Faculté des Sciences et Techniques *M1 Informatique*
Université du Maine

MÉMOIRE

présenté en vue d'obtenir

Le Diplôme De Master I Informatique

en

CHM

BOUAZIZ Mohamed

RECONFIGURATION DYNAMIQUE DES ARCHITECTURES, SENSIBLE AUX PROPRIÉTÉS TEMPORISÉES

ORGANISME : LAAS-CNRS

Code : *Master 1 12 6*

Faculté des Sciences et Techniques – Avenue Olivier Messiaen - 72085 LE MANS Cedex 9
Tel : 02 43.83.38.38 Fax : 02 43.83 38.48

Faculté des Sciences et Techniques *M1 Informatique*
Université du Maine

MÉMOIRE

présenté en vue d'obtenir

Le Diplôme De Master I Informatique

en

CHM

BOUAZIZ Mohamed

RECONFIGURATION DYNAMIQUE DES ARCHITECTURES, SENSIBLE AUX PROPRIÉTÉS TEMPORISÉES

ORGANISME : LAAS-CNRS

Responsable du stage : Madame GUERMOUCHE Nawal

Parrain enseignant : Monsieur LEMEUNIER Thierry

RÉSUMÉ :

Le sujet de ce stage se situe dans le cadre des architectures distribuées des systèmes orientés services. Étant donné la dynamicité des services ainsi que leur environnement d'exécution, il est nécessaire de s'appuyer sur des techniques et des outils de gestion de la composition, notamment des techniques de reconfiguration. Le travail abordé dans ce mémoire constitue une extension d'un méta-modèle de reconfiguration des architectures dynamiques, basée sur des concepts théoriques, telles que les grammaires de graphes, afin de prendre en compte les propriétés temporisées. La reconfiguration temporisée permet d'assurer la stabilité du système et de respecter les contraintes temporelles relatives aux applications distribuées temps réel.

Mots clés (5) :

Architectures orientées services, adaptation structurelle, reconfiguration temporisée, grammaires de graphes, transformation de graphes

Code : *Master 1 12 6*

Dédicace

Je dédie ce travail à mes êtres les plus chers, en guise de reconnaissance et de gratitude pour leur amour, leur confiance, et tous les sacrifices qu'ils ont faits pour moi :

A

mes chers parents,

A

toute ma famille,

A

tous mes amis,

et à toute personne qui m'a soutenu
et soulagé de près ou de loin.

Je vous exprime à travers ce mémoire tout l'amour
et le respect que je vous dois.

Remerciements

*Je tiens à exprimer ma gratitude et mes vifs remerciements à mon responsable de stage, Madame **Nawal Guermouche**, pour sa grande qualité d'encadrement, sa disponibilité et ses conseils fort judicieux. Ses qualités humaines et scientifiques ont largement contribué au bon déroulement de ce stage.*

*Je remercie vivement Monsieur **Khalil Drira** pour m'avoir accepté dans son équipe et aidé.*

*Mes remerciements s'adressent également à mon enseignant référent, Monsieur **Thierry Lemeunier**, pour l'honneur qu'il m'a offert en suivant mon avancement durant mon stage et en évaluant ce mémoire.*

*Je tiens à remercier Monsieur **Jérôme LEHUEN**, pour avoir accepté de participer comme examinateur au jury de la soutenance de ce stage.*

Enfin, je remercie tous ceux qui ont participé à l'élaboration de ce mémoire, qu'ils trouvent ici l'expression de ma profonde gratitude.

Table des matières

Table des figures

Introduction générale

Ce mémoire présente les travaux réalisés dans le cadre de mon stage qui se situent dans le domaine des architectures distribuées des systèmes orientés services. Ces systèmes peuvent subir des changements fréquents aussi bien au niveau du comportement de leurs composants que dans leur composition structurelle. Ces mutations peuvent provenir de certaines défaillances relatives à la qualité de service ainsi que de l'évolution des besoins requis par le système concerné.

Nous nous intéressons précisément, dans le cadre de ces travaux, à l'adaptation structurelle des systèmes distribuées dynamiques. À titre d'exemple, la non-disponibilité d'un service impliqué dans une composition peut causer le dysfonctionnement de la composition globale. Comme solution alternative, on pourrait envisager de remplacer ce service par un autre ou par une composition de services. L'objectif primordial de l'auto-reconfiguration de l'architecture de ces systèmes est ainsi d'améliorer leur fiabilité et la sûreté de leur fonctionnement, en offrant des services de plus en plus autonomes qui s'adaptent aux différentes mutations qui peuvent survenir.

Afin de modéliser ce type d'architectures, les graphes représentent un outil très expressif s'agissant de la description statique des systèmes coopératifs quelle que soit la complexité de leur structure. Quant à l'évolution dynamique de l'architecture de ces systèmes, les grammaires de graphes sont considérés comme l'un des outils théoriques les plus puissants, permettant de spécifier aussi bien la construction des structures architecturales que leur évolution au cours du temps.

Dans ce contexte, les travaux abordés dans [Guennoun, 2007] et [Bouassida et al., 2008] offrent un méta-modèle complet traitant de l'évolution dynamique de l'architecture des systèmes coopératifs, ainsi que l'application des concepts théoriques définis à travers un cas d'étude concret, à savoir le système des Opérations d'interventions d'urgence (OIU). L'auto-reconfiguration des architectures, évoluant dans le temps, représente ainsi la problématique primordiale de ces travaux. Cependant, ce méta-modèle ne prend pas en considération les contraintes temporelles que peut avoir un système collaboratif. En effet, l'orchestration de la collaboration entre les composants d'un tel système peut dépendre de certaines contraintes relatives au temps, à savoir des délais nécessaires à l'installation et à la mise en marche de certains composants, un temps de vérification de l'état d'une connexion entre deux composants, ainsi qu'une fréquence maximale des passages d'une configuration à une autre. Ces contraintes ont pour but d'assurer la stabilité du système et d'échapper aux anomalies engendrées par les actions de reconfiguration pouvant avoir lieu à des instants considérés comme critiques pour ledit système.

Les travaux présentés dans ce mémoire abordent une extension du méta-modèle décrit par [Guennoun, 2007] et [Bouassida et al., 2008], prenant en compte les propriétés et les contraintes temporelles spécifiques à la reconfiguration des architectures dynamiques. Nous définissons ainsi un cadre formel traitant de la reconfiguration temporisée basée sur les concepts des grammaires de graphes. Nous projetons au fur et à mesure les nouveaux concepts abordés sur les besoins temporels requis au sein du système OIU.

