q)$ sont disjoints, c'est donc impossible. m_q^L remplit donc la troisième condition d'applicabilité pour ce type de nœuds.

Ainsi, m_q^L remplit la troisième et dernière condition d'applicabilité pour q et G.

B.2.2 Il existe un morphisme m_p^L selon lequel p est applicable à $q\_m_q^L(G)$

Existence d'un morphisme cohérent. Par hypothèse, $L_r \xrightarrow{m_r} G$, donc $L_p^S \to G$, et L_p et R_q sont C-compatibles, et par définition $R_q \xrightarrow{(Id, I \cup I_r^L)} q\_m_q^L(G)$.

Soit $m_p^L = (f_p^L, I \cup I_r^L)$ avec f_p^L la fonction coïncidant avec f sur $Dom(f)$ et avec f_r^L sur $V_{L_r} \cap V_{L_p}$.

Par construction, V_{L_r} contient exactement les nœuds de V_{L_p} n'étant pas dans $Dom(f)$ et il est immédiat que $Dom(f) \cup (V_{L_r} \cap V_{L_p}) = V_{L_p}$. Comme L_p et R_q sont C-compatibles et que $R_q \xrightarrow{(Id, I \cup I_r^L)} q\_m_q^L(G)$, trivialement $Dom(f) \subseteq V_{q\_m_q^L(G)}$. Par construction de L_r et m_q, les nœuds $f_r((V_L \cap V_p))$ sont ceux n'étant pas matchés par m_q et donc invariants pour l'application de q selon m_q^L, de sorte que $f_r((V_L \cap V_p)) \subseteq V_{q\_m_q^L(G)}$.

Montrons que m_p^L est bien un homomorphisme de L_p vers $q\_m_q^L(G)$, c'est à dire qu'il préserve les arcs : $\forall (v_1^p, v_2^p) \in E_{L_p}, f_p^L(v_1^p, v_2^p) \in E_{q\_m_q^L(G)}$.

Si $(v_1^p, v_2^p) \in (V_{L_r} \cap V_{L_p})^2$, $f_r^L(v_1^p, v_2^p) \in E_G$ comme $L_r \xrightarrow{m_r} G$. Comme il est invariant pour l'application de q selon m_q^L (car non matché par m_q^L), $f_r^L(v_1^p, v_2^p) \in E_{q\_m_q^L(G)}$.

Si une extrémité est dans $Dom(f)$ et l'autre non, la pré-condition 1. impose que cette extrémité est invariante pour l'application de q selon m_q^L. L'arc est dans G comme $L_r \xrightarrow{m_r} G$ et donc dans $q\_m_q^L(G)$ comme une extrémité n'est pas matché par m_q^L et l'autre est invariante pour l'application de q.

Si les deux extrémités sont dans $Dom(f)$,

- si au moins une est ajoutée par l'application de q selon m_q^L, la pré-condition 2. (a) impose que l'arc est également ajouté lors de la transformation, et donc présent dans $q\_m_q^L(G)$.
- si les deux sont invariantes pour l'application de q, la pré-condition 2. (b) impose que l'arc est invariant pour l'application de q et donc présent dans $q\_m_q^L(G)$ comme présent dans G.

m_p^L est donc un homomorphisme de L_p vers $q\_m_q^L(G)$ dont la cohérence est garantie par la cohérence de m_r^L.

Satisfaction de la seconde condition d'applicabilité. Supposons que m_p^L ne remplisse pas la seconde condition d'applicabilité, soit :
$\exists(v_1, v_2) \in V^2_{L_p^I}, (v_1, v_2) \notin E_{L_p} \wedge f_p((v_1, v_2)) \in E_{R_p} \wedge f_p^L((v_1, v_2)) \in E_{q\_m_q^L(G)}$ (1).

Par hypothèse, m_r^L remplit cette seconde condition et : $\forall(v_1^r, v_2^r) \in V^2_{L_r^I}, (v_1^r, v_2^r) \notin E_{L_r} \wedge f_r((v_1^r, v_2^r)) \in E_{R_r}) \implies f_r^L((v_1^r, v_2^r)) \notin E_G$. (2)

Par disjonction de cas :

Si $(v_1, v_2) \in Dom(f)^2$, la pré-condition 3.a donne $f(v_1, v_2) \notin E_{R_q}$. Il est immédiat que $(f(v_1) \notin Im(f_q) \vee v_2 \notin Im(f_q)) \implies f_p^L((v_1, v_2))(= f((v_1, v_2))) \notin E_{q\_m_q^L(G)}$: si une règle ajoute un nœud, un arc ne peut l'avoir pour extrémité après son application si l'arc n'apparaît pas dans sa partie droite.

Si $f((v_1, v_2)) \in Im(f_q)^2 = V^2_{R_q^I}$ (3.1), i.e. s'ils sont invariants pour q, comme $f(v_1, v_2) \notin E_{R_q} \supseteq E_{R_q^I}$ l'arc est soit supprimé par q, auquel cas il ne peut être dans $q\_m_q^L(G)$, soit invariant pour q, i.e. sans antécédents pour f_q dans E_{L_q}. Avec $(v_1, v_2) \notin E_{L_p}$ ceci donne $\exists(v_1^r, v_2^r) = f((v_1, v_2)) \in V^2_{L_r}, (v_1^r, v_2^r) \notin E_{L_r}$ (3.2). Comme l'arc $f(v_1, v_2)$ est invariant pour q est présent dans le résultat de son application, il était présent dans G et $f_r^L((v_1^r, v_2^r)) \in E_G$.(3.3)

Par construction de Inv_V, (3.1), $(v_1, v_2) \in V^2_{L_p^I}$ et la précondition 3.(b) impliquent la validité de $P_1(f(v_1))$ et $P_1(f(v_2))$, d'où $f(v_1, v_2) = (v_1^r, v_2^r) \in V^2_{L_r^I}$.(3.4)

Par construction de R_r, $f_p((v_1, v_2)) \in E_{R_p} \implies f_p((v_1, v_2)) \in E_{R_r}$. Par définition de f_r, $f_r((v_1^r, v_2^r)) = f_p(v_1, v_2)$ (3.5). Les relations (3.2) à (3.5) contredisent la relation (2), m_p^L remplit donc la seconde condition d'applicabilité dans ce cas.

Si $(v_1, v_2) \in (V_{L_r} \cap V_{L_p})^2$**,** $f_p^L((v_1, v_2)) = f_r^L((v_1, v_2))$. Par construction de L_r, $(v_1, v_2) \notin E_{L_p} \implies (v_1, v_2) \notin E_{L_r}$. Par construction de Inv_V, comme $(v_1, v_2) \in V^2_{L_p^I} \cap V_{L_r}$, $(v_1, v_2) \in V^2_{L_r^I}$. D'après (1), $f_p(v_1, v_2) \in E_{R_p}$ et donc, par construction de R_r et de f_r, $f_p(v_1, v_2) = f_r(v_1, v_2) \in E_{R_r}$. D'après (1), $f_p^L((v_1, v_2)) = f_r^L((v_1, v_2)) \in E_{q\_m_q^L(G)}$. Comme $f_r^L((v_1, v_2))$ n'est pas matché par m_q^L, il est invariant pour l'application de q et donc $f_r^L((v_1, v_2)) \in E_G$. Ceci contredit (2), et m_p^L remplit donc la seconde condition d'applicabilité dans ce cas.

Si $(v_1 \in Dom(f) \wedge v_2 \in V_{L_r} \cap V_{L_p}) \vee (v_2 \in Dom(f) \wedge v_1 \in V_{L_r} \cap V_{L_p}))$, les démonstrations étant similaires dans les deux cas, supposons $v_1 \in Dom(f) \wedge v_2 \in V_{L_r} \cap V_{L_p}$.

Par définition de f_p^L, $f_p^L((v_1,v_2)) = (f(v_1), f_r^L(v_2))$ et seul $f(v_1)$ est touché par l'application de q selon m_q^L. Par construction de L_r, comme $(v_1,v_2) \notin E_{L_p}$, $(f(v_1),v_2) \notin E_{L_r}$. Comme $f_r^L((f(v_1),v_2))$ n'est pas matché par m_q^L, il est invariant pour l'application de q et la présence de $f_p^L((v_1,v_2))$ dans $E_{q\_m_q^L(G)}$ implique que $f_r^L((f(v_1),v_2)) \in E_G$. (4.1)

D'après (1), $(f(v_1), f_r^L(v_2)) \in E_{q\_m_q^L(G)}$, comme $f_r^L(v_2)$ n'est pas matché par m_q^L, $f(v_1)$ ne peut être ajouté par q, i.e., $f(v_1) \in V_{R_q^I}$. Par conséquent et par construction de L_r, $(f(v_1),v_2) \in E_{L_r}$. (4.2)

Par construction de Inv_V, comme $f(v_1) \in V_{R_q^I}$, comme d'après (1) $v_1 \in V_{L_p^I}$ et d'après la pré-condition 3.(b), $P_1(f(v_1))$ et $f(v_1) \in V_{L_r^I}$. Comme $v_2 \in V_{L_r} \cap V_{L_p^I}, v_2 \in V_{L_r^I}$. (4.3)

Par construction de R_r et f_r, $f_p((v_1,v_2)) \in E_{R_p} \implies f_p((v_1,v_2)) = f_r(f(v_1),v_2) \in E_{R_r}$. (4.4)

Les relations (4.1) à (4.4) contredisent (2), et m_p^L remplit donc la seconde condition d'applicabilité dans ce cas.

Satisfaction de la troisième condition d'applicabilité. Supposons que m_p^L ne remplisse pas la troisième condition d'applicabilité, donc $\exists (v_1,v_2) \in Dom(f_p)^2, f_p^L((v_1)) = f_p^L(v_2)) \wedge ((v_1 \in Dom(f_p) \wedge v_2 \notin Dom(f_p)) \vee ((v_1 \notin Dom(f_p) \wedge v_2 \in Dom(f_p)) \vee ((v_1,v_2) \in Dom(f_p)^2 \wedge f_p(v_1) \neq f_p(v_2))$ (6).

Par hypothèse, m_r^L remplit cette troisième condition pour G et r d'où :
$\forall (v_1^r, v_2^r) \in Dom(f_r^L)^2, f_r^L(v_1^r) = f_r^L(v_2^r) \implies (v_1^r \notin Dom(f_r) \wedge v_2^r \notin Dom(f_r)) \vee ((v_1^r, v_2^r) \in Dom(f_r)^2 \wedge f_r(v_1^r) = f_r(v_2^r)))$ (7).

Par définition, f_p^L coïncide avec f sur $Dom(f)$ et avec f_r^L sur $V_{L_r} \cap V_{L_p}$.

Par disjonction de cas :

Si $(v_1,v_2) \in Dom(f)^2$, $f_p^L(v_i) = f(v_i)$ et (6) donne $f(v_1) = f(v_2)$. D'après la pré-condition 3.(b), $(v_1 \notin Dom(f_p) \wedge v_2 \notin Dom(f_p)) \vee ((v_1,v_2) \in Dom(f_p)^2 \wedge f_p(v_1) = f_p(v_2))$, ce qui contredit (6).

m_p^L remplit donc la troisième condition d'applicabilité dans ce cas.

Si $(v_1 \in Dom(f) \wedge v_2 \in V_{L_r} \cap V_{L_p}) \vee (v_2 \in Dom(f) \wedge v_1 \in V_{L_r} \cap V_{L_p}))$, les démonstrations étant similaires dans les deux cas, supposons $v_1 \in Dom(f) \wedge v_2 \in V_{L_r} \cap V_{L_p}$.

Par définition, $f_p^L(v_1) = f(v_1)$ et avec $f_p^L(v_2) = f_r^L(v_2)$.

La relation (6) donne en particulier $(f(v_1) = f_r^L(v_2))$. Or $f_r^L(v_2))$ est présent dans G comme r est applicable à G selon m_r, donc $f(v_1)$ est présent dans G. Comme $f_(v_1) \in V_{R_q}$ et comme les nœuds de graphes de deux règles sont disjoints, $f_(v_1)$ ne peut être présent avant l'application de q.

m_p^L remplit donc la troisième condition d'applicabilité dans ce cas.

Si $(v_1,v_2) \in (V_{L_r} \cap V_{L_p})^2$, par définition $f_p^L(v_i) = f_r^L(v_i)$.

Par construction de Inv_V, $\forall v_i \in V_{L_r} \cap V_{L_p}, v_i \in Dom(f_p) \Leftrightarrow v_i \in Dom(f_r)$. Par conséquent, (7) entraîne $(v_1 \notin Dom(f_p) \wedge v_2 \notin Dom(f_p)) \vee (v_1,v_2) \in Dom(f_p)^2$ (8).

Par construction de f_r, $\forall v_i \in V_{L_r^I} \cap V_{L_p^I}, f_r(v_i) = f_p(v_i)$. Donc, $(v_1,v_2) \in Dom(f_r)^2 \wedge f_r(v_1) = f_r(v_2) \implies (v_1,v_2) \in Dom(f_p)^2 \wedge f_p(v_1) = f_p(v_2)$. Ceci, 7 et 8 donnent donc $(v_1 \notin Dom(f_p) \wedge v_2 \notin Dom(f_p)) \vee ((v_1,v_2) \in Dom(f_p)^2 \wedge f_p(v_1) = f_p(v_2))$.

m_p^L remplit donc la troisième condition d'applicabilité dans ce dernier cas.

B.2.3 Égalité, équivalence des applications

B.2.3.1 Équivalence des suppressions de nœuds

Tout nœud supprimé par q est supprimé par r. Un nœud v est supprimé par q si $\exists v_q \in V_{L_q}, v_q \notin V_{L_{q^I}}, m_L^q v_q = v$.

Montrons que v est également supprimé par r, i.e. $\exists v_r \in V_{L_r} \backslash V_{L_{r^I}}, m_L^r(v_q) = v$.