Nous répartissons la description du travail réalisé au cours du stage sur trois chapitres. Le premier chapitre évoque l'état de l'art des différents travaux traitant des thématiques de base de notre contribution. Le deuxième chapitre présente une description formelle des différents concepts théoriques proposés s'agissant de la reconfiguration temporisée. Enfin, nous abordons dans le troisième chapitre une étude préliminaire pour une mise en application potentielle de la reconfiguration temporisée au sein d'un outil de transformation de graphes. La reprise de ce travail est envisagée dans la deuxième phase de mon stage, devant se dérouler aux mois de juillet et Août 2012.

Présentation du cadre

Les travaux présentés dans ce mémoire sont réalisés à l'issue de la première période de mon stage au sein du laboratoire de recherche « LAAS-CNRS », faisant partie du deuxième semestre de la première année de Master en Génie Informatique, à l'UFR des Sciences et Techniques du Mans, appartenant à l'université du Maine.

Le Laboratoire d'Analyse et d'Architecture des Systèmes LAAS-CNRS est une unité appartenant au CNRS (Centre National de la Recherche Scientifique) qui représente un organisme public de recherche placé sous la tutelle du Ministère de l'Enseignement Supérieur et de la Recherche. Il produit du savoir et met ce savoir au service de la société [Site 4].

Le CNRS, une des premières institutions scientifiques du monde [Site 3], mène des recherches dans l'ensemble des domaines scientifiques, technologiques et sociétaux. Il couvre la totalité de la palette des champs scientifiques, qu'il s'agisse des mathématiques, de la physique, des sciences et technologies de l'information et de la communication, de la physique nucléaire et des hautes énergies, des sciences de la planète et de l'Univers, de la chimie, des sciences du vivant, des sciences humaines et sociales, des sciences de l'environnement ou des sciences de l'ingénierie [Site 4].

Le LAAS-CNRS est rattaché à l'Institut des Sciences de l'Ingénierie et des Systèmes (INSIS) et à l'Institut des Sciences de l'Information et de leurs Interactions (INS2I). Situé à Toulouse, il est associé par convention aux six membres fondateurs du Pôle de Recherche et d'Enseignement Supérieur, « Université de Toulouse », à savoir :

- Université Paul Sabatier (UPS),
- Institut National des Sciences Appliquées de Toulouse (INSA),
- Institut National Polytechnique de Toulouse (INP),
- Institut Supérieur de l'Aéronautique et de l'Espace (ISAE, issu de la fusion ENSICA-SUPAERO),
- Université du Mirail (UTM), et
- Université Toulouse 1 Capitole (UT1) [Site 2].

La logique de recherche du LAAS-CNRS est de modéliser, de concevoir et de maîtriser les systèmes complexes, hétérogènes, en interaction avec d'autres systèmes ou avec l'Homme, dans une approche constructiviste et intégrative autour de ces objets de recherche. Les recherches menées par ce laboratoire concernent les sciences et technologies de l'information, de la communication et des systèmes dans huit thèmes scientifiques, à savoir :

- l'Informatique Critique,
- les Réseaux et Communications
- la Robotique,
- la Décision et l'Optimisation,
- la Conversion de l'énergie,
- les Micronanobiotechnologies,
- les Micronanosystèmes RF et optiques, et
- la Nanoingénierie et l'Intégration [Site 2].

Le LAAS-CNRS se compose de 21 équipes de recherche conduisant l'activité scientifique du laboratoire. Durant mon stage, j'étais accueilli par l'équipe de recherche SARA (Services et Architectures pour les Réseaux Avancés) dirigée par M. Khalil Drira. Les travaux du groupe SARA concernent précisément les réseaux et les systèmes de communication de nouvelle génération. Les études menées par cette équipe visent la maîtrise de leur conception, planification, gestion du déploiement, et supervision [Site 5] [Site 6].

Les travaux qui m'ont été demandés visent l'extension du méta-modèle décrit par [Guennoun, 2007] et [Bouassida et al., 2008] pour y intégrer la prise en compte des propriétés et des contraintes temporelles spécifiques à la reconfiguration des architectures dynamiques. Le déroulement des activités de ce stage a été planifié comme suit :

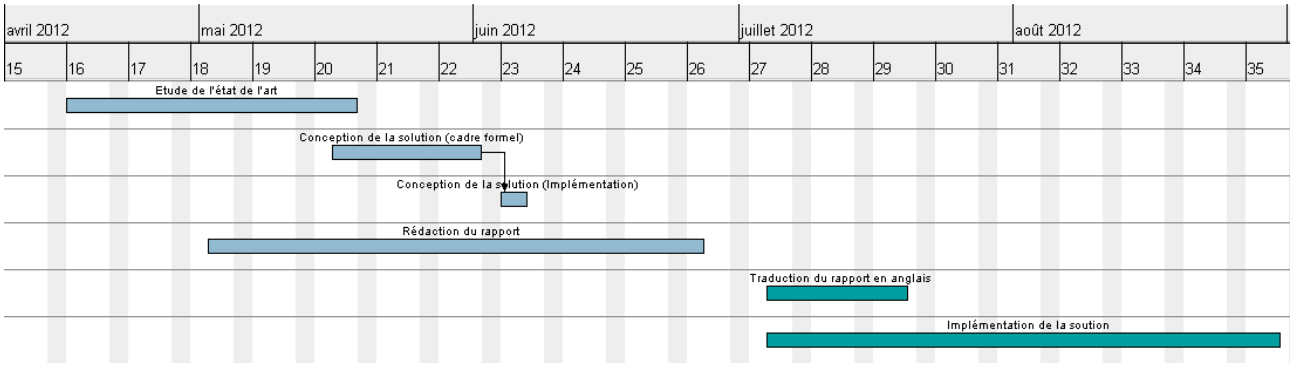


Figure 1 : Planning des activités durant le stage

Chapitre 1 : ***État de l'art***

Dans ce premier chapitre, nous allons, en premier lieu, faire un tour d'horizon sur les notions de base de la description des graphes et des grammaires de graphes. Nous évoquons ensuite des travaux antérieurs abordant certaines extensions sur lesdites grammaires en appliquant des combinaisons d'approches de transformation de graphes sur un cas d'études concret des architectures dynamiques.

Bien que l'extension des grammaires de graphes ait fait l'objet de nombreux travaux de recherche, aucune tentative n'a examiné l'extension de ces grammaires par les propriétés et/ou les contraintes temporisées. De ce fait, nous clôturons la présentation des travaux servant comme source d'inspiration de la contribution visée par notre projet, par des exemples de formalisation et d'application des aspects temporels dans le cadre d'autres domaines de recherche, à savoir, la logique temporelle et les automates temporisés.