Par définition de f_q^+, $f_q^+(v_q) \in V_{L_r}$ et

- si $v_q \notin \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}$, $f_q^+(v_q) = v_q$. Alors, par construction de L_{r^I}, $v_q \in V_{L_r^I} \Leftrightarrow v_q \in V_{L_q^I}$. Comme $v_q \in V_{L_q} \backslash V_{L_{q^I}}, v_q \notin V_{L_r^I}$. L'application de r selon m_L^r supprime donc $m_L^r(v_q)$. Par définition de m_L^q, il coïncide avec m_L^r sur $V_{L_r} \cap V_{L_q}$ et $m_L^r(v_q) = m_L^q(v_q)$.
- si $v_q \in \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}$, $f_q^+(v_q) = f_q(v_q)$. Par définition de $V_{R_q^I}, v_q \notin V_{L_q^I} \implies v_q \notin V_{R_q^I}$. $f_q(v_q)$ ne satisfait donc pas P_1 et $f_q(v_q) \notin V_{L_r^I}$. L'application de r selon m_L^r supprime donc $m_L^r(f_q(v_q))$. Par définition de m_L^q, $f_q^L = f_r^L \circ f_q^+$ et $f_q^L(v_q) = f_L^r(f_q(v_q))$.

La règle r appliquée selon m_r^L supprime donc également v.

Tout nœud supprimé par l'application de q puis p est supprimé par r Un nœud v de G est supprimé par la séquence p, q est supprimé soit par q soit il est supprimé par p mais pas ajouté par q.

Supposons donc qu'un nœud v de $q\_m_q^L(G)$ est supprimé par p, i.e. $\exists v_p \in V_{L_p} \backslash V_{L_{p^I}}, m_L^p v_p = v$.

Montrons que soit v est également supprimé par r, soit v est invariant pour q et son antécédent par f_q est supprimé par r, soit v est ajouté par q i.e. :
$(\exists v_r \in V_{L_r}, v_r \notin V_{L_{r^I}} \wedge (m_L^r(v_r) = v \vee (\exists v_q \in V_{L_q^I}, f_q(v_q) = f(v_p) \wedge m_L^r(v_r) = m_L^q(v_q)))) \vee (v \in V_{R_q} \wedge v \notin Im(f_q))$

Par définition, f_p^L coïncide avec f sur $Dom(f)$ et avec f_r^L sur $V_{L_r} \cap V_{L_p}$, et :

- si $v_p \in V_{L_r} \cap V_{L_p}$: $v_p \in V_{L_r}$ et alors, par construction de $V_{L_r^I}, v_p \in V_{L_r^I} \Leftrightarrow v_p \in V_{L_p^I}$. Comme $v_p \notin V_{L_p^I}, v_p \notin V_{L_r^I}$. L'application de r selon m_L^r supprime donc $m_L^r(v_p)$. Par définition de f_L^p, il coïncide avec f_L^r sur $V_{L_r} \cap V_{L_p}$ et $m_L^r(v_p) = m_L^p(v_p)$.
- si $v_p \in Dom(f)$: par construction de L_r, :
  - si $f(v_p) \in V_{L_p^{C_q^I}}$, $f(v_p) \in V_{L_r}$ et comme $f(v_p) \wedge v_p \notin (V_{L_p^I} \cup E_{L_p^I})$, $f(v_p)$ ne satisfait pas P_1, donc $f(v_p) \notin V_{L_r^I}$. L'application de r selon m_L^r supprime donc $m_L^r(f(v_p))$. Par définition de m_L^q, $f_q^L = f_r^L \circ f_q^+$ et $f_q^L(v_q) = f_L^r(f_q(v_q))$ avec $v_q \in V_{L_{R_q^I}}$ tel que $f_q(v_q) = f(v_p)$.
  - si $f(v_p) \notin V_{L_p^{C_q^I}}$, $f(v_p) \in V_{L_r}$. Par définition de $L_p^{C_q^I}$, $f(v_p) \in V_{R_q} \wedge f(v_p) \notin Im(f_q)$. $f(v_p)$ est donc ajouté par q, puis supprimé par p.

Tout nœud supprimé par r est supprimé par l'application de q puis p. Un nœud v est supprimé par r si $\exists v_r \in V_{L_r}, v_r \notin V_{L_{r^I}}, m_L^r v_r = v$.

Par disjonction de cas :

— si $v_r \in V_{L_q} \cup V_{L_p}$, il est immédiat que $m_L^r v_r$ est également supprimé par q ou p. En effet, par construction de $V_{L_r^I}, \forall v_r \in V_{L_r}, (v_r \in V_{L_q} \implies ((v_r) \in V_{L_r^I} \Leftrightarrow v_r \in V_{L_q^I})) \wedge (v_r \in V_{L_p} \implies ((v_r) \in V_{L_r^I} \Leftrightarrow v_r \in V_{L_p^I}))$. De plus m_L^r coïncide avec m_L^p sur $V_{L_p} \cap V_{L_p}$ et avec m_L^q sur $V_{L_q} \cap V_{L_q}$.

— sinon, $v_r \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$. Par définition de R_q^I ceci implique que $\exists v_q \in V_{L_q^I}, f_q(v_q) = v_r$. Par construction de $V_{L_r^I}$, $v_r \notin V_{L_r^I}$ implique alors $\not P_1(v_r)$, soit $\exists v_p \in V_{L_p}, f(v_p) = v_r \wedge v_p \notin V_{L_p^I}$. $m_L^p(v_p)$ est donc supprimé par p, et par définition de m_L^p, $m_L^p(v_p) = m_L^r(v_r)$.

On obtient donc l'équivalence des suppressions de nœuds.

B.2.3.2 Équivalence des suppressions d'arcs

L'équivalence des suppressions de nœuds implique que tout arc implicitement supprimé d'un côté est supprimé de l'autre. Intéressons nous donc aux suppressions explicites.

Tout arc explicitement supprimé par q est supprimé par r. Un arc $e = (v^1, v^2)$ de G est explicitement supprimé par q si $\exists (v_q^1, v_q^2) \in V^2_{L_q^I} \cap E_{L_q}, (v_q^1, v_q^2) \notin E_{L_q^I} \wedge m_L^q((v_q^1, v_q^2)) = e$.

Montrons que v est également supprimé par r, i.e. $\exists e_r \in E_{L_r}, e_r \notin E_{L_{r^I}}, m_L^r(e_r) = e$.

Par définition de f_q^+, $f_q^+((v_q^1, v_q^2)) \in V^2_{L_r}$ et

— si $v_q^1 \notin \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\} \wedge v_q^2 \notin \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}$, $f_q^+((v_q^1, v_q^2)) = (v_q^1, v_q^2)$. Alors, par construction de L_r, $(v_q^1, v_q^2) \in E_{L_r}$. Par construction de L_{r^I}, $(v_q^1, v_q^2) \in V^2_{L_r^I} \Leftrightarrow (v_q^1, v_q^2) \in V^2_{L_q^I}$. D'où $(v_q^1, v_q^2) \in V^2_{L_r^I}$. De même, par construction de E_{r^I} on a alors, $(v_q^1, v_q^2) \in V^2_{L_r^I} \implies ((v_q^1, v_q^2) \in E_{L_r^I} \Leftrightarrow (v_q^1, v_q^2) \in E_{L_q^I})$. Par conséquent, $(v_q^1, v_q^2) \notin E_{L_r^I}$. L'application de r selon m_L^r supprime donc $m_L^r((v_q^1, v_q^2))$. Par définition de m_L^q, il coïncide avec m_L^r sur $V_{L_r} \cap V_{L_q}$ et $m_L^r((v_q^1, v_q^2)) = m_L^q((v_q^1, v_q^2))$.

— si $(v_q^1, v_q^2) \in \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}^2$, alors $f_q^+((v_q^1, v_q^2)) = f_q((v_q^1, v_q^2))$. Par construction de L_r, $f_q(v_q^1, v_q^2) \in E_{L_r}$. Par définition de m_L^q, $f_q^L = f_r^L \circ f_q^+$ et $f_q^L(((v_q^1, v_q^2))) = f_L^r(f_q((v_q^1, v_q^2)))$. Donc, il s'agit de montrer que r entraîne la suppression de $f_q((v_q^1, v_q^2))$. Si $f_q((v_q^1, v_q^2)) \notin V^2_{L_r^I}$, la suppression est immédiate. Supposons donc $f_q((v_q^1, v_q^2)) \in V^2_{L_r^I}$. Comme $(v_q^1, v_q^2) \notin E_{L_q^I}, (v_q^1, v_q^2) \notin R_q^I$, par construction de $E_{L_r^I}$ $f_q((v_q^1, v_q^2)) \notin E_{L_r^I}$.

— si $((v_q^1) \in \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\} \wedge (v_q^2) \notin \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}) \vee ((v_q^1) \notin \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\} \wedge (v_q^2) \in \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\})$, les démonstrations étant similaires dans les deux cas, supposons $(v_q^1) \in \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\} \wedge (v_q^2) \notin \{v \in V_{L_q} | f_q(v) \in V_{L_p^S}\}$. Alors $f_q^+((v_q^1, v_q^2)) = (f_q(v_q^1), v_q^2)$ et par définition de m_L^q, $f_q^L = f_r^L \circ f_q^+$ et $f_q^L(((v_q^1, v_q^2))) = f_L^r((f_q(v_q^1), v_q^2))$. Donc, il s'agit de montrer que r entraîne la suppression de $(f_q(v_q^1), v_q^2)$. Si $(f_q(v_q^1), v_q^2)) \notin V^2_{L_r^I}$, la suppression est immédiate. Supposons donc $(f_q(v_q^1), v_q^2) \in V^2_{L_r^I}$. Ici, $f_q(v_q^1) \in V_{R_q^I}$ et $v_q^2 \in V_{L_q^I}$. Or $(v_q^1, v_q^2) \in E_{L_q} \wedge (v_q^1, v_q^2) \notin E_{L_q^I}$. Par construction de $E_{L_r^I}$, ceci implique que $(f_q(v_q^1), v_q^2) \in E_{L_r^I}$.

Tout arc explicitement supprimé par r **est supprimé par l'application de** q **puis** p**.** Un nœud v de G est explicitement supprimé par la séquence p, q s'il est explicitement supprimé soit par q ou qu'il est supprimé par p mais pas ajouté par q, ses extrémités restants invariantes.

Supposons donc qu'un arc (v_1, v_2) de $q\_m^L_q(G)$ est explicitement supprimé par p, i.e. $\exists (v_p^1, v_p^2) \in V^2_{L^I_p} \cap E_{L_p}, (v_p^1, v_p^2) \notin E_{L_{pI}} \wedge m^p_L(v_p^1, v_p^2) = (v_1, v_2)$.

Montrons que soit (v_1, v_2) est également supprimé par r, soit (v_1, v_2) est invariant pour q et son antécédent par f_q est supprimé par r, soit (v_1, v_2) est ajouté par q i.e. :
$(\exists (v_r^1, v_r^2) \in E_{L_r} \backslash E_{L^I_r}$,
$(m^r_L((v_r^1, v_r^2)) = (v_1, v_2)) \vee$
$(\exists (v_q^1, v_q^2) \in V^2_{L^I_q}, f_q((v_q^1, v_q^2)) = f((v_p^1, v_p^2)) \wedge m^r_L((v_r^1, v_r^2)) = m^q_L((v_q^1, v_q^2))) \vee$
$(\exists v_q \in V_{L^I_q}, f_q(v_q) = f(v_p^1) \wedge m^r_L((v_r^1)) = m^q_L((v_q^1)) \wedge m^r_L((v_r^2)) = v_2) \vee$
$(\exists v_q \in V_{L^I_q}, f_q(v_q) = f(v_p^2) \wedge m^r_L((v_r^2)) = m^q_L((v_q^2)) \wedge m^r_L((v_r^1)) = v_1)))$
$\vee (e \in E_{R_q} \wedge e \notin E_{R^I_q}))$

Par définition, f^L_p coïncide avec f sur $Dom(f)$ et avec f^L_r sur $V_{L_r} \cap V_{L_p}$, et :

- Si $(v_p^1, v_p^2) \in (V_{L_r} \cap V_{L_p})^2$, alors $f^L_p((v_p^1, v_p^2)) = f^L_r((v_p^1, v_p^2))$. Comme $(v_p^1, v_p^2) \in (V_{L_r} \cap V_{L_p})^2$, on a alors :
  par construction de $V_{L^I_r}$, $(v_p^1, v_p^2) \in V^2_{L^I_r} \Leftrightarrow (v_p^1, v_p^2) \in V^2_{L^I_p}$.
  par construction de L_r, $(v_p^1, v_p^2) \in E_{L_r} \Leftrightarrow (v_p^1, v_p^2) \in E_{L_p}$ et
  par construction de $E_{L^I_r}$, $(v_p^1, v_p^2) \notin E_{L^I_r} \Leftrightarrow (v_p^1, v_p^2) \notin E_{L^I_p}$. Par conséquent, l'arc $f^L_p((v_p^1, v_p^2)) = f^L_r((v_p^1, v_p^2))$ est bien supprimé par r.
- Si $(v_p^1, v_p^2) \in Dom(f)^2$, alors $f^L_p((v_p^1, v_p^2)) = f((v_p^1, v_p^2))$. Si $f((v_p^1, v_p^2)) \notin V^2_{R^I_q} \vee f((v_p^1, v_p^2)) \in E_{R^I_q}$, l'arc est nécessairement ajouté par q.
  Supposons donc que $f((v_p^1, v_p^2)) \in V^2_{R^I_q} \wedge f((v_p^1, v_p^2)) \in E_{R^I_q}$. Par définition de R^I_q, il existe par conséquent $(v_1^q, v_2^q) \in V^2_{L^I_q}, (v_1^q, v_2^q) \notin E_{L_q} \wedge f_q((v_1^q, v_2^q)) = f((v_1^p, v_2^p))$. Ainsi, $f((v_1^p, v_2^p) \in V^2_{L_p^{C^I_q} \downarrow_C R^I_q}$ et, par construction de L_r, $f((v_1^p, v_2^p)) \in V^2_{L_r}$. Comme $(v_1^p, v_2^p) \in E_{L_p}, f((v_1^p, v_2^p)) \in E_{L_r}$.
  De plus, $(v_q^1, v_q^2) \in V^2_{L^I_q}, f_q((v_q^1, v_q^2)) = f((v_p^1, v_p^2))$ et par définition de m^q_L, $m^r_L(f_q(v_q^1, v_q^2)) = m^q_L((v_q^1, v_q^2))$. Ainsi, il s'agit de prouver que r supprime bien $m^r_L(f_q(v_q^1, v_q^2))$. Si $f_q(v_q^1, v_q^2) \notin V^2_{L^I_r}$, c'est immédiat.
  Supposons donc que $f_q(v_q^1, v_q^2) \in V^2_{L^I_r}$. Comme $(v_p^1, v_p^2) \notin E_{L^I_p} \wedge f((v_p^1, v_p^2)) = f_q((v_q^1, v_q^2))$, $P_1(f_q((v_q^1, v_q^2)))$ n'est pas vraie et, par construction de $E_{L^I_r}, f_q((v_q^1, v_q^2)) \notin E_{L^I_r}$.
- Si $(v_p^1 \in (V_{L_r} \cap V_{L_p}) \wedge v_p^2 \in Dom(f)) \vee (v_p^1 \in Dom(f) \wedge v_p^2 \in (V_{L_r} \cap V_{L_p}))$, les démonstrations étant les mêmes dans les deux cas, supposons que $v_p^1 \in (V_{L_r} \cap V_{L_p}) \wedge v_p^2 \in Dom(f)$. Alors $f^L_p((v_p^1, v_p^2)) = (f^L_r(v_p^1), f(v_p^2))$. De même que précédemment, on peut supposer que $f(v_p^2) \in V_{R^I_q}$ et $(v_p^1, f(v_p^2)) \in V^2_{L^I_r}$. Par construction de L^S_p, comme $(v_p^1, v_p^2) \in E_{L_p}$, $v_p 1 \notin V_{L_p^{C_q}}$ et $f(v_p^2) \in V_{L_p^{C^I_q} \downarrow_C R^I_q}$, $(v_p^1, f(v_p^2)) \in E_{L^S_p}$ et donc $(v_p^1, f(v_p^2)) \in E_{L_r}$. Ici, on a donc $v_1^p \in V_{L^I_p} \wedge f(v_2^p) \in V_{R^I_q} \wedge (v_1^p, v_2^p) \in E_{L_p} \wedge (v_1^p, v_2^p) \notin E_{L^I_p}$. Par construction de $E_{L^I_r}, (v_p^1, f(v_p^2)) \notin E_{L^I_r}$, et donc r supprime l'arc $f^L_r((v_p^1, f(v_p^2)))$. Or, comme $f(v_p^2) \in V_{R^I_q}, \exists v_q \in V_{L^I_q}, f_q(v_q) = f(v_p^2)$. Par définition de m^q_L, $m^q_L(v_q) = m^r_L(f_q(v_q))$ et $f^L_p(v_p^1)) = f^L_r(v_p^1)$.