1. Notions Fondamentales

1.1. Description des graphes

Un graphe est constitué d'un ensemble de points et d'un ensemble de liens reliant certains ensembles de points avec une cardinalité égale à 2. Les points d'un graphe sont nommés des « sommets » ou des « nœuds ». Cependant, les liens dépendent de la nature du graphe dans lequel ils se situent. Dans un graphe dit « orienté », les liens entre les sommets possèdent un sens et sont appelés ainsi des « arcs ». Dans un graphe dit « non orienté », le sens des liens n'a pas d'importance. Les liens sont donc appelés des « arêtes » [Eichler et al., 2012] [Guennoun, 2007].

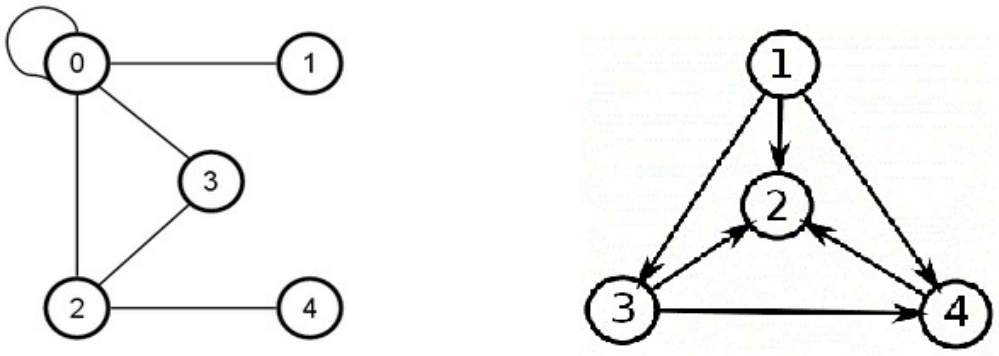


Figure 2 : Graphes orientés et graphes non orientés

Ces graphes sont décrits par un couple G = (N, A) où N représente l'ensemble des sommets du graphe et A représente l'ensemble des arcs si G est orienté ou bien l'ensemble des arêtes si G est non orienté.

- Le graphe non orienté de la figure 1.a est décrit comme suit :

  G = (N, A)

  N = {0, 1, 2, 3, 4}

  A = {{0,0}, {0,1}, {0,2}, {0,3}, {2,3}, {2,4}}

- Le graphe orienté de la figure 1.b est décrit comme suit :

  G = (N, A)

  N = {1, 2, 3, 4}

  A = {(1,2), (1,3), (1,4), (3,2), (3,4), (4,2)

Certains graphes permettent de satisfaire des spécifications plus riches. Les nœuds et les liens entre ces graphes peuvent comprendre plusieurs types. On pourra associer alors des valeurs, appelés des attributs, à ces nœuds et liens. Un graphe est alors dit attribué si ses nœuds et/ou ses liens contiennent des attributs. Les attributs qui concernent les nœuds sont appelés des labels et ceux qui concernent les liens sont appelés des étiquettes. Un graphe ayant des nœuds (respectivement des liens) avec des labels (respectivement des étiquettes) multiples est considéré comme un graphe multi-labellé (respectivement multi-étiqueté). Conformément à ses notions, nous présentons les différents types de graphe dans la figure 3.

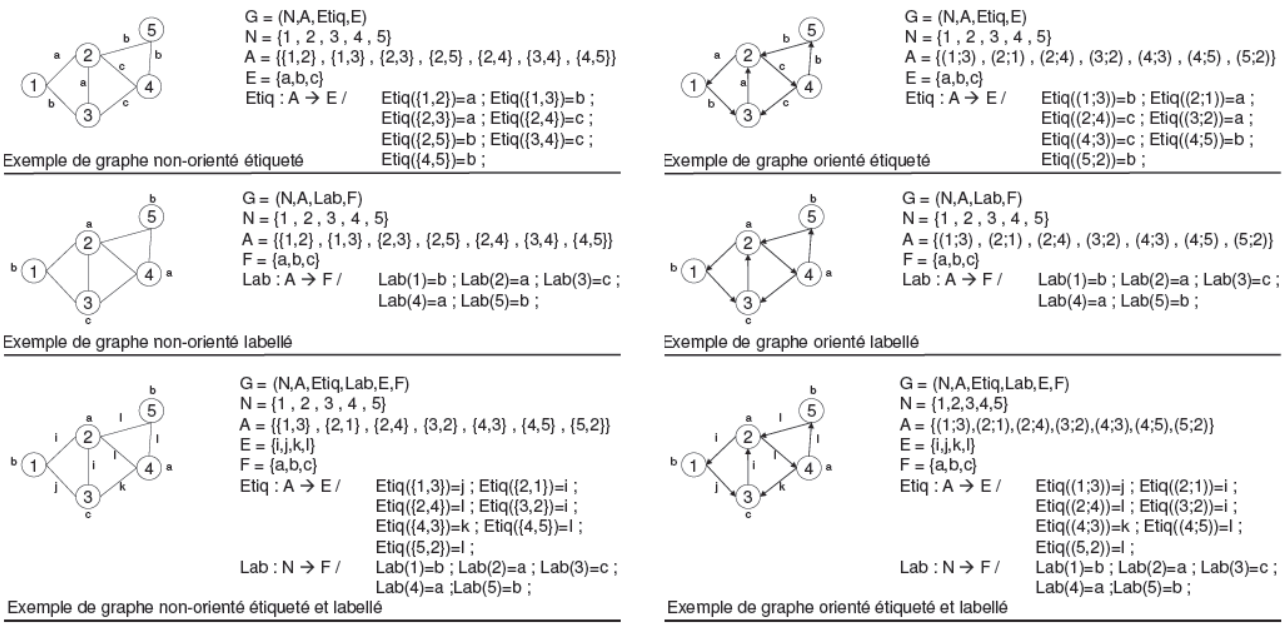


Figure 3 : Les différents types de graphes attribués [Guennoun, 2007]

De ce fait, la description formelle des graphes labellés est étendue par un ensemble F représentant l'ensemble des labels des nœuds et par une fonction Lab : N → F permettant d'associer à chaque nœud son label. De la même manière, dans le cas des graphes étiquetés, la valuation des arcs ou des arêtes est assurée par une fonction Etiq : A → E où E représente l'ensemble des étiquettes portées par les arcs ou par les arêtes.