Tout arc explicitement supprimé par r **est supprimé par l'application de** q **puis** p**.** Un arc (v_1, v_2) est explicitement supprimé par r si $\exists (v_r^1, v_r^2) \in V^2_{L_r^I}, (v_r^1, v_r^2) \in E_{L_r} \wedge (v_r^1, v_r^2) \notin E_{L_{r^I}} \wedge m_L^r(v_r^1, v_r^2) = (v_1, v_2)$.

Par disjonction de cas :

- si $(v_r^1, v_r^2) \in V^2_{L_q} \vee (v_r^1, v_r^2) V_{L_p}$, il est immédiat que $m_L^r(v_r^1, v_r^2)$ est également supprimé par q ou p. En effet, par construction de $V_{L_r^I}, \forall v_r \in V_{L_r}, (v_r \in V_{L_q} \implies ((v_r) \in V_{L_r^I} \Leftrightarrow v_r \in V_{L_q^I})) \wedge (v_r \in V_{L_p} \implies ((v_r) \in V_{L_r^I} \Leftrightarrow v_r \in V_{L_p^I}))$. Donc $(v_r^1, v_r^2) \in V^2_{L_q^I} \vee (v_r^1, v_r^2) V^2_{L_p^I}$. Par construction de E_{L_r} et $E_{L_r^I}, \forall e_r \in E_{L_r}, (e_r \in V^2_{L_q} \implies e_r \in E_{L_q} \wedge (e_r \in E_{L_r^I} \Leftrightarrow e_r \in E_{L_q^I})) \wedge (e_r \in V^2_{L_p} \implies e_r \in E_{L_p} \wedge (e_r \in E_{L_r^I} \Leftrightarrow e_r \in E_{L_p^I}))$. Donc $(v_r^1, v_r^2) \in E_{L_q} \backslash E_{L_q^I} \vee (v_r^1, v_r^2) E_{L_p} \backslash E_{L_p^I}$. Finalement, m_L^r coïncide avec m_L^p sur $V_{L_p} \cap V_{L_p}$ et avec m_L^q sur $V_{L_q} \cap V_{L_q}$.
- si $(v_r^1, v_r^2) \in V^2_{L_p^{C_q^I} \downarrow_C R_q^I}$. Puisque $(v_r^1, v_r^2) \in V^2_{L_r^I}$, par construction de $V_{L_r^I}, P_1(v_r^1)$ et $P_1(v_r^2)$. Par construction de L_p, $(v_r^1, v_r^2) \in E_{L_p} \implies (\exists (v_q^1, v_q^2) \in E^2_{L_q}, f_q(v_q^1, v_q^2) = (v_r^1, v_r^2)) \vee (\exists (v_p^1, v_p^2) \in E^2_{L_p}, f(v_q^1, v_q^2) = (v_r^1, v_r^2))$. Comme $(v_r^1, v_r^2) \in V^2_{L_r^I} \wedge (v_r^1, v_r^2) \in E_{L_r} \wedge (v_r^1, v_r^2) \in E_{L_r^I}$, par construction de E_{L_r}, $(v_r^1, v_r^2) \in V_{R_q^I} \implies \not P_1((v_1^r, v_2^r)) \vee \exists e_q \in E_{L_q}, f_q(e_q) = ((v_r^1, v_r^2)) \wedge e_q \notin E_{L_q^I}$. Alors :
  - Si $\not P_1((v_1^r, v_2^r))$, $\exists e_p \in E_{L_p}, f(e_p) = (v_1^r, v_2^r) \wedge e_p \notin E_{L_p^I}$. $m_L^p(e_p)$ est donc supprimé par p, et par définition de m_L^p, $m_L^p(e_p) = m_L^r((v_1^r, v_2^r))$.
  - Si $\exists e_q \in E_{L_q}, f_q(e_q) = ((v_r^1, v_r^2)) \wedge e_q \notin E_{L_q^I}$, $m_L^q(e_q)$ est donc supprimé par q, et par définition de m_L^q, $m_L^q(e_p) = m_L^r(v_1^r, v_2^r)$.
  - si $(v_r^1 \in V_{L_p^{C_q^I} \downarrow_C R_q^I} \wedge v_r^2 \in V_{L_p}) \vee (v_r^1 \in V_{L_p^{C_q^I} \downarrow_C R_q^I} \wedge v_r^2 \in V_{L_q}) \vee (v_r^1 \in V_{L_p} \wedge v_r^2 \in V_{L_p^{C_q^I} \downarrow_C R_q^I}) \vee (v_r^1 \in V_{L_q} \wedge v_r^2 \in V_{L_p^{C_q^I} \downarrow_C R_q^I})$, les démonstrations étant similaires, supposons que $(v_r^1 \in V_{L_p^{C_q^I} \downarrow_C R_q^I} \wedge v_r^2 \in V_{L_q})$. Par définition de la restriction, $v_r^1 \in V_{L_p^{C_q^I} \downarrow_C R_q^I} \implies v_r^1 \in V_{R_q^I}$. Comme $(v_1^r, v_2^r) \in V^2_{L_r^I} \wedge (v_1^r, v_2^r) \in E_{L_r}, v_1^r \in V_{R_q^I} \wedge v_2^r \in V_{L_q} \wedge (v_1^r, v_2^r) \notin E_{L_r^I} \implies \exists v_q \in V_{L_q^I}, f_q(v_q) = v_1^r \wedge (v_q, v_2^r) \in E_{L_q} \wedge (v_q, v_2^r) \notin E_{L_q^I}$. Ainsi, $m_L^q((v_q, v_2))$ est supprimé lors de l'application de q et par définition de m_L^q, $m_L^q((v_q, v_2^r)) = m_L^r(f_q(v_q), v_2^r) = m_L^r(v_1^r, v_2^r)$.

Ainsi, on obtient équivalence entre suppressions d'éléments découlant de l'application de r et de celle de la séquence q puis p.

B.2.3.3 Équivalence des ajouts de nœuds

Tout élément ajouté par p **est ajouté par** r**.** Tout élément el ajouté par p est tel que $el \in V_{R_p} \cup E_{R_p} \wedge \forall el_L \in V_{L_p^I} \cup E_{L_p^I}, f_p(el_L) \neq el$. (9)

Montrons que $el \in V_{R_r} \cup E_{R_r}$. Comme $L^I_{r_R} = L^I_p \cap L_{p^{C_q}} \subseteq L^I_p$, que $\forall el_L \in V_{L_p^I} \cup E_{L_p^I}, f_p(el_L) \neq el$ et que f_{r_R} est une restriction de f_p, alors $\forall el_L \in V_{L_{r_R}^I} \cup E_{L_{r_R}^I}, f_{r_R}(el_L) \neq el$. Or, $R_{L_{r_R}} = R_p$, donc $el \in V_{R_{r_R}} \cup E_{R_{r_R}}$ et el est ajouté lors de l'application de r_R.

Par construction de R_r, on a donc $el \in V_{R_r} \cup E_{R_r}$.

Montrons à présent que $\forall el_L \in V_{L_r^I}, f_r(el_L) \neq el$. Supposons que ça ne soit pas le cas, $\exists el_L \in V_{L_r^I}, f_r(el_L) = el$:

— supposons $el_L \in V^I_q$. Par définition de f_r, $f_r(el_L) = f_q(el_L)$. Or $el \in V_{R_p}$ et $V_{R_p} \cap Im(f_q) = \emptyset$. C'est donc impossible.
— supposons $el_L \in V^I_p$. Par définition de f_r, $f_r(el_L) = f_p(el_L)$. Donc $el_L \in V^I_p \wedge f_q(el_L) = el$, ce qui contredit (9).
— supposons donc que $el_L \in V_{L_p{}^{C^I_q}\downarrow_C R^I_q}$. Par définition de f_r, $f_r(el_L) = f_p(el_\nu)$ avec n'importe quel $el_p \in V_{L_p{}^{C^I_q}}$ tel que $f(el_p) = el_L$. Par définition de la restriction, si $el_L \in V_{L_p{}^{C^I_q}\downarrow_C R^I_q}$ un tel el_p existe effectivement. Comme $el_p \in Dom(f_p)$, $el_p \in V_{L^I_p}$. Ceci et $f_p(el_p) = el$ contredisent (9).

Ainsi, tout nœud ajouté par p est ajouté par r. Comme tout arc ajouté par p est présent dans R_r, il est immédiat que tout arc ajouté par p dont au moins une extrémité est ajouté par p est également ajouté par r.

Montrons donc que tout arc ajouté par p dont les deux extrémités sont invariantes par p est ajouté par r. Supposons que ça ne soit pas le cas, $el \in E_{R_q} \cap V^2_{R^I_q} \wedge \exists el_L \in E_{L^I_r}, f_r(el_L) = el$:

— si une extrémité ν de el_L est dans $V_{L^I_q}$, alors une extrémité de el est son image par f_q, ce qui est impossible comme $V_{R_p} \cap Im(f_q) = \emptyset$.
— supposons $el_L = (\nu_1, \nu_2) \in V^2_{L^I_p}$. Par construction de $E_{L^I_r}$, $(\nu_1, \nu_2) \in V^2_{L^I_p} \wedge (\nu_1, \nu_2) \in E_{L^I_r} \implies E_{L^I_p}$. D'où $el_L \in E_{L^I_p}$. Or, par définition de f_r, $f_r(el_L) = f_p(el_L)$. Ces deux dernières relations contredisent (9).
— supposons $el_L = (\nu_1, \nu_2) \in V^2_{L_p{}^{C^I_q}\downarrow_C R^I_q}$. Par construction de $E_{L^I_r}$, $P_1(el_L)$ est vraie et donc : $\forall el_p \in E_{L_p}, f(el_p) = el_L \implies el_p \in E_{L^I_p}$. Par définition de la restriction, si $el_L \in V_{L_p{}^{C^I_q}\downarrow_C R^I_q}$, $\exists el_p \in V^2_{L_p}, f(el_p) = el_L \wedge \exists el_q \in R^I_q, el_L = el_q$. Comme $el_L \in E_{L_r}$, par construction de L_r et définition de l'expansion : un tel e_l est dans E_{L_p} ou un tel el_q est dans E_{R_q}. Cette deuxième possibilité et (9) invalide la pré-condition 3.(a). Il existe donc effectivement un $el_p \in E_{L_p}, f(el_p) = el_L$, et donc $el_p \in E_{L^I_p}$. Par définition de f_r, on a alors $f_r(el_L) = f_p(el_p)$. Comme $P_1(el_L), el_p \in E_{L^I_p}$ et (9) est contredite.
— Finalement, supposons qu'une extrémité de el_L est dans $V_{L_p{}^{C^I_q}\downarrow_C R^I_q}$ et l'autre dans $V_{R^I_q}$. Les démonstrations étant similaires dans les deux cas, supposons $el_L = (\nu_1, \nu_2)$ et $\nu_1 \in V_{L^I_p} \wedge \nu_2 \in V_{R^I_q}$. Par construction de $E_{L^I_r}$, la propriété suivante est donc vraie : $\forall \nu_p \in V_{L^I_p}, f(\nu_p) = \nu_2 \wedge (\nu_1, \nu_p) \in E_{L_p} \implies (\nu_1, \nu_p) \in E_{L^I_p}$. (10)
Par définition de la restriction, si $\nu_2 \in V_{L_p{}^{C^I_q}\downarrow_C R^I_q}$, $\exists \nu_p \in V_{L_p{}^{C^I_q}}, f(\nu_p) = \nu_2$. Par construction de L_r et définition de l'expansion, comme $(\nu_1, \nu_2) \in E_{L_r}$, il existe un tel ν_p tel que $(\nu_1, \nu_p) \in E_{L_p}$. D'après (10), on a donc $(\nu_1, \nu_p) \in E_{L^I_p}$ et par définition de f_r, $f_r(\nu_1, \nu_2) = (f_p(\nu_1), f_p(\nu_p))$. Ceci invalide (9).

Ainsi, tout élément ajouté par p est ajouté par r.

Tout élément ajouté par l'application de q suivi de p est ajouté par r. Un élément el de G est ajouté par la séquence d'application q puis p s'il est ajouté par p ou s'il est ajouté par q et invariant pour p.

Montrons donc que tout élément ajouté par q et invariant pour p est ajouté par r. Pour ce faire, montrons tout d'abord que tout élément ajouté par q et invariant pour p est dans R_r. Puis, montrons qu'il est bien ajouté par r.

1. Montrons que tout élément ajouté par l'application de q *suivi de* p *est dans* R_r.

Un élément el est ajouté par q si et seulement si $el \in V_{R_q} \cup E_{R_q} \wedge (\forall el_L \in V_{L_q^I} \cup E_{R_q^I}, f_q(el_L) \neq el)$.

Il est en plus invariant pour p s'il n'est pas matché par m_L^q (i.e. pas dans R_p^C) ou qu'il est matché avec un élément invariant pour p et qu'il n'est pas fusionné avec un élément invariant pour q et p, i.e. :
$el \notin V_{R_p^C} \cup E_{R_p^C} \vee$
$(el \in V_{R_p^C} \cup E_{R_p^C} \wedge (\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies ((el_p \in V_{L_p^I} \cup E_{L_p^I}) \wedge (\forall el_p^2 \in (V_{L_p^I} \cup E_{L_p^I}), el_p^2 \notin (V_{L_p^{C_q}} \backslash V_{L_p^{C_q^I}}) \cup (E_{L_p^{C_q}} \backslash E_{L_p^{C_q^I}}) \implies f_p(el_p^2) \neq f_p(el_p)))))\vee$
$(el = (v_1, v_2) \wedge v_1 \notin V_{R_p^C} \wedge v_2 \in V_{R_p^C} \wedge \forall el_p \in V_{L_p^{C_q}}, f(el_p) = v_2 \implies el_p \in V_{L_p^I})\vee$
$(el = (v_1, v_2) \wedge v_1 \in V_{R_p^C} \wedge v_2 \notin V_{R_p^C} \wedge \forall el_p \in V_{L_p^{C_q}}, f(el_p) = v_1 \implies el_p \in V_{L_p^I})$. (11)

Supposons que el est explicitement supprimé par l'application de r_R à R_q selon (f, I). Cela implique par définition, $el \in f(R_r) \wedge el \notin f(R_r^I)$, i.e., $el \in f(V_{L_p^{C_q}}) \cup f(E_{L_p^{C_q}}) \wedge el \notin f(V_{L_p^{C_q}} \cap V_{L_p^I}) \cup f(E_{L_p^{C_q}} \cap E_{L_p^I})$. Ainsi, $\exists el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el$ (12). Par définition de R_p^C, comme $el \in f(V_{L_p^{C_q}}) \cup f(E_{L_p^{C_q}})$, on a $el \in V_{R_p^C} \cup E_{R_p^C}$. Alors, d'après (11), $\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies el_p \in V_{L_p^I} \cup E_{L_p^I}$. Ceci et (12) contredisent $el \notin f(V_{L_p^{C_q}} \cap V_{L_p^I}) \cup f(E_{L_p^{C_q}} \cap E_{L_p^I})$.

Supposons que el est implicitement supprimé par l'application de r_R à R_q selon (f, I). Cela implique par définition, $el = (v_1, v_2)$,
$(v_2 \notin f(V_{L_p^{C_q}}) \wedge v_1 \in f(V_{L_p^{C_q}}) \wedge v_1 \notin f(V_{L_p^{C_q}} \cap V_{L_p^I}))\vee$
$(v_1 \notin f(V_{L_p^{C_q}}) \wedge v_2 \in f(V_{L_p^{C_q}}) \wedge v_2 \notin f(V_{L_p^{C_q}} \cap V_{L_p^I}))$.
Les démonstrations étant similaires dans les deux cas, supposons que $(v_1 \notin f(V_{L_p^{C_q}}) \wedge v_2 \in f(V_{L_p^{C_q}}) \wedge v_2 \notin f(V_{L_p^{C_q}} \cap V_{L_p^I}))$. Ainsi, $\exists v_p \in V_{L_p^{C_q}}, f(v_p) = v_2$ (13). Par définition de R_p^C, comme $v_2 \in f(V_{L_p^{C_q}})$, on a $v_2 \in V_{R_p^C}$ et comme $v_1 \notin f(V_{L_p^{C_q}})$, on a $v_1 \notin V_{R_p^C}$. D'après (11) on a alors $\forall el_p \in V_{L_p^{C_q}}, f(el_p) = v_2 \implies el_p \in V_{L_p^I}$. Ceci et (13) contredisent $v_2 \notin f(V_{L_p^{C_q}} \cap V_{L_p^I})$, car $\exists v_p \in f(V_{L_p^{C_q}}), f(v_p) = v_2 (13) \wedge v_p \in V_{L_p^I}$.

Ainsi, tout élément ajouté par q et invariant pour p est dans R_r.

2. Montrons que tout élément ajouté par q *et invariant pour* p *est ajouté par* r.

Comme tout élément ajouté par q et invariant pour p est dans R_r, un élément el ajouté par q, invariant pour p et qui n'est pas ajouté par r et alors nécessairement invariant pour r. Montrons qu'un tel élément n'existe pas. Supposons qu'il existe un tel élément el, soit :

$el \in R_q \backslash R_q^I$, (13)
$(el \notin V_{R_q^C} \cup E_{R_q^C} \wedge \exists el_L \in V_{L_r^I} \cup E_{L_r^I}, f_r(el_L) = el)\vee$ (14)
$(el \in V_{R_q^C} \cup E_{R_q^C} \wedge (\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies ((el_p \in V_{L_p^I} \cup E_{L_p^I}) \wedge (\forall el_p^2 \in (V_{L_p^I} \cup E_{L_p^I}), el_p^2 \notin (V_{L_p^{C_q}} \backslash V_{L_p^{C_q^I}}) \cup (E_{L_p^{C_q}} \backslash E_{L_p^{C_q^I}}) \implies f_p(el_p^2) \neq f_p(el_p)))) \wedge (\exists el_L \in V_{L_r^I} \cup E_{L_r^I}, \exists el_L^p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_L^p) = el \wedge f_r(el_L) = f_p(el_L^p)))\vee$ (15)
$(el = (v_1, v_2) \wedge v_1 \notin V_{R_p^C} \wedge v_2 \in V_{R_p^C} \wedge (\forall v_p \in V_{L_p^{C_q}}, f(v_p) = v_2 \implies (v_p \in V_{L_p^I} \wedge (\forall v \in V_{L_p^{C_q}}, f_p(v) = f_p(v_p) \implies (v_1, f(v)) \notin E_{R_q^I}))) \wedge (\exists (v_1^L, v_2^L) \in E_{L_r^I}, \exists el_2^p \in V_{L_p^{C_q}}, f(el_2^p) = v_2^L \wedge f_r((v_1^L, v_2^L)) = (v_1^L, f_p(v_2^p)))\vee$
$(el = (v_1, v_2) \wedge v_1 \in V_{R_p^C} \wedge v_2 \notin V_{R_p^C} \wedge (\forall el_p \in V_{L_p^{C_q}}, f(el_p) = v_1 \implies (el_p \in V_{L_p^I} \wedge (\forall v \in$

$V_{L_p^{C_q}}, f_p(v) = f_p(v_p) \implies (f(v), v_2) \notin E_{R_q^I}))) \wedge (\exists(v_1^L, v_2^L) \in E_{L_r^I}, \exists el_1^p \in V_{L_p^{C_q}}, f(el_1^p) = v_1^L \wedge f_r((v_1^L, v_2^L)) = (f_p(v_1^p), v_2^L)) \vee$ (16)
$(el = (v_1, v_2) \wedge v_1 \in V_{R_p^C} \wedge v_2 \notin V_{R_p^C} \wedge (\forall el_p \in V_{L_p^{C_q}}, f(el_p) = v_1 \implies el_p \in V_{L_p^I}) \wedge (\exists(v_1^L, v_2^L) \in E_{L_r^I}, \exists el_1^p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_1^p) = v_1^L \wedge f_r((v_1^L, v_2^L)) = (f_p(v_1^p), v_2^L))$. (17)

Par disjonction de cas :

Supposons (14), soit $el \notin V_{R_q^C} \cup E_{R_q^C} \wedge \exists el_L \in V_{L_r^I} \cup E_{L_r^I}, f_r(el_L) = el$.
Par définition de f_r, $f_r(el_L) = el \in V_{R_q} \cup E_{R_q} \implies el_L \in (V_{L_r^I} \cap V_{L_q}) \cup (E_{L_r^I} \cap E_{L_q})$. De plus, par construction de $V_{L_r^I}$, un nœud dans L_r et dans L_q est dans L_r^I si et seulement s'il est dans L_q^I, donc $el_L \in (V_{L_r^I} \cap V_{L_q}) \cup (E_{L_r^I} \cap E_{L_q}) \implies el_L \in (V_{L_r^I} \cap V_{L_q^I}) \cup (V_{L_r^I} \cap V_{L_q^I})^2$.
Si $el \in V_{R_q}$, on obtient donc $el_L \in V_{R_q^I}$ et $f_r(el_L) = f_q(el)$, or $f_r(el_L) = el$ d'après (14). Ainsi, $el \in V_{R_q^I}$ et ceci contredit (13).
Si $el \in E_{R_q}$, comme $el_L \in V_{L_q^I}^2 \wedge el_L \in E_{L_r^I}$, par construction de $E_{L_r^I}$, $el_L \in E_{L_q^I}$. Par définition de f_r, $f_r(el_L) = f_q(el)$, or $f_r(el_L) = el$ d'après (14). Ainsi, $el \in E_{R_q^I}$ et ceci contredit (13).

Supposons (15), soit $el \in V_{R_q^C} \cup E_{R_q^C} \wedge (\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies ((el_p \in V_{L_p^I} \cup E_{L_p^I}) \wedge (\forall el_p^2 \in (V_{L_p^I} \cup E_{L_p^I}), el_p^2 \notin (V_{L_p^{C_q}} \backslash V_{L_p^{C_q^I}}) \cup (E_{L_p^{C_q}} \backslash E_{L_p^{C_q^I}}) \implies f_p(el_p^2) \neq f_p(el_p)))) \wedge (\exists el_L \in V_{L_r^I} \cup E_{L_r^I}, \exists el_L^p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_L^p) = el \wedge f_r(el_L) = f_p(el_L^p))$.

Par définition de f_r, $f_r(el_L) \in V_{R_p} \implies el_L \notin V_{L_q^I}$. De même, $f_r(el_L) \in E_{R_p}$ implique qu'aucune extrémité de el_L n'est dans $V_{L_q^I}$.

Supposons que $el_L \in V_{L_p^I} \cup E_{L_p^I}$. Par construction de f_r, $f_r(el_L) = f_p(el_L)$ et, d'après (15), $\exists el_L^p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_L^p) = el \wedge f_r(el_L) = f_p(el_L^p)$. (18)
Or, comme $el_L \in V_{L_p^I} \cup E_{L_p^I}$, par construction de L_r, $el_L \in L_p^S$. Donc, par définition de L_p^S et de r_L, $el_L \notin f_{r_L}(L_{r_L}) = L_p^{C_q}$. Par conséquent, $el_L \notin (V_{L_p^{C_q}} \backslash V_{L_p^{C_q^I}}) \cup (E_{L_p^{C_q}} \backslash E_{L_p^{C_q^I}})$. Ceci et $el_L \in (V_{L_p^I} \cup E_{L_p^I})$, donnent d'après (15), $\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies f_p(el_p) \neq f_p(el_L)$. Ceci contredit (18).

Supposons que $el_L = (el_L^1, el_L^2)$ avec une extrémité dans $V_{L_p^I}$ et l'autre dans $V_{L_p^{C_q^I} \downarrow_C R_q^I}$. Les démonstrations étant similaires dans les deux cas, supposons $el_L^1 \in V_{L_p^I} \wedge el_L^2 \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$.
Par définition de f_r, $f_r((el_L^1, el_L^2)) = (f_p(el_L^1), f_p(el_L^r))$ avec n'importe quel $el_p^r \in V_{L_p^{C_q^I}}$ tel que $f(el_p^r) = el_L^2$. Comme précédemment, $el_L^1 \notin V_{L_p^{C_q}}$, et donc $(el_L^1, el_p^r) \notin E_{L_p^{C_q}} \backslash E_{L_p^{C_q^I}}$. Donc, d'après (15), $\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies f_p(el_p) \neq f_p((el_L^1, el_p^r))$.
Or, d'après (15), $\exists el_L^p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_L^p) = el \wedge f_r(el_L) = f_p(el_L^p)$. Comme $f_r(el_L) = f_p((el_L^1, el_p^r))$, on obtient $f_p((el_L^1, el_p^r)) = f_p(el_L^p)$ ce qui est impossible.

Supposons finalement que $el_L \in L_p^{C_q^I} \downarrow_C R_q^I$. Par définition de f_r et de $E_{L_r^I}$, $f_r(el_L) = f_p(el_p^r)$ avec n'importe quel $el_p^r \in V_{L_p^{C_q^I}} \cup E_{L_p^{C_q^I}} \vee (el_p^r \in V_{L_p^{C_q^I}}^2 \wedge el_p^r \notin E_{L_p^{C_q}} \wedge f(el_p^r) \in E_{R_q^I})$ tel que $f(el_p^r) = el_L$.
Comme $el_p^r \in V_{L_p^{C_q^I}} \cup E_{L_p^{C_q^I}} \vee (el_p^r \in V_{L_p^{C_q^I}}^2 \wedge el_p^r \notin E_{L_p^{C_q}})$, immédiatement $el_p^r \notin (V_{L_p^{C_q}} \backslash V_{L_p^{C_q^I}}) \cup (E_{L_p^{C_q}} \backslash E_{L_p^{C_q^I}})$. Donc, d'après (15), $\forall el_p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_p) = el \implies f_p(el_p) \neq f_p(el_p^r)$.(19)

D'après (15), $\exists el_L^p \in V_{L_p^{C_q}} \cup E_{L_p^{C_q}}, f(el_L^p) = el \wedge f_r(el_L) = f_p(el_L^p)$. Comme $f_r(el_L) = f_p(el_p^r)$, on obtient $f_p(el_p^r) = f_p(el_L^p)$ ce qui contredit (19).

Le cas (17) est similaire au cas (15) et sa démonstration est quasi-identique.