1.2. Les grammaires de graphes

Dans des architectures complexes comportant un nombre important de composants ou de services ayant des interactions et/ou des interdépendances complexes, les graphes représentent une solution idéale pour modéliser les différentes instances d'une architecture [Eichler et al., 2012]. Une instance

architecturale constitue une configuration, un état et/ou une composition d'une architecture à un instant particulier.

Cependant, les architectures sont souvent très dynamiques avec des structures adoptant des changements fréquents, voire critiques. Les grammaires des graphes constituent un formalisme permettant de décrire l'aspect dynamique des architectures d'une manière très expressive.

Les grammaires de graphes sont décrites généralement à travers un système classique s'inspirant de la logique des « grammaires génératives » définies par Chomsky [Chomsky, 1956]. Une grammaire de graphes est constituée ainsi d'un quadruplet (AX, NT, T, P) où :

- AX : représente l'axiome, c'est à dire, le point de départ de la construction des graphes de la grammaire,
- NT : représente l'ensemble des nœuds dits, « non terminaux »,
- T : représente l'ensemble des nœuds « terminaux », et
- P : représente l'ensemble des productions de la grammaire assurant le passage d'un graphe à un autre en appliquant des changements structuraux, à savoir la suppression ou l'adjonction de certains composants dans le premier graphe. Ces productions sont appelées aussi des « règles de transformation » ou encore des « règles de réécriture » de graphes [Ehrig et al., 2006].

Un graphe appartenant à la grammaire est déduit à travers des applications successives de certaines productions de cette grammaire.

Nous expliquerons, dans ce qui suit, une variété d'approches de transformation de graphes ainsi que des exemples d'extensions établies sur les grammaires de graphes.

1.2.1. Approche basique

La règle de réécriture de base qui permet de transformer un graphe hôte G en un graphe G' est de la forme p = (L, R). L et R sont deux graphes respectivement désignés dans la littérature par la partie gauche (« left-hand side » : LHS) et la partie droite (« right-hand side » : RHS). L'application de cette règle consiste à trouver une occurrence du graphe L, aussi appelé « graphe mère », dans le graphe G et de la remplacer par une copie du graphe R, appelé aussi « graphe fille ». La figure 4 donne une description schématique de la règle de transformation de type (L, R).

Le « mapping » entre L et G, i.e., le fait de faire correspondre le graphe de la partie L à une des ses occurrences dans le graphe G, ou encore, unifier les nœuds du graphe L à ceux d'un sous-graphe de G, représentant une occurrence de L dans G, est formalisé à travers le concept d'« homomorphisme de graphes », appliqué de L vers G.

Définition : **Homomorphisme de graphes** [Guennoun, 2007] :

Un homomorphisme d'un graphe G1 = (N1, A1) vers un graphe G2 = (N2, A2) est une injection $F \subseteq N1 \times N2$ telle que pour chaque paire n_i, n_j appartenant à N1, si l'arc $(n_i, n_j) \in A1$, alors l'arc $(F(n_i), F(n_j)) \in A2$.

Le remplacement d'une partie du graphe L par une autre peut engendrer une suppression de certains nœuds dans le graphe G. Le nouveau graphe peut donc contenir des ars qui ont perdu leurs nœuds sources ou cibles. Cette anomalie est appelée le problème des « arcs suspendus ». Un exemple

d'application de ce type de règles est présenté dans la figure 5.

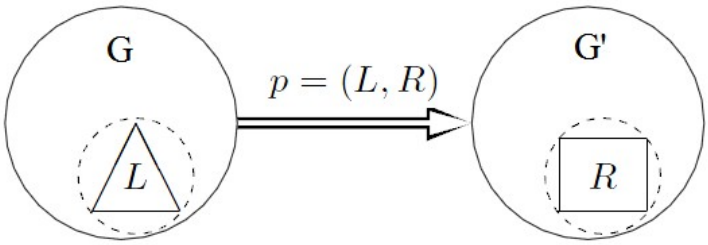


Figure 4 : Règle de transformation de graphes de type (L, R) [Ehrig et al., 2006]

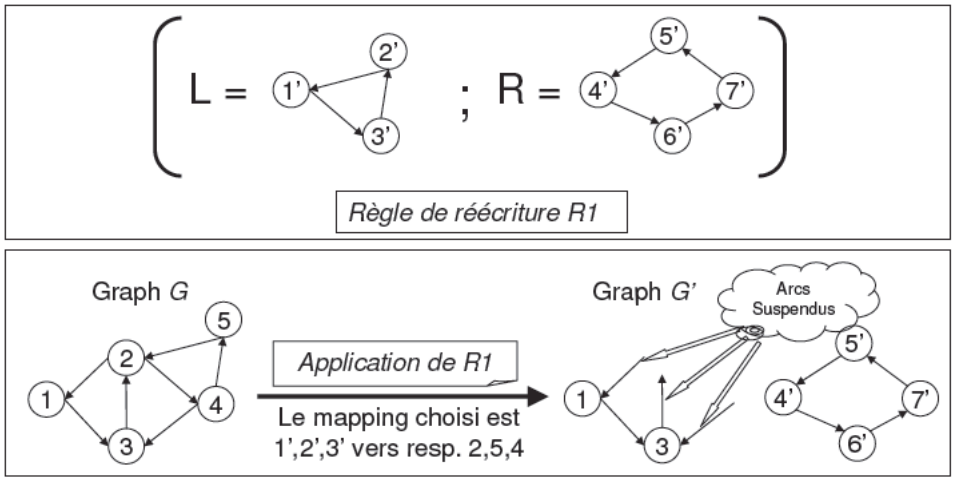


Figure 5 : Application d'une règle de réécriture avec le problème des arcs suspendus [Guennoun, 2007]

Afin de pallier à ce problème, deux solutions principales ont été proposées s'agissant des approches de transformation de graphes SPO et DPO [Ehrig et al., 2006] [Guennoun, 2007].

1.2.2. Traitement des « arcs suspendus »

a) L'approche SPO (Single PushOut)

L'approche SPO garde la forme basique de la règle de transformation, i.e., le couple (L,R). Mais contrairement à l'approche basique, certains nœuds et arcs présents dans la partie L ne seront plus supprimés. Ce sont ceux qui sont présents à la fois dans la partie L et la partie R de la règle. Par conséquent, lors de l'application de la règle, le sous-graphe de G qui sera supprimé est celui qui correspond à (L \ (L∩R)). Le graphe correspondant à (R \ (L∩R)) sera ajouté. Enfin, tous les arcs suspendus qui vont apparaître suite à l'application de la règle seront supprimés. La figure 6 présente un exemple d'application d'une règle de transformation adoptant l'approche SPO. Dans cet exemple, dans le graphe G1, les nœuds 2 et 5 appartenant à L∩R vont être gardés, tandis que le nœud 4 ainsi que les arcs le reliant avec les nœuds 2 et 5 vont être supprimés. Le sous-graphe correspondant à R \ (L∩R) est inséré. L'application de R2 va également engendrer l'apparition d'un arc suspendu, qui reliait le nœud supprimé 4 avec le nœud 3 de G1. Par conséquent, cet arc est supprimé.