Supposons (16), c'est à dire que el est un arc dont une extrémité est dans $V_{R_q^C}$ et l'autre non. Les démonstrations étant similaires, fixons cette extrémité et supposons que $el = (v_1, v_2) \wedge v_1 \in V_{R_p^C} \wedge v_2 \notin V_{R_p^C} \wedge (\forall el_p \in V_{L_p^{C_q}}, f(el_p) = v_1 \implies (el_p \in V_{L_p^I} \wedge (\forall v \in V_{L_p^{C_q}}, f_p(v) = f_p(v_p) \implies (f(v), v_2) \notin E_{R_q^I}))) \wedge (\exists (v_1^L, v_2^L) \in E_{L_r^I}, \exists el_1^p \in V_{L_p^{C_q}}, f(el_1^p) = v_1^L \wedge f_r((v_1^L, v_2^L)) = (f_p(v_1^p), v_2))$.

Par définition de f_r, $v_2^L \in V_{L_r^I} \cap V_{L_q^I}$ et $v_1^L \in V_{L_r^I} \cap V_{L_p^I} \vee v_1^L \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$. Par construction de L_r et par définition de l'expansion, il ne peut exister d'arc entre un nœud de $V_{L_r^I} \cap V_{L_q^I}$ et un nœud de $V_{L_r^I} \cap V_{L_p^I}$. Nécessairement donc, $v_1^L \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$.

Par construction de L_r et par définition de l'expansion, $(v_1^L, v_2^L) \in E_{L_r}$, $v_2^L \in V_{L_r^I} \cap V_{L_q^I}$ et $v_1^L \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$ impliquent que $\exists v_q^1 \in V_{L_q^I}$ tel que $(v_q^1, v_L^2) \in E_{L_q}$ et $f_q(v_q^1) = v_1^L$.

Par construction de $E_{L_r^I}$, $(v_1^L, v_2^L) \in E_{L_r^I}$, $v_2^L \in V_{L_r^I} \cap V_{L_q^I}$ et $v_1^L \in V_{L_p^{C_q^I} \downarrow_C R_q^I}$ impliquent que $\forall v_q \in V_{L_q^I}, f_q(v_q) = v_1^L \wedge (v_q, v_2^L) \in E_{L_q} \implies (v_q, v_2^L) \in E_{L_q^I}$. Ainsi, $(v_q^1, v_2^L) \in E_{L_q^I}$. Donc $f_q(v_q^1, v_2^L) \in E_{R_q^I}$, et $f_q(v_q^1, f_q(v_2^L)) = (v_1^L, f_q(v_2)$.

Par définition de f_r, $f_r(v_1^L, v_2^L) = (f_p(v_p), f_q(v_2^L))$ avec n'importe quel $v_p \in V_{L_p^{C_q}}$, tel que $f(v_p) = v_1^L$. D'après (16), on a donc $f_q(v_2^L) = v_2$ et $f_p(v_p) = f_p(v_1^p)$. Or $(v_1^L, f_q(v_2)) = (f(v_p), v_2)$ et $(v_1^L, v_2) \in E_{R_q^I}$. Ceci contredit (16).

Ainsi, tout élément ajouté par l'application de q suivi de p est dans R_r et n'est pas invariant par r. Tout élément ajouté par l'application de q suivi de p est donc ajouté par r.

Tout élément ajouté par r **est également ajouté par l'application de** q **suivi de** p**.** Par construction de R_r, tout élément de R_r est soit un élément de R_p, soit un élément de R_q qui n'est pas supprimé lors de l'application de p suivant m_p^L (soit non matché par m_p^L soit matché avec un élément invariant), soit un arc (v_1, v_2) tel que :
$(v_1 \in V_{R_q} \wedge v_1 \notin V_{R_q^C} \wedge v_2 \in V_{R_p} \wedge (\exists v_p \in V_{L_p^I}, f_p(v_p) = v_2 \wedge (v_1, f(v_p)) \in E_{R_q}))\vee$
$(v_1 \in V_{R_p} \wedge v_2 \in V_{R_q} \wedge v_2 \notin V_{R_q^C} \wedge (\exists v_p \in V_{L_p^I}, f_p(v_p) = v_1 \wedge (f(v_p), v_2) \in E_{R_q}))\vee$
$(v_1 \in V_{R_p} \wedge v_2 \in V_{R_p} \wedge (\exists (v_p^1, v_p^2) \in V_{L_p^I}^2, f_p((v_p^1, v_p^2)) = (v_1, v_2) \wedge (f(v_p^1), f(v_p^2)) \in E_{R_q} \wedge (f_p((v_p^1, v_p^2)) \notin E_{R_p})))$.

Il est immédiat que tout élément de R_r est présent après l'application de q suivi de p. La règle r peut donc ajouter un élément el que ni q ni p n'ajoutent uniquement si el est invariant pour q et p.

Supposons donc qu'il existe un tel élément. Supposons que cet élément soit dans R_q ou R_p, soit :
$el \in R_r \wedge (\forall el_r \in L_r^I, f_r(el_L) \neq el) \wedge$
(20) $((el \in R_q \wedge (\exists el_q \in L_q^I, f_q(el_q) = el))\vee$
(21) $(el \in R_p \wedge (\exists el_p \in L_p^I, f_p(el_p) = el \wedge$

$(el_p \in L_p^{C_q} \implies el_p \in L_p^{C_q^I}) \wedge (el_p = (v_p^1, v_p^2) \wedge (v_p^1 \in V_{L_p^{C_q}} \implies v_p^1 \in V_{L_p^{C_q^I}}) \wedge (v_p^2 \in V_{L_p^{C_q}} \implies v_p^2 \in V_{L_p^{C_q^I}}))$. (22)

Par disjonction de cas,

1. si (21) et $el \in R_q$, comme $R_r = r_R(f, I)(R_q)$ et que $R_{r_R} = R_p$ qui est disjoint de R_q, $el \notin R_q^C$, et, si el est un arc, aucune de ses extrémités n'est dans R_q^C. Posons $el_q \in V_{L_q^I} \cup E_{L_q^I}$ tel que $f_q(el_q) = el$.
   Par définition de la réécriture, les éléments de L_p^S appartenant à R_q sont ceux appartenant à R_{r_L}, soit $L_p^{C_q^I} \downarrow_C R_q^I$. Par définition de la restriction, ces éléments sont images par f d'un élément de $L_p^{C_q^I}$. Comme $el \notin R_q^C$ et que, si el est un arc, aucune de ses extrémités n'est dans R_q^C, ni el ni une de ses potentielles extrémité n'est image d'un élément de L_p par f. Par conséquent, $el \notin L_p^S$ et si el est un arc, aucune de ses extrémité n'est dans L_p^S.
   Par définition de l'expansion, les éléments de L_q n'appartenant pas à $L_q \uparrow_(f_q, I)L_p^S$ sont ceux dont l'image par f_q, ou dont l'image d'au moins une extrémité par f_q est dans L_p^S. Comme $f_q(el_q) = el$ et que ni el ni une de ses extrémité n'est dans L_p^S, $el_q \in L_q \uparrow_(f_q, I)L_p^S$. C'est à dire que $el_q \in L_r$.
   Comme $el_q \in (V_{L_q} \cap V_{L_r}) \cup (E_{L_q} \cap E_{L_r}), el_q \in V_{L_q^I} \cup E_{L_q^I} \implies el_q \in V_{L_r^I} \cup E_{L_r^I}$. Ainsi, $el_q \in V_{L_q^I} \cup E_{L_q^I}$. Or par définition de f_r, $f_r(el_q) = f_q el_q$, qui est égal à el. Ceci contredit (20).
2. si (22) et $el \in R_p$, posons $el_p \in L_p^I$ tel que $f_p(el)$.
   (a) Si $el_p \in L_p^{C_q}$, d'après (22) $el_p \in L_p^{C_q^I}$. Ainsi, $f(el_p) \in L_p^{C_q^I} \downarrow_C R_q^I = r_R^I$ et donc $f(el_p) \in L_p^S$. Alors, par définition de l'expansion, $f(el_p) \in L_q \uparrow_(f_q, I)L_p^S = L_r$. Comme $el_p \in L_p^I$, la précondition 3.(b) implique la validité de $P_1(f(el_p))$. Si $f(el)$ est un nœud, on a donc $f(el_p) \in L_r^I$. Si $f(el)$ est un arc, son appartenance à $L_p^{C_q^I} \downarrow_C R_q^I$ valide la seconde portion de la condition nécessaire à son appartenance à L_r^I. Donc $f(el_p) \in L_r^I$. Par définition de f_r, $f_r(f_e l_p) = f_p(el_p)$ qui est elle même égale à el. Ceci contredit (20).
   (b) Si $el_p \notin L_p^{C_q}$, il est par définition dans L_p^S et comme il n'est pas l'image d'un élément de L_q par f_q, il est aussi dans L_r. Or $el_p \in L_p^I \cap L_r \implies el_p \in L_r^I \wedge f_r(el_p) = f_p(el_p)$ et donc $= el$. Ceci contredit (20).
   (c) Si $el_p = (v_p^1, v_p^2)$ et qu'une de ses extrémités est dans $V_{L_p^{C_q}}$, les démonstrations étant les mêmes, supposons que ce soit v_p^1. Comme dans 2.(b), on obtient $v_p^2 \in V_{L_r^I}$. Comme dans 2.(a), on obtient $f(v_p^1) \in V_{L_r^I}$. Comme el_p est invariant pour p par hypothèse $\forall(v_p \in V_{L_p^I}), f(v_p) = f(v_p^1) \wedge (v_p, v_p^2) \in E_{L_r} \implies (v_p, v_p^2) \in E_{L_r^I}$, et donc $(f(v_p^1), v_p^2) \in E_{L_r^I}$. Par définition de f_r, $f_r(f(v_p^1), v_p^2) = f_p((v_p^1, v_p^2))$ qui est elle même égale à el. Ceci contredit (20).
   (d) Finalement, si $el_p = (v_p^1, v_p^2) \notin E_{L_p^{C_q}} \wedge el_p \in V^2_{L_p^{C_q}}$. Comme dans 2.(a), on obtient $f(v_p^1) \in V_{L_r^I}$ et $f(v_p^2) \in V_{L_r^I}$. Comme $el_p \notin E_{L_p^{C_q}} = L_{r_L}$, el_p est invariant pour r_L et son appartenance à L_p implique que $f(el_p) \in E_{L_r}$. Son invariance par p et q implique $P_1((f(v_p^1), f(v_p^2)))$ et $\forall eq \in E_{L_q}, f_q(eq) = (f(v_p^1), f(v_p^2)) \implies eq \in E_{L_q^I}$. Donc $f(el_p) \in E_{L_r^I}$ et par définition de f_r, $f_r(f(el_p)) = f_p(el_p)$. Comme = $f_p(el_p) = el$, ceci contredit (20).

Supposons que ce ne soit pas un élément de R_q ou R_p. C'est donc un arc de R_r, invariant par p et q tel que : $(v_1, v_2) \in R_r \wedge (\forall el_r \in L_r^I, f_r(el_L) \neq el) \wedge$ (23) $((v_1 \in V_{R_q^I} \wedge v_1 \notin V_{R_q^C} \wedge v_2 \in V_{R_p^I} \wedge (\exists v_p \in V_{L_p^I}, f_p(v_p) = v_2 \wedge (v_1, f(v_p)) \in E_{R_q^I}))\vee$
$(v_1 \in V_{R_p^I} \wedge v_2 \in V_{R_q^I} \wedge v_2 \notin V_{R_q^C} \wedge (\exists v_p \in V_{L_p^I}, f_p(v_p) = v_1 \wedge (f(v_p), v_2) \in E_{R_q^I}))\vee$
(24) $(v_1 \in V_{R_p^I} \wedge v_2 \in V_{R_p^I} \wedge (\exists (v_p^1, v_p^2) \in V^2_{L_p^I}, f_p((v_p^1, v_p^2)) = (v_1, v_2) \wedge (f(v_p^1), f(v_p^2)) \in E_{R_q^I} \wedge (f_p((v_p^1, v_p^2)) \notin E_{R_p}))))$. (25).

Supposons (24). Les démonstrations étant similaires dans les deux cas, supposons $v_1 \in V_{R_q^I} \wedge v_2 \in V_{R_p^I} \wedge (\exists v_p \in V_{L_p^I}, f_p(v_p) = v_2 \wedge (v_1, f(v_p)) \in E_{R_q^I})$. Par définition de $L_p^{C_q^I}$, $v_p \in V_{L_p^{C_q^I}}$. Comme dans 2.(a), on obtient donc $f(v_p) \in V_{L_r^I}$. Par définition de R_q^I, $\exists (v_q^1, v_q^2) \in V^2_{L_q^I} \cap E_{L_q^I}$ tel que $f_q((v_q^1, v_q^2)) = (v_1, f(v_p))$. Comme dans 1. on obtient $v_q^1 \in V_{L_r^I}$.

Comme $(v_q^1, f(v_p)) \in V^2_{L_q \uparrow_{(f_q, I)} L_p^S} (= V^2_{L_r})$, que $f_q(v_q^1) = f(v_p)$ et que $(v_q^1, v_q^2) \in E_{L_q^I}$, par définition de l'expansion $(v_q^1, f(v_p)) \in E_{L_q \uparrow_{(f_q, I)} L_p^S}$ et donc $(v_q^1, f(v_p)) \in E_{L_r}$.

Comme (v_q^1, v_q^2) est invariant par q, $\forall (v_1^q, v_2^q) \in E_{L_q}, f_q((v_1^q, v_2^q)) = f_q((v_q^1, v_q^2)) \implies (v_1^q, v_2^q) \in E_{L_q^I}$. En particulier, ceci donne $\forall v_2^q \in V_{L_q^I}, f_q(v_2^q) = f(v_p) \implies (v_1, v_2^q) \in E_{L_q^I}$. Avec $v_q^1 \in V_{L_r^I} \cap V_{L_q^I}$ et $f(v_p) \in V_{L_r^I} \cap V_{R_q^I}$, ceci donne par définition de $E_{L_r^I}$, $(v_q^1, f(v_p)) \in E_{L_r^I}$.

Par définition de f_r, $f_r(v_q^1, f(v_p)) = (f_q(v_q^1), f_p(v_p))$, ce qui d'après (24) est égal à (v_1, v_2). Ceci contredit (23).

Supposons finalement (25) $(v_1 \in V_{R_p^I} \wedge v_2 \in V_{R_p^I} \wedge (\exists (v_p^1, v_p^2) \in V^2_{L_p^I}, f_p((v_p^1, v_p^2)) = (v_1, v_2) \wedge (f(v_p^1), f(v_p^2)) \in E_{R_q^I}) \wedge (f_p((v_p^1, v_p^2)) \notin E_{R_p}))$.