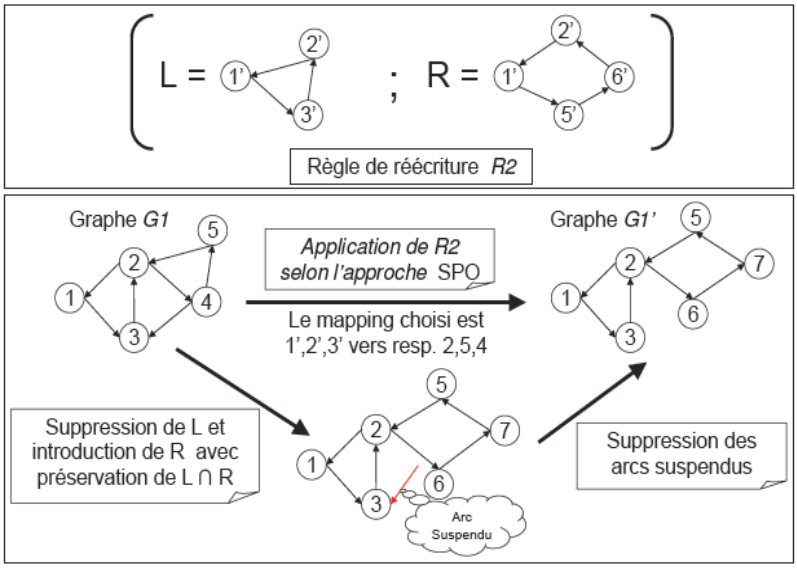


Figure 6 : Application d'une règle de réécriture selon l'approche SPO [Guennoun, 2007]

b) L'approche DPO (Double PushOut)

Cette approche étend la structure de la règle de transformation de base (L, R) en ajoutant le paramètre K qui représente clairement le sous-graphe à préserver du graphe L. DPO stipule que la règle n'est applicable que si son application ne va pas engendrer le problème des arcs suspendus. Ayant vérifié cette condition, avec l'existence d'une occurrence du graphe L dans G, la règle conduira à la suppression du sous-graphe de G, correspondant à l'occurrence de (L\K), et à l'insertion de la partie correspondante à (R\K) dans G. La figure 7 illustre une application d'une règle de réécriture adoptant l'approche DPO.

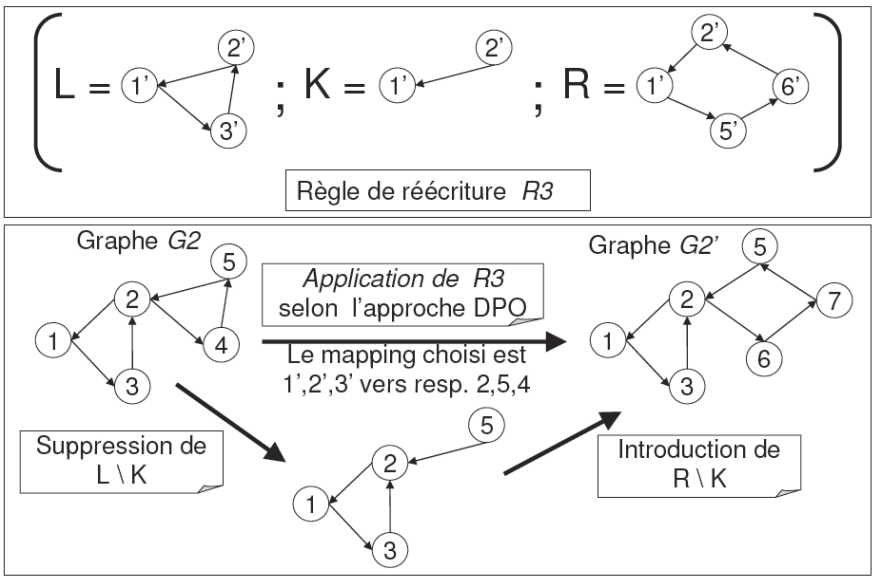


Figure 7 : Application d'une règle de réécriture selon l'approche DPO [Guennoun, 2007]

Dans cet exemple, le graphe G2 est presque identique au graphe G1 de la figure 6. Il en diffère seulement par l'absence de l'arc partant du nœud 3 vers le nœud 4. L'application de R3 est donc applicable puisque elle ne fait pas apparaître des arcs suspendus.

c) Combinaison des approches SPO et DPO :

Une approche combinant les solutions SPO et DPO consiste à adopter la structure des règles de transformation de type DPO, tout en traitant le problème des arcs suspendus selon la logique de l'approche SPO. Dans cette approche, une règle de transformation est de la forme (L, K, R). La règle est applicable s'il y a une occurrence du graphe L dans G. La transformation est appliquée d'une manière équivalente à une règle DPO. Cependant, l'absence potentielle des arcs suspendus n'est pas une condition d'application de cette règle.

1.2.3. Les conditions d'application négatives (NAC)

L'approche de base des règles de transformation stipule qu'un graphe G doit contenir une occurrence du graphe L afin que la règle correspondante soit applicable. Une extension de cette approche précise, à contrario, qu'un graphe G ne doit pas contenir une occurrence du graphe spécifié dans la partie NAC. Les conditions d'application négatives ajoutent donc une contrainte exigeant l'absence d'un homomorphisme de NAC vers G, afin que la règle puisse être applicable. L'extension d'une règle de transformation, qu'elle soit de type SPO ou DPO, par la partie NAC n'agit que sur l'applicabilité de la règle et n'affecte donc pas l'opération de transformation du graphe hôte. La règle R4 de type SPO, présentée dans l'exemple de la figure 8, est identique à celle présentée dans l'exemple de la figure 6. Cependant, une contrainte supplémentaire y est posée. Ayant fait un mapping respectif partant des nœuds 1', 2' et 3' vers les nœuds 2, 5 et 4 du graphe G1, la prochaine étape consiste à vérifier qu'il n'existe pas, dans G1, un arc ayant le nœud 4 comme source, et un nœud autre que 2 et 5 comme destination. Par conséquent, R4 n'est pas applicable dans ce cas vu la présence de l'arc (4,3). R4 est par contre applicable si on considère l'occurrence constituée des nœuds 4, 2 et 5.