Par définition de $L_p^{C_q^I}$, $(v_p^1, v_p^2) \in V^2_{L_p^{C_q^I}}$. Comme dans 2.(a), on obtient donc $(f(v_p^1), f(v_p^2)) \in V^2_{L_p^S}$ et $(f(v_p^1), f(v_p^2)) \in V^2_{L_r^I}$. Par définition de R_q^I, $(f(v_p^1), f(v_p^2)) \in E_{R_q^I}) \implies \exists (v_q^1, v_q^2) \in E_{L_q^I}, f_q((v_q^1, v_q^2)) = (f(v_p^1), f(v_p^2))$. Par conséquent et par définition de l'expansion, comme $(f(v_p^1), f(v_p^2)) \in V^2_{L_p^S}$, on obtient $(f(v_p^1), f(v_p^2)) \in E_{L_q \uparrow_{(f_q, I) L_p^S}} = E_{L_r}$.

L'invariance par p et q implique $P_1((f(v_p^1), f(v_p^2)))$ et $\forall e_q \in E_{L_q}, f_q(e_q) = (f(v_p^1), f(v_p^2)) \implies e_q \in E_{L_q^I}$. Par conséquent $(f(v_p^1), f(v_p^2)) \in E_{L_r^I}$. Par définition de f_r, $f_r((f(v_p^1), f(v_p^2))) = f_p((v_p^1, v_p^2))$ qui est égal à (v_1, v_2) d'après (25). Ceci contredit (23).

Ainsi, tout élément invariant par l'application de q suivi de p est également invariant par r. Comme tout élément de R_r est présent après l'application de q suivi de p, tout élément ajouté par r est également ajouté par l'application de q suivi de p.

Finalement, l'application de r selon m_r^L et l'application de $p\_m_p^L.q\_m_q^L$ ont les mêmes effets sur G et $p\_m_p^L.q\_m_q^L(G) = r\_m_r^L(G)$.

□

B.3 Spécialisation : preuve du théorème 4.6

Pour prouver ce théorème et s'intéresser au cas le plus général, nous supposons que la recherche de motif se fait à l'aide d'un morphisme. La démonstration peut aisément être transposée à la considération d'un homomorphisme ou un isomorphisme de sous graphe induit.

Démonstration. Soit un graphe G, une règle de réécriture $p = (L_p \xrightarrow{m_p} R_p)$ et une règle $q = (L_q \xrightarrow{m_q} R_q)$ vérifiant les hypothèses du théorème 4.6. Reprenons les notations introduites dans le dit théorème, et supposons qu'il existe un morphisme $m_L^q = (f_L^q, I_L^q) : L_q \to G$ tel que q est applicable à G selon m_L^q.

Pour prouver le théorème, il s'agit de démontrer l'existence d'un morphisme m_L^p selon lequel p est applicable G et tel que $p\_m_L^p(G) = q\_m_L^q(G)$.

B.3.1 Existence et applicabilité

Existence d'un morphisme. Par hypothèse comme q et p vérifie les hypothèses du théorème, il existe un morphisme (cohérent) $m_L = (f_L, I_L)$ tel que $L_p \xrightarrow{m_L} L_q$ (prop. 1) et $Aff_{I_L}(L_q) = L_q$ (prop 1. (a)).

Donc, $L_p \xrightarrow{m_L} Aff_{I_L}(L_q) = L_q \xrightarrow{(Id, I_L^q)} Aff_{I_L^q}(L_q) \xrightarrow{m_L^q} G$. D'après la remarque 3.1, $(f_L^q \circ f_L, I_L^p \cup I_L)$ est un morphisme (cohérent) de L_p vers G.

Par hypothèse, les morphismes m_q et m_p ont un ensemble d'identification vide, avec de plus $L_p \xrightarrow{(f_L, I_L)} L_q$ et $R_p \xrightarrow{(f_R, I_R)} R_q$. Une partie de l'ensemble I_R est donc contenu dans I_L, toute identification touchant un attribut apparaissant dans $Dom(f_q)$ étant commune aux deux. Ainsi, toute identification pouvant avoir un impact sur la consistance de l'ensemble $I_L^p \cup I_L \cup I_R$ est contenue dans I_L (les autres identifications touchant les attributs de nœuds ajoutés lors de l'application de la règle et dont les variables sont libres).

Par conséquent, $m_L^p = (f_L^q = f_L^q \circ f_L, I_L^p = I_L^p \cup I_L \cup I_R)$ est un morphisme (cohérent) tel que $L_p \xrightarrow{m_L^p} G$.

Satisfaction de la seconde condition d'applicabilité. Supposons que m_L^p ne remplisse pas la seconde condition d'applicabilité, soit $\exists (v_1, v_2) \in V^2_{L_p^I}, (v_1, v_2) \notin E_{L_p} \wedge f_p((v_1, v_2)) \in E_{R_p} \wedge f_L^p((v_1, v_2)) \in E_G$. (i)

Dans ce cas, en notant $(v_3, v_4) = f_L((v_1, v_2)) \in V^2_{L_q}$, on a $f_L^q((v_3, v_4)) \in E_G$ (ii) comme $f_L^p((v_1, v_2)) \in E_G$ et $f_L^p = f_L^q \circ f_L$.

D'après (1), $(v_1, v_2) \in V^2_{L_p^I}$. Avec l'hypothèse 1.(b) on a donc $(v_3, v_4) \in V^2_{L_q^I}$ (iii).

D'après l'hypothèse 2.(b) du théorème, on a $f_q((v_3, v_4)) = f_R(f_p((v_1, v_2)))$ (iv), avec $f_p((v_1, v_2)) \in E_{R_p}$ selon (i). Comme h_R est un homomorphisme de R_p dans R_q (prop. 2), $f_q((v_3, v_4)) \in E_{R_q}$ (v).

D'après (i), $f_p(v_1, v_2) \in E_{R_p} \backslash f_p(E_{L_p^I})$. L'hypothèse 2.(c) et (iv) donnent $f_q((v_3, v_4)) \in E_{R_q} \backslash f_q(E_{L_q^I})$ et donc $(v_3, v_4) \notin E_{L_q^I}$ (vi) comme $f_q((v_3, v_4)) \in E_{R_q}$ d'après (v). Supposons que $(v_3, v_4) \in E_{L_q}$. Selon (vi), l'hypothèse 1.(e) et comme $(v_1, v_2) \notin E_{L_p}$ d'après (i), on obtient $v_3 \in (V_{L_q} \backslash V_{L_q^I}) \vee (v_4 \in (V_{L_q} \backslash V_{L_q^I})$. Ceci contredit (iii) d'où $(v_3, v_4) \notin E_{L_q}$ (vii).

(iii), (vii), (v) et (ii) donnent $(v_3, v_4) \in V^2_{L^I_q} \wedge (v_3, v_4) \notin E_{L_q} \wedge f_q((v_3, v_4)) \in E_{R_q} \wedge f^q_L((v_3, v_4)) \in E_G$. Comme q est applicable à G par rapport à m^q_L, ceci est impossible car contrariant la seconde condition d'application. Par conséquent, m^p_L remplit la seconde condition d'applicabilité.

Satisfaction de la troisième condition d'applicabilité. Supposons que m^p_L ne remplisse pas la troisième condition d'applicabilité, soit $\exists (v_1, v_2) \in L^2_p, f^p_L(v_1) = f^p_L(v_2) \wedge ((v_1 \notin Dom(f_p) \wedge v_2 \in Dom(f_p)) \vee (v_1 \in Dom(f_p) \wedge v_2 \notin Dom(f_p)) \vee ((v_1, v_2) \in Dom(f_p)^2 \wedge f_p(v_1) \neq f_p(v_2)))$.

Rappelons que $f^p_L = f^q_L \circ f_L$ et posons $(v_3, v_4) = f_L((v_1, v_2))$ avec $(v_3, v_4) \in V_{L_q}$. Comme $f^p_L(v_1) = f^p_L(v_2)$, $f^q_L(v_3) = f^q_L(v_4)$. Par disjonction de cas :

- Si $(v_1 \notin Dom(f_p) \wedge v_2 \in Dom(f_p)) \vee (v_1 \in Dom(f_p) \wedge v_2 \notin Dom(f_p))$: la démonstration étant similaire dans les deux cas, supposons $(v_1 \notin Dom(f_p) \wedge v_2 \in Dom(f_p))$. Comme $Dom(f_p) = V_{L^I_p}$ et $V_{L^I_q} = Dom(f_q)$, $v_2 \in V_{L^I_p}$. D'après l'hypothèse 1. (b), $v_4 \in V_{L^I_q} (= Dom(f_q))$. m^q_L remplit la troisième condition d'applicabilité par hypothèse, d'où $f^q_L(v_3) = f^q_L(v_4) \wedge v_4 \in Dom(f_q) \implies v_3 \in Dom(f_q)$. Donc $v_3 \in Dom(f_q)$. D'après la condition 1. (c), $v_1 \notin Dom(f_p) \implies f_L(v_1) = v_3 \notin Dom(f_q)$. On obtient donc une contradiction.
- Si $(v_1, v_2) \in Dom(f_p)^2 \wedge f_p(v_1) \neq f_p(v_2)$: d'après l'hypothèse 1. (b) et comme $(v_1, v_2) \in Dom(f_p)^2$ on a également $(v_3, v_4) \in Dom(f_q)^2$. Comme m^q_L remplit la troisième condition d'applicabilité, ceci implique que $f^q_L(v_3) = f^q_L(v_4)$.
  D'après l'hypothèse 2. (b), $f_R(f_p(v_1)) = f_q(f_L(v_1)$ et $f_R(f_p(v_2)) = f_q(f_L(v_2))$. Ainsi, on a $f_R(f_p(v_1)) = f_q(v_3)$ et $f_R(f_p(v_2)) = f_q(v_4)$, et donc $f_R(f_p(v_1)) = f_R(f_p(v_2))$. La fonction f_R étant injective, $f_p(v_1) = f_p(v_2)$. Ceci contredit l'hypothèse de disjonction.

Le morphisme m^p_L remplit donc la troisième condition d'applicabilité.

La règle p est par conséquent applicable à G selon m^p_L.

B.3.2 Égalité, équivalence des applications

Pour démontrer que q_m^q_L(G) = p_m^p_L(G), montrons que l'application de q selon m^q_L est équivalente à l'application de p selon m^p_L, c'est à dire que les deux suppriment et ajoutent les mêmes éléments (l'ajout étant à un isomorphisme près).

Équivalence des suppressions de nœuds. Supposons que l'application de q à G selon m^q_L entraîne la suppression d'un nœud v. Il existe donc $v_q \in V_{L_q} \backslash V_{L^I_q}$ tel que $f^q_L(v_q) = v$. D'après l'hypothèse 1.(c), $\exists v_p \in V_{L_p} \backslash V_{L^I_p}$ tel que $f_L(v_p) = v_q$. Or par construction de f^p_L, $f^p_L(v_p) = f^q_L(v_q)$ et donc $f^p_L(v_p) = v$. L'application de p à G selon m^p_L entraîne donc la suppression de v.

Respectivement, supposons que l'application de p à G selon m^p_L entraîne la suppression d'un nœud v. Il existe donc $v_p \in V_{L_p} \backslash V_{L^I_p}$ tel que $f^p_L(v_p) = v$. D'après 1.(c), $f_L(v_p) \in V_{L_q} \backslash V_{L^I_q}$. Or par construction de f^p_L, $f^q_L(f_L(v_p)) = f^p_L(v_p) (= v)$ et donc $\exists v_q (= f_L(v_p)) \in V_{L_q} \backslash V_{L^I_q}$ tel que $f^q_L(v_q) = v$. L'application de q à G selon m^q_L entraîne donc la suppression de v.

Les deux transformations considérées suppriment donc les mêmes nœuds de G.

Équivalence des suppressions d'arcs. Supposons que l'application de p à G selon m_L^p entraîne la suppression d'un arc e. Si la suppression est implicite (i.e., e devenant suspendu), l'équivalence des suppressions de nœuds implique immédiatement que e est également supprimé par l'application de q selon m_L^q. Si la suppression est explicite, cela signifie que $\exists e_p \in E_{L_p} \backslash E_{L_p^I}$ tel que $f_L^p(e_p) = e$. D'après 1.(d), $f_L(e_p) \in E_{L_q} \backslash E_{L_q^I}$. Or par construction de f_L^p, $f_L^q(f_L(e_p)) = f_L^p(e_p)(= e)$ et donc $\exists e_q (= f_L(e_p)) \in E_{L_q} \backslash E_{L_q^I}$ tel que $f_L^q(v_q) = v$. L'application de q à G selon m_L^q entraîne donc la suppression de e.

Supposons à présent que l'application de q à G selon m_L^q entraîne la suppression d'un arc e. Si la suppression est implicite (i.e., e devenant suspendu), l'équivalence des suppressions de nœuds implique immédiatement que e est également supprimé par l'application de p selon m_L^p. Si la suppression est explicite, cela signifie que $\exists e_q = (v_1, v_2) \in E_{L_q} \backslash E_{L_q^I}$ tel que $f_L^q(e_q) = e$. D'après l'hypothèse 1.(e), on peut alors faire une disjonction de cas :

- Si $e_q \in f_L(E_{L_p} \backslash E_{L_p^I})$: alors $\exists e_p \in E_{L_p} \backslash E_{L_p^I}$ tel que $f_L(e_p) = e_q$. Or par construction de f_L^p, $f_L^p(e_p) = f_L^q(f_q)$ et donc $f_L^p(e_p) = e$. L'application de p à G selon m_L^p entraîne donc la suppression de e.
- Si $\exists v_q \in \{v_1, v_2)\}$ tel que $v_q \in V_{L_q} \backslash V_{L_q^I}$: l'extrémité $f_L^q(v_q)$ de e est supprimée par l'application de q selon m_L^q. Comme il y a équivalence des suppression de nœud, cette extrémité, et donc l'arc e, sont également supprimés par l'application de p selon m_L^p.

Les deux transformations considérées suppriment donc les mêmes arcs de G.

En considérant les hypothèses du théorème, il est immédiat que son application est équivalente à celle de q selon m_L^q (p et q supprimant et ajoutant les mêmes éléments). D'où $q\_m_L^q(G) = p\_m_L^p(G)$.

Équivalence des ajout d'éléments. Supposons que l'application de p à G selon m_L^p entraîne l'ajout d'un élément el. Par conséquent, $el \in R_p \backslash f_p(L_p)$ et d'après 2.(c) $f_R(el) \in R_q \backslash f_q(L_q)$. L'application de q à G selon m_L^q entraîne donc l'ajout d'un élément $f_R(el)$.