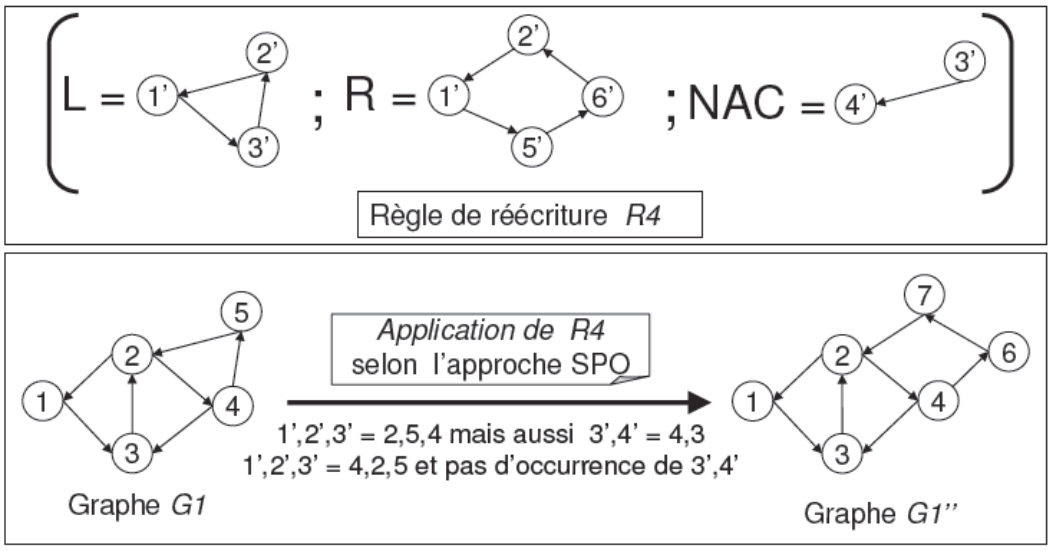


Figure 8 : Application d'une règle de type SPO, étendue par une condition d'application négative [Guennoun, 2007]

1.3. Les grammaires de graphes étendues

Les productions des grammaires de graphe classiques qui sont définies par le quadruplet (AX, NT, T, P) se limitent à la gestion du sous graphe à supprimer et de celui à insérer dans un graphe hôte. Ils ne donnent pas ainsi la possibilité d'ajouter d'éventuels liens entre les nœuds déjà existants dans le graphe hôte, et ceux qui sont insérés suite à l'application d'une règle de transformation. Des extensions de la structure des grammaires de graphes, à savoir NLC et NCE [Rozenberg, 1997] [Guennoun, 2007], apportent de nouvelles options de connexion des nœuds rajoutés dans un graphe G avec les anciens nœuds, à travers le concept des « instructions de connexion ».

1.3.1. Le mécanisme NLC (Node Label Controlled mechanism)

Une grammaire de graphes adoptant le mécanisme NLC est décrite par le quintuplet (AX, NT, T, P, C). Le nouvel élément C spécifie les instructions de connexion du graphe fille au graphe hôte qui sont de la forme (μ, δ), où μ, δ sont les labels correspondant aux nœuds concernés par la transformation. Ces instructions sont définies d'une manière globale et sont exécutées une par une, après chaque application de l'une des productions décrites dans l'élément P. Ce mécanisme s'applique dans les graphes labellés et non orientés et se base sur les labels des nœuds pour déterminer les arêtes à insérer entre les nœuds du graphe fille et les nœuds voisins du graphe mère.

Les productions d'une grammaire de graphes de type NLC sont de la forme X → D, où la partie X est équivalente à la partie L d'une règle de transformation de base, à l'exception qu'elle contient seulement un nœud non terminal comportant un label. Quant à la partie D, elle est équivalente à la partie R d'une règle de transformation de base. Elle spécifie un graphe labellé et non orienté comportant des nœuds terminaux et/ou des nœuds non terminaux.

L'application des instructions de connexion consiste à insérer une arête entre chaque nœud du graphe spécifié par D, ayant un label μ, et les nœuds voisins du nœud non terminal spécifié dans X,

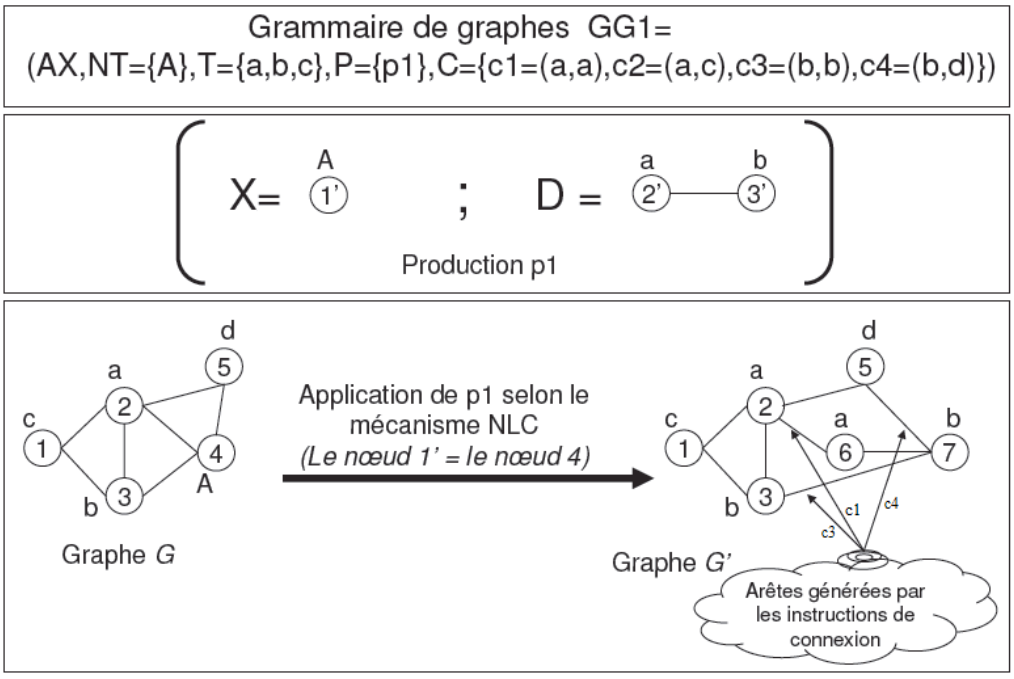


Figure 9 : Exemple d'une grammaire avec le mécanisme NLC [Guennoun, 2007]

qui comportent le label δ. Si une telle combinaison n'est pas vérifiée, l'instruction de connexion est tout simplement ignorée et n'affecte pas l'applicabilité des productions de la grammaire concernée.

Dans l'exemple de la figure 9, le nœud 4 est remplacé par les nœuds 6 et 7 reliés par une arête. L'instruction c1 = (a, a) relie le nœud 6, labellé par a, au nœud 2, labellé lui aussi par a. L'instruction c2 = (a, c) est ignorée vu l'absence d'un nœud voisin du nœud 4, ayant le label c.

NLC est un mécanisme simple mais n'offre que des possibilités de connexions limitées. Il n'est applicable que dans des graphes non orientés tout en se limitant aux labels des nœuds. D'autres extensions de ce mécanisme offrent des techniques de manipulation plus puissantes.

a) Directed NLC (dNLC)

Cette extension s'applique dans les graphes orientés. Elle s'intéresse ainsi au sens des arcs. Une instruction de connexion est donc décrite par un triplet (μ, δ, d). L'élément d, pouvant avoir l'une des valeurs « in » ou « out », permet à l'instruction de préciser le sens de l'arc cherché entre le graphe spécifié par X et les nœuds qui en sont voisins, et d'introduire le nouvel arc dans le même sens précédemment décrit par d.

Une autre extension permet de traiter également le sens des arcs à introduire. Une instruction de connexion est ainsi décrite par un quadruplet (μ, δ, d, d'). Le sens du nouvel arc est spécifié, dans ce cas, dans l'attribut d'.

Dans l'exemple de la figure 10, le nœud 4 est remplacé par les nœuds 6 et 7. L'instruction c4 = (b, d, out) de la grammaire GG2 réinsère l'arc qui reliait le nœud 4 au nœud 5, entre les nœuds 7 et 5 en

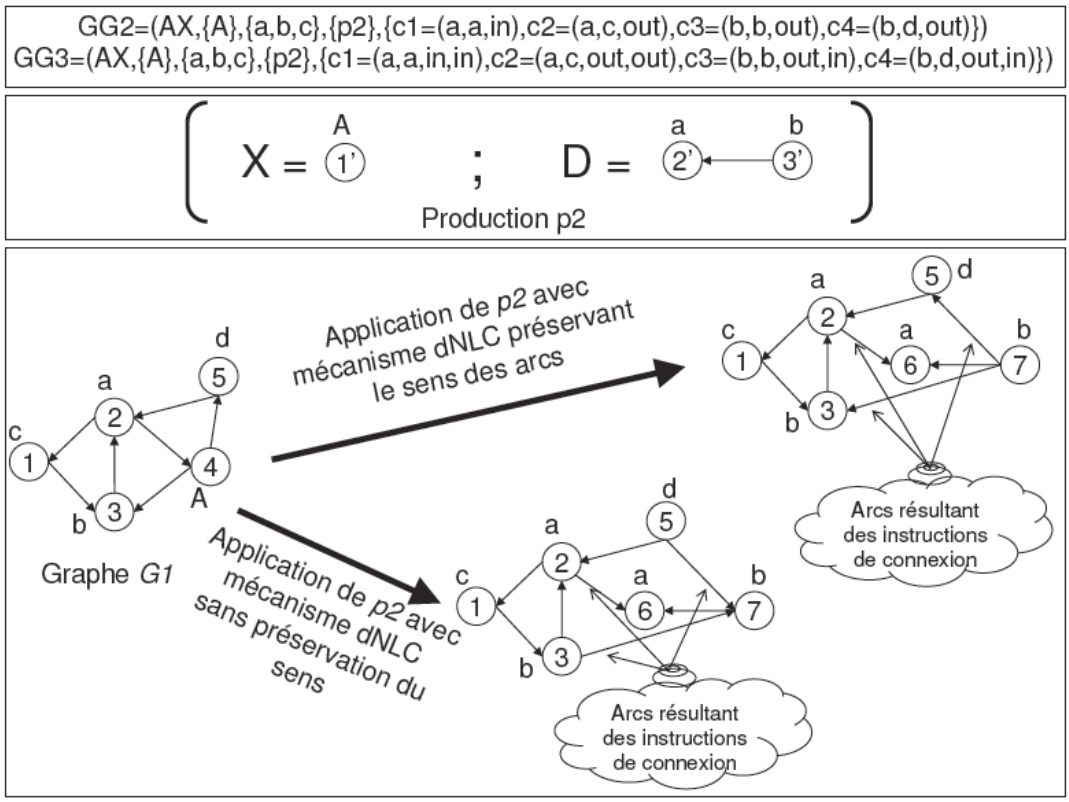


Figure 10 : Exemple d'une grammaire avec le mécanisme dNLC [Guennoun, 2007]

conservant le même sens. L'instruction c4 de la grammaire GG3 effectue la même transformation mais en inversant le sens de l'arc introduit.

b) Edge label NLC (eNLC)

Cette extension s'intéresse aux graphes étiquetés. Une instruction de connexion eNLC est décrite par un triplet (μ, p/q, δ). Le champ p/q permet à l'instruction de connexion de préciser qu'une arête d'étiquette q est introduite entre tous les nœuds du graphe mère qui sont p-voisins du nœud non terminal spécifié par X, et les nœuds concernés du graphe spécifié par D. Deux nœuds sont p-voisins s'ils sont liés par un lien étiqueté par p.

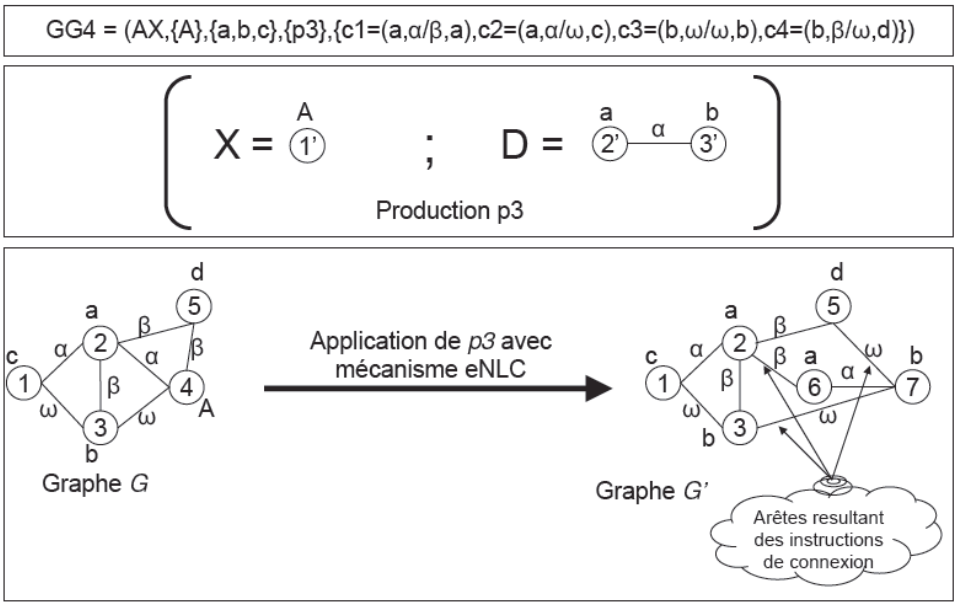


Figure 11 : Exemple d'une grammaire avec le mécanisme eNLC [Guennoun, 2007]