Réciproquement, supposons que l'application de q à G selon m_L^q entraîne l'ajout d'un élément el. Par conséquent, $el \in R_q \backslash f_q(L_q)$. D'après l'hypothèse 2.(c), $\exists el_p \in R_p \backslash f_p(L_p)$ tel que $f_R(el_p) = el$. Comme f_R est injective, el_p est unique. Notons le f_R^{-1}. L'application de p à G selon m_L^p entraîne donc l'ajout d'un élément $f_R^{-1}(el)$.

Cette équivalence est vraie à f_R près. Notons que si l'élément ajouté est un arc dont au moins une extrémité est invariante pour l'application des règles, $f_L^p = f_L^q \circ f_L$ et la condition 2.(b) garantit que l'extrémité est effectivement la même dans les deux cas.

Les règles p et q se conforment donc à la définition 4.7, et q est une spécialisation de p.

□

Bibliographie

[1] A. Simalatsar, Liangpeng Guo, M. Bozga, and R. Passerone. Integration of correct-by-construction bip models into the metroii design space exploration flow. In *30th IEEE International Conference on Computer Design (ICCD)*, pages 490–491, 2012.

[2] F. Oquendo. Dynamic software architectures : Formally modelling structure and behaviour with Pi-ADL. In *"The Third International Conference on Software Engineering Advances"*, pages 352–359, Oct 2008.

[3] Mohamed Hadj Kacem, Mohamed Jmaiel, Ahmed Hadj Kacem, and Khalil Drira. Evaluation and comparison of ADL based approaches for the description of dynamic of software architectures. In *7th International Conference on Enterprise Information Systems (ICEIS)*, pages 189–195, Miami, USA, 2005.

[4] Jeremy S. Bradbury, James R. Cordy, Juergen Dingel, and Michel Wermelinger. A survey of self-management in dynamic software architecture specifications. In *ACM SIGSOFT workshop on Self-managed systems (WOSS)*, pages 28–33, New York, NY, USA, 2004. ACM.

[5] Francisco de la Parra and Thomas Dean. Survey of graph rewriting applied to model transformations. In *2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 431–441, Jan 2014.

[6] Mohammed Karim Guennoun. *Architectures Dynamiques dans le cadre des applications à base de composants et orientées services*. PhD thesis, université Toulouse III-Paul Sabatier, 2007.

[7] Paul Horn. Autonomic Computing : IBM's Perspective on the State of Information Technology. Technical report, IBM, 2001.

[8] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, Jan 2003.

[9] D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mohindra, D. G. Shea, and M. Vanover. Autonomic personal computing. *IBM Systems Journal*, 42(1) :165–176, January 2003.

[10] A. Berns and S. Ghosh. Dissecting self-* properties. In *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 10–19, Sept 2009.

[11] Aaron Computing. An architectural blueprint for autonomic computing. Technical report, IBM, 2006.

[12] ISO/IEC/IEEE Standards. Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010 :2011(E) (Revision of ISO/IEC 42010 :2007 and IEEE Std 1471-2000)*, pages 1–46, Dec 2011.

[13] IEE Standards. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE Std 1471-2000*, pages i–23, 2000.

[14] Ron Morrison, Dharini Balasubramaniam, Flavio Oquendo, Brian Warboys, and R. Mark Greenwood. An active architecture approach to dynamic systems co-evolution. In Flavio Oquendo, editor, *Software Architecture*, volume 4758 of *Lecture Notes in Computer Science*, pages 2–10. Springer Berlin Heidelberg, 2007.

[15] David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[16] Lukman Ab. Rahim and Jon Whittle. A survey of approaches for verifying model transformations. *Software & Systems Modeling*, pages 1–26, 2013.

[17] M. Baleani, A. Ferrari, L. Mangeruca, A.L. Sangiovanni-Vincentelli, U. Freund, E. Schlenker, and H.-J. Wolff. Correct-by-construction transformations across design environments for model-based embedded software development. In *Design, Automation and Test in Europe*, pages 1044–1049 Vol. 2, March 2005.

[18] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis. Automated conflict-free distributed implementation of component-based models. In *International Symposium on Industrial Embedded Systems (SIES)*, pages 108–117, July 2010.

[19] M. Tounsi, M. Mosbah, and D. Mery. From event-b specifications to programs for distributed algorithms. In *IEEE 22nd International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*, pages 104–109, June 2013.

[20] Flavio Oquendo. Formally modelling software architectures with the UML 2.0 profile for π-ADL. *SIGSOFT Softw. Eng. Notes*, 31(1) :1–13, January 2006.

[21] Rémi Sharrock, Thierry Monteil, Patricia Stolf, Daniel Hagimont, and Laurent Broto. Non-Intrusive Autonomic Approach with Self-Management Policies Applied to Legacy Infrastructures for Performance Improvements. *Int. J. Adapt. Resilient Auton. Syst.*, 2(1) :58–76, January 2011.

[22] Dan Hirsch and Ugo Montanari. Consistent transformations for software architecture styles of distributed systems. *Electronic Notes in Theoretical Computer Science*, 28(0) :4–25, 2000.

[23] Christian Percebois, Martin Strecker, and Hanh Nhi Tran. Rule-level verification of graph transformations for invariants based on edges' transitive closure. In Robert Hierons, Mercedes Merayo, and Mario Bravetti, editors, *Software Engineering and Formal Methods*, volume 8137 of *Lecture Notes in Computer Science*, pages 106–121. Springer Berlin Heidelberg, 2013.

[24] Flavio Oquendo. π-ARL : an architecture refinement language for formally modelling the stepwise refinement of software architectures. *SIGSOFT Softw. Eng. Notes*, 29(5) :1–20, September 2004.

[25] Cédric Eichler, Thierry Monteil, Patricia Stolf, Alfredo L. Grieco, and Khalil Drira. Enhanced graph rewriting systems for complex software domains. *Software & Systems Modeling*, pages 1–21, 2014. doi : 10.1007/s10270-014-0433-1.

[26] Remi Sharrock. *Gestion autonomique de performance, d'énergie et de qualité de service. Application aux réseaux filaires, réseaux de capteurs et grilles de calcul.* PhD thesis, Institut National Polytechnique de Toulouse - INPT, December 2010.

[27] Grabriele Taentzer. *AGG : A Graph Transformation Environment for Modeling and Validation of Software*, pages 446–453. Volume 3062 of Pfaltz et al. [143], 2004.

[28] Arend Rensing. *The GROOVE Simulator : A Tool for State Space Generation*, pages 479–485. Volume 3062 of Pfaltz et al. [143], 2004.

[29] Alban Tiberghien, Philippe Merle, and Lionel Seinturier. Specifying Self-configurable Component-Based Systems with FracToy. In *Second International Conference on Abstract State Machines, Alloy, B and Z*, pages 91–104, 2010.

[30] Nassima Izerrouken, Xavier Thirioux, Marc Pantel, and Martin Strecker. Certifying an automated code generator using formal tools : Preliminary experiments in the geneauto project. In *European Congress on Embedded Real-Time Software (ERTS)*, volume 29, 2008.

[31] P.E.S. Barbosa, F. Ramalho, J.C.A. de Figueiredo, and A.D. dos S.Junior. An extended MDA architecture for ensuring semantics-preserving transformations. In *Software Engineering Workshop, 2008. SEW '08. 32nd Annual IEEE*, pages 33–42, Oct 2008.

[32] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information preserving bidirectional model transformations. In Matthew B. Dwyer and Antónia Lopes, editors, *Fundamental Approaches to Software Engineering*, volume 4422 of *Lecture Notes in Computer Science*, pages 72–86. Springer Berlin Heidelberg, 2007.

[33] Prashant Saxena, Noel Menezes, Pasquale Cocchini, and Desmond A. Kirkpatrick. The scaling challenge : can correct-by-construction design help ? In *International Symposium on Physical Design (ISPD)*, pages 51–58, New York, NY, USA, 2003. ACM. ISBN 1-58113-650-1.

[34] F. Ciccozzi and M. Sjodin. Enhancing the generation of correct-by-construction code from design models for complex embedded systems. In *IEEE 17th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4, Sept 2012.

[35] D. Bresolin, L. Di Guglielmo, L. Geretti, and T. Villa. Correct-by-construction code generation from hybrid automata specification. In *7th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1660–1665, July 2011.

[36] Iman Poernomo and Jeffrey Terrell. Correct-by-construction model transformations from partially ordered specifications in coq. In JinSong Dong and Huibiao Zhu, editors, *Formal Methods and Software Engineering*, volume 6447 of *Lecture Notes in Computer Science*, pages 56–73. Springer Berlin Heidelberg, 2010.

[37] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1) :70–93, January 2000.

[38] Lijun Lun and Xin Chi. Relationship between testing criteria for architecture configuration testing based on wright specification. In *International Conference on Computational Intelligence and Software Engineering (CiSE)*, pages 1–4, 2010.

[39] Mei Rong, Changlin Liu, and Guangquan Zhang. Modeling aspect-oriented software architecture based on ACME. In *6th International Conference on Computer Science Education (ICCSE)*, pages 1159–1164, 2011.

[40] Bhiri Mohamed Tahar, Sakka Rouis Taoufik, and Kmimech Mourad. Checking non-functional properties of UML2.0 components assembly. In *22nd IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*, pages 278–283, 2013.

[41] David Wile. Using dynamic ACME. In *Working Conference on Complex and Dynamic Systems Architecture*, 2001.

[42] David Garlan, Robert T. Monroe, and David Wile. ACME : Architectural description of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.

[43] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In *4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 3–14, 1996.

[44] Robert Allen, Rémi Douence, and David Garlan. Specifying and analyzing dynamic software architectures. In *Fundamental Approaches to Software Engineering*, volume 1382 of *Lecture Notes in Computer Science*, pages 21–37. Springer Berlin Heidelberg, 1998.

[45] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3) :pp 213–249, July 1997.

[46] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. Specification and analysis of system architecture using rapide. *IEEE Trans. Software Eng.*, 21(4) :336–355, 1995.

[47] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quema, and Jean-Bernard Stefani. An open component model and its support in java. In Ivica Crnkovic, Judith A. Stafford, Heinz W. Schmidt, and Kurt Wallnau, editors, *Component-Based Software Engineering*, volume 3054 of *Lecture Notes in Computer Science*, pages 7–22. Springer Berlin Heidelberg, 2004.

[48] Nicolas Pessemier, Lionel Seinturier, Thierry Coupaye, and Laurence Duchien. A model for developing component-based and aspect-oriented systems. In Welf Löwe and Mario Südholt, editors, *Software Composition*, volume 4089 of *Lecture Notes in Computer Science*, pages 259–274. Springer Berlin Heidelberg, 2006.

[49] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java : Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11-12) :1257–1284, 2006.

[50] Sylvain Sicard, Fabienne Boyer, and Noel De Palma. Using components for architecture-based management : The self-repair case. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 101–110, New York, NY, USA, 2008. ACM.

[51] Noël de Palma, Sara Bouchenak, Fabienne Boyer, Daniel Hagimont, Sylvain Sicard, and Christophe Taton. Jade : Un Environnement d'Administration Autonome. *Techniques et Sciences Informatiques*, 2008.

[52] Laurent Broto, Patricia Stolf, Jean-Paul Bahsoun, Daniel Hagimont, and Noel Depalma. Spécification de politiques d'administration autonome avec Tune. In *Conférence Française sur les Systèmes d'Exploitation (CFSE)*. Chapitre Français de l'ACM SIGOPS, février 2008.

[53] Laurent Broto, Daniel Hagimont, Patricia Stolf, Noel De Palma, and Suzy Temate. Autonomic management policy specification in tune. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1658–1663, New York, NY, USA, 2008. ACM.

[54] Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little. *Documenting Software Architectures : Views and Beyond*. Pearson Education, 2002.

[55] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods : Practice and experience. *ACM Comput. Surv.*, 41(4) :19 :1–19 :36, October 2009.

[56] Flavio Oquendo. Pi-ADL : an architecture description language based on the higher-order typed pi-calculus for specifying dynamic and mobile software architectures. *SIGSOFT Softw. Eng. Notes*, pages 1–14, 2004.

[57] Radu Mateescu and Flavio Oquendo. π-AAL : an architecture analysis language for formally specifying and verifying structural and behavioural properties of software architectures. *SIGSOFT Softw. Eng. Notes*, 31(2) :1–19, March 2006. ISSN 0163-5948.

[58] Petri Selonen and Jianli Xu. Validating UML models against architectural profiles. *SIGSOFT Software Engineering Notes*, 28 :pp.58–67, September 2003.

[59] I. Loulou, A. Hadj Kacem, M. Jmaiel, and K. Drira. Towards a unified graph-based framework for dynamic component-based architectures description in Z. In *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS)*, pages 227–234, July 2004.

[60] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. Modeling software architectures in the Unified Modeling Language. *ACM Trans. Softw. Eng. Methodol.*, 11 :pp.2–57, January 2002.

[61] Mohamed Mancona Kandé and Alfred Strohmeier. Towards a UML profile for software architecture descriptions. In *3rd International Conference on The Unified Modeling Language : advancing the standard (UML)*, pages 513–527, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 3-540-41133-X.

[62] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. Style-Based Modeling and Refinement of Service-Oriented Architectures. *Journal of Software and Systems Modelling*, 5(2) : 187–207, June 2006.

[63] Cedric Eichler, Ghada Gharbi, Nawal Guermouche, Thierry Monteil, and Patricia Stolf. Graph-Based Formalism for Machine-to-Machine Self-Managed Communications. In *IEEE 22nd International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*, pages 74–79, 2013.

[64] Ismael Bouassida Rodriguez, Khalil Drira, Christophe Chassot, Karim Guennoun, and Mohamed Jmaiel. A rule-driven approach for architectural self adaptation in collaborative activities using graph grammars. *International Journal of Autonomic Computing*, 1(3/2010) :226–245, 2010.

[65] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1 : Foundations. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[66] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2 : Applications, Languages, and Tools. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[67] Andy Schürr, Dániel Varró, and Gergely Varró, editors. *Applications of Graph Transformations with Industrial Relevance*, volume 7233 of *Programming and Software Engineering*. Springer-Verlag Berlin Heidelberg, 2012.

[68] Andy Schürr, Manfred Nagl, and Albert Zündorf, editors. *Applications of Graph Transformations with Industrial Relevance*, volume 5088 of *Programming and Software Engineering*. Springer-Verlag Berlin Heidelberg, 2008.

[69] Steve Awodey. *Category Theory*, volume 49 of *Oxford Logic Guides*. Oxford University Press, 2006.

[70] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Handbook of Graph Grammars and Computing by Graph Transformations*, chapter Algebraic Approaches to Graph Transformation. Part I : Basic Concepts and Double Pushout Approach, pages 163–245. Volume 1 : Foundations of Rozenberg [65], 1997.

[71] Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109(1–2) :181 – 224, 1993.

[72] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. *Handbook of Graph Grammars and Computing by Graph Transformations*, chapter Algebraic Approaches to Graph Transformation. Part II : Single Pushout Approach and Comparison with Double Pushout Approach, pages 247–312. Volume 1 : Foundations of Rozenberg [65], 1997.

[73] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae*, 26 :287–313, 1995.

[74] Annegret Habel and Karl-Heinz Pennemann. Nested Constraints and Application Conditions for High-Level Structures. In Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin Heidelberg, 2005.

[75] J. Engelfriet and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations*, chapter Node Replacement Graph Grammars, pages 1–94. Volume 1 : Foundations of Rozenberg [65], 1997.

[76] Ismael Bouassida. *Gestion dynamique des architectures pour les systèmes communicants collaboratifs.* PhD thesis, Ecole nationale d'ingénieurs de Sfax, 2011.

[77] Michel Wermelinger and José Luiz Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Science of Computer Programming*, 44(2) :133–155, 2002. Special Issue on Applications of Graph Transformations (GRATRA 2000).

[78] Daniel Le Métayer. Software Architecture Styles As Graph Grammars. *SIGSOFT Softw. Eng. Notes*, 21(6) :15–23, October 1996.

[79] Dan Hirsch, Paola Inverardi, and Ugo Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In Patrick Donohoe, editor, *Software Architecture*, volume 12 of *The International Federation for Information Processing*, pages 127–143. Springer US, 1999.

[80] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundam. Inf.*, 26(3-4) :241–265, June 1996.

[81] N. Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3) :113–124, 1956.

[82] Hartmut Ehrig, Hans-Jörg Kreowski, Grzegorz Rozenberg, editor. *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1991.

[83] Gregor Engels, Claus Lewerentz, and Wilhelm Schäfer. Graph grammar engineering : A software specification method. In Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld, editors, *Proceedings of the 2nd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 186–201. Springer Berlin Heidelberg, 1987.

[84] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundam. Inf.*, 26(3,4) :241–265, December 1996.

[85] Roberto Bruni, Antonio Bucchiarone, Stefania Gnesi, and Hernán Melgratti. Modelling dynamic software architectures using typed graph grammars. *Electronic Notes in Theoretical Computer Science*, 213(1) :39 – 53, 2008.

[86] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. *From graph grammars to high level replacement systems*, pages 269–291. Volume 532 of Hartmut Ehrig, Hans-Jörg Kreowski, Grzegorz Rozenberg [82], 1991.

[87] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graphs and graph transformation based on adhesive hlr categories. *Fundamenta Informaticae*, 74(1) :pp. 31–61, October 2006.

[88] Hartmut Ehrig and Bernd Mahr. *Fundamentals of algebraic specification*. EATCS Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, New York, 1985.

[89] Wolfgang Hesse. Two-level graph grammars. In Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 255–269. Springer Berlin Heidelberg, 1979.

[90] Detlef Plump and Sandra Steinert. Towards Graph Programs for Graph Algorithms. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 128–143. Springer Berlin Heidelberg, 2004.

[91] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental Theory for Typed Attributed Graph Transformation. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 161–177. Springer Berlin Heidelberg, 2004.

[92] Berthold Hoffmann. Graph Transformation with Variables. In Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 101–115. Springer Berlin Heidelberg, 2005.

[93] Ulrich Nickel, Jörg Niere, and Albert Zündorf. The FUJABA environment. In *22nd International Conference on Software Engineering*, pages 742–745, New York, NY, USA, 2000. ACM.

[94] Sven Burmester, Holger Giese, Jörg Niere, Matthias Tichy, Jörg P. Wadsack, Robert Wagner, Lothar Wendehals, and Albert Zündorf. Tool integration at the meta-model level : the fujaba approach. *International Journal on Software Tools for Technology Transfer*, 6(3) :203–218, 2004.

[95] Juande Lara and Hans Vangheluwe. Atom3 : A tool for multi-formalism and meta-modelling. In Ralf-Detlef Kutsche and Herbert Weber, editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2002.

[96] Juan de Lara, Hans Vangheluwe, and Manuel Alfonseca. Meta-modelling and graph grammars for multi-paradigm modelling in atom3. *Software and Systems Modeling*, 3(3) :194–209, 2004.

[97] Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam Szalkowski. GrGen : A Fast SPO-Based Graph Rewriting Tool. In Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 4178 of *Lecture Notes in Computer Science*, pages 383–397. Springer Berlin Heidelberg, 2006.

[98] D. Plumb. The Design of GP 2. In *International Workshop on Reduction Strategies in Rewriting and Programming*, (WRS), pages 1–16, 2012.

[99] Sergio Segura, David Benavides, Antonio Ruiz-Cortés, and Pablo Trinidad. Automated merging of feature models using graph transformations. In Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering II*, volume 5235 of *Lecture Notes in Computer Science*, pages 489–505. Springer Berlin Heidelberg, 2008.

[100] Michael Rudolf. Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching. In *6th International Workshop on Theory and Application of Graph Transformatons (TAGT)*, pages 238–251. Springer, 1998.

[101] Mohamed Amine Hannachi, Ismael Bouassida Rodriguez, Khalil Drira, and Saul Eduardo Pomares Hernandez. Gmte : A tool for graph transformation and exact/inexact graph matching. In Walter G. Kropatsch, NicoleM. Artner, Yll Haxhimusa, and Xiaoyi Jiang, editors, *Graph-Based Representations in Pattern Recognition*, volume 7877 of *Lecture Notes in Computer Science*, pages 71–80. Springer Berlin Heidelberg, 2013.

[102] H. Khlif, H. Hadj Kacem, S.E. Pomares Hernandez, C. Eichler, A. Hadj Kacem, and A. Calixto Simon. A graph transformation-based approach for the validation of checkpointing algorithms in distributed systems. In *IEEE 23rd International WETICE Conference (WETICE)*, pages 80–85, June 2014.

[103] Aymen Kamoun, Saïd Tazi, and Khalil Drira. Fadyrcos, a semantic interoperability framework for collaborative model-based dynamic reconfiguration of networked services. *Computers in Industry*, 63(8) :756 – 765, 2012. Special Issue on Sustainable Interoperability : The Future of Internet Based Industrial Enterprises.

[104] Aeiman Gadafi, Daniel Hagimont, Laurent Broto, Remi Sharrock, Alain Tchana, and Noel De Palma. Energy-QoS tradeoffs in J2EE hosting centres. *Int. J. Autonomic Comput.*, 2(1) :54–72, February 2014.

[105] David Luckham. *The Power of Events : An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, 1st edition, 2002.

[106] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. Using complex event processing for dynamic business process adaptation. In *IEEE International Conference on Services Computing*, pages 466–473, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[107] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora : a new model and architecture for data stream management. *The International Journal on Very Large Data Bases*, 12(2) : 120–139, August 2003.

[108] A. Arasu, B. Babcock, Shivnath Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM : The Stanford Data Stream Management System. Technical Report 2004-20, Stanford InfoLab, 2004.

[109] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu, and Zheng Zhang. TimeStream : Reliable Stream Computation in the Cloud. In *Proceedings of the 8th ACM European Conference on Computer Systems*, page 1, New York, New York, USA, 2013. ACM Press.

[110] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, and Patrick Valduriez. StreamCloud : A Large Scale Data Streaming System. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 126–137. Ieee, 2010.

[111] Gianpaolo Cugola and Alessandro Margara. Processing flows of information : from data stream to complex event processing. *ACM Computing Surveys*, 44(3) :1–62, June 2012.

[112] Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, and Lionel Seinturier. A middleware platform to federate complex event processing. In *17th IEEE International Enterprise Distributed Object Computing Conference*, pages 113–122, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[113] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language : semantic foundations and query execution. *The VLDB Journal*, 15(2) :121–142, July 2005.

[114] Namit Jain, Shailendra Mishra, Anand Srinivasan, Johannes Gehrke, Jennifer Widom, Hari Balakrishnan, Uvgur Çetintemel, Mitch Cherniack, Richard Tibbetts, and Stan Zdonik. Towards a Streaming SQL Standard. *Proc. VLDB Endow.*, 1(2) :1379–1390, August 2008.

[115] Michael Eckert, François Bry, Simon Brodt, Olga Poppe, and Steffen Hausmann. A CEP Babelfish : Languages for Complex Event Processing and Querying Surveyed. In Sven Helmer, Alexandra Poulovassilis, and Fatos Xhafa, editors, *Reasoning in Event-Based Distributed Systems SE - 3*, volume 347 of *Studies in Computational Intelligence*, pages 47–70. Springer Berlin Heidelberg, 2011.

[116] Evangelia Kalyvianaki, Wolfram Wiesemann, Quang Hieu Vu, Daniel Kuhn, and Peter Pietzuch. SQPR : Stream query planning with reuse. In *2011 IEEE 27th International Conference on Data Engineering*, pages 840–851. IEEE, April 2011.

[117] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Er Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the Borealis stream processing engine. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, pages 277–289, 2005.

[118] Stratis D Viglas, Jeffrey F Naughton, and Josef Burger. Maximizing the Output Rate of Multiway Join Queries over Streaming Information Sources. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, volume 29 of *VLDB '03*, pages 285–296. VLDB Endowment, 2003.

[119] W.A. Higashino, C. Eichler, M.A.M. Capretz, T. Monteil, M.B.F. De Toledo, and P. Stolf. Query analyzer and manager for complex event processing as a service. In *IEEE 23rd International WETICE Conference (WETICE)*, pages 107–109, June 2014.

[120] D. M. Chess, C. Palmer, and S. R. White. Security in an Autonomic Computing Environment. *IBM Syst. J.*, 42(1) :107–118, January 2003.

[121] E. Caron and F. Desprez. DIET : A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 20(3) :pp. 335–352, 2006.

[122] Dominique Duval, Rachid Echahed, and Frédéric Prost. Categorical Abstract Rewriting Systems and Functoriality of Graph Transformation. *ECEASST*, 41, 2011.

[123] Arend Rensink. Compositionality in Graph Transformation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 309–320. Springer Berlin Heidelberg, 2010.

[124] András Balogh and Dániel Varró. Pattern composition in graph transformation rules. In *European Workshop on Composition of Model Transformations*, Bilbao, Spain, 2006.

[125] D. Boswarthick, O. Elloumi, and O. Hersent, editors. *M2M Communications : A Systems Approach*. Wiley, 2012.

[126] S. Pandey, Mi-Jung Choi, Myung-Sup Kim, and J.W. Hong. Towards management of machine to machine networks. In *13th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–7, Sept 2011.

[127] ETSI M2M group. ETSI Technical Specification 102 689 Machine-to-Machine communications (M2M) ; M2M service requirements. Technical report, ETSI, Sophia-Antipolis, France, 2010.

[128] ETSI M2M group. ETSI Technical Specification 102 690 Machine-to-Machine communications (M2M) ; Functional architecture. Technical report, ETSI, Sophia-Antipolis, France, 2011.

[129] ETSI M2M group. ETSI Technical Specification 102 921, Machine-to-Machine communications (M2M) ; mIa, dIa and mId interfaces. Technical report, ETSI, Sophia-Antipolis, France, 2012.

[130] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. OM2M : Extensible ETSI-compliant M2M Service Platform with Self-configuration Capability. *Procedia Computer Science*, 32(0) :1079 – 1086, 2014.

[131] Sunghwan Roh, Kyungrae Kim, and Taewoong Jeon. Architecture Modeling Language Based on UML2.0. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC)*, pages 663–669, Washington, DC, USA, 2004. IEEE Computer Society.

[132] ETSI M2M group. ETSI Technical Report 102 691, Machine-to-Machine communications (M2M) ; Smart Metering Use-Cases. Technical report, ETSI, Sophia-Antipolis, France, 2010.

[133] C Eichler, G. Gharbi, T. Monteil, N. Guermouche, and P. Stolf. Self-management of machine-to-machine communications : a multi-models approach. *International Journal of Autonomous and Adaptive Communications System (IJAACS)*, à paraître.

[134] Damien Borgetto, Henri Casanova, Georges Da Costa, and Jean-Marc Pierson. Energy-aware service allocation. *Future Gener. Comput. Syst.*, 28(5) :769–779, may 2012.

[135] Terence Parr and Kathleen Fisher. Ll(*) : the foundation of the antlr parser generator. *SIGPLAN Not.*, 47(6) :pp.425–436, June 2011.

[136] Carlos Parra, Daniel Romero, Sébastien Mosser, Romain Rouvoy, Laurence Duchien, and Lionel Seinturier. Using constraint-based optimization and variability to support continuous self-adaptation. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 486–491, New York, NY, USA, 2012. ACM.

[137] Stephen Cole Kleene. On notation for ordinal number. *The Journal of Symbolic Logic*, 3(4) : 150–155, Dec. 1938.

[138] Stephen Cole Kleene. *Introduction to metamathematics*, volume 1 of *Bibliotheca mathematica*. North-Holland, Amsterdam, 1952.

[139] Houda Khlif, Hatem Hadj Kacem, Saúl E. Pomares Hernández, Cédric Eichler, Ahmed Hadj Kacem, and Alberto Calixto Simon. A graph transformation-based approach for the validation of checkpointing algorithms in distributed systems. In *IEEE 23rd International WETICE Conference*, pages 80–85, June 2014.

[140] D. Borgetto and P. Stolf. An energy efficient approach to virtual machines management in cloud computing. In *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 229–235, Oct 2014.

[141] M Ben Alaya and T Monteil. Frameself : An ontology-based framework for the self-management of m2m systems. *Concurr. Comput. Pract. Exp*, 18, 2013.

[142] D. Borgetto, R. Chakode, B. Depardon, C. Eichler, Garcia J.M., H. Hbaieb, T. Monteil, E. Pelorce, Rachdi A., A. Sheikh, and P. Stolf. Hybrid approach for energy aware management of multi-cloud architecture integrating user machines. *Journal of Grid Computing*, à paraître.

[143] John L. Pfaltz, Manfred Nagl, and Boris Böhlen, editors. *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.