Dans l'exemple de la figure 11, le nœud 4 est remplacé par les nœuds 6 et 7. L'instruction c1 = (a, α/β, a) de la grammaire GG4 remplace l'arête d'étiquette α qui reliait le nœud 4 au nœud 2, par une arête entre les nœuds 6 et 2 en changeant l'étiquette par β.

Nous pouvons prendre en compte une combinaison des deux extensions dNLC et eNLC pour obtenir l'extension edNLC. Une instruction de connexion de type edNLC est de la forme (μ, p/q, δ, d, d').

1.3.2. L'approche NCE

Une autre extension des grammaires de graphes adoptant la technique des instructions de connexion est désignée par NCE (Neighbourhood Controlled Embedding). Contrairement au mécanisme NLC, les nœuds du graphe fille, insérés suite à l'application d'une production, ne sont plus différenciés par leurs labels. Les instructions se réfèrent plutôt directement aux nœuds concernés. Par conséquent, dans une grammaire adoptant l'approche NCE, ces instructions ne sont pas définies d'une manière globale, mais, à chaque production est associé un ensemble d'instructions de connexion exécutées suite à l'application des transformations décrites par ladite production.

Une grammaire de graphes comportant des productions de type NCE est ainsi définie par le

quadruplet (AX, NT, T, P). Les productions présentes dans la partie P sont de la forme ((X, D), C). X et D gardent la même signification que dans les règles d'une grammaire adoptant le mécanisme NLC. C représente l'ensemble des instructions de connexion spécifiques à la production correspondante. Une instruction de connexion NCE est de la forme (n, δ), où n remplace la partie μ pour bien désigner un nœud bien déterminé appartenant au graphe fille, plutôt que tous ceux qui portent le label μ.

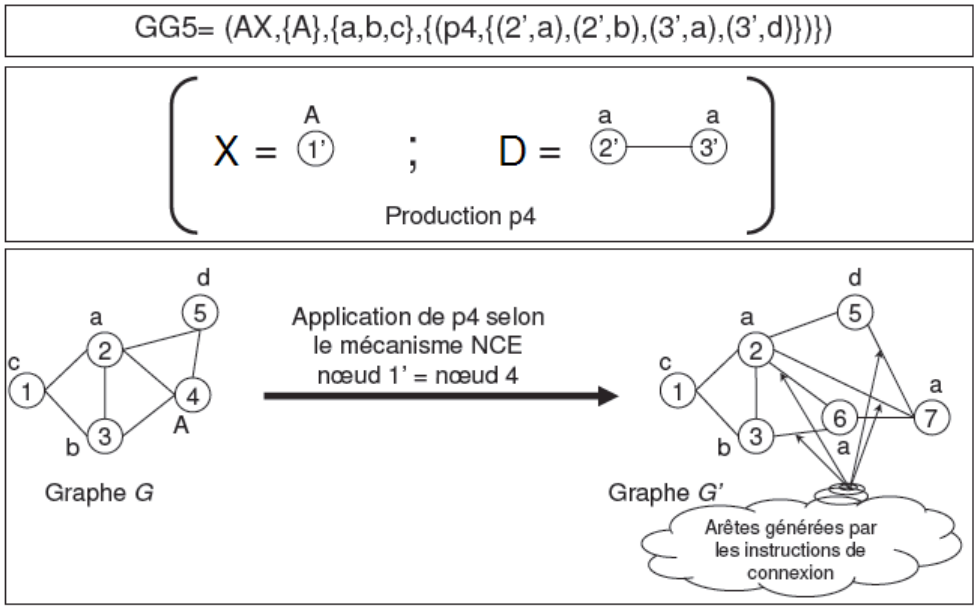


Figure 12 : Exemple d'une grammaire avec l'approche NCE [Guennoun, 2007]

Dans l'exemple de la figure 12, bien que les nœuds 6 et 7, qui sont insérés relativement aux nœuds respectifs 2' et 3' du graphe fille, portent le même label, NCE a permis de connecter indépendamment chacun de ces deux nœuds aux voisins du graphe mère spécifié par X. Cette option n'est pas possible en utilisant le mécanisme NLC. En effet, le nœud 6 a été lié aux nœuds 2 et 3 portant respectivement les labels a et b, tandis que le nœud 7 a été lié aux nœuds 2 et 5 portant respectivement les labels a et d.

Une grammaire de graphes de type NCE peut, elle aussi, adopter les mêmes extensions possibles au sein du mécanisme NLC. L'approche eNCE s'intéresse aux étiquettes portées par les arêtes, alors que dNCE s'intéresse au sens des arcs. En combinant les deux extensions, une instruction de connexion edNCE est de la forme (n, p/q, δ, d, d'). Elle concerne tous les nœuds qui sont p-voisins du nœud non terminal spécifié par X, dans le sens d, et qui sont labellés par δ. L'exécution réinsère ainsi un arc labellé par q connectant le nœud n aux nœuds voisins concernés, dans le sens d'.

1.4. Exemple combinatoire (edNCE avec SPO, DPO et NAC)

Plusieurs combinaisons réunissant, d'une part, les diverses approches de réécriture de graphes expliquées dans la partie 1.2, et, d'autre part, une parmi les extensions des grammaires de graphes expliquées dans la partie 1.3, sont possibles. Dans ce qui suit, nous présentons un exemple d'une grammaire de graphes étendue par l'approche edNCE et qui utilise des productions adoptant une combinaison des approches SPO, DPO et NAC. Cette grammaire de graphes est donc décrite par le quadruplet (AX, NT, T, P) où P représente l'ensemble des productions de type (L, K, R, N, C). Les

instructions de connexion edNCE, définies dans C, sont de la forme (n, p/q, δ, d, d').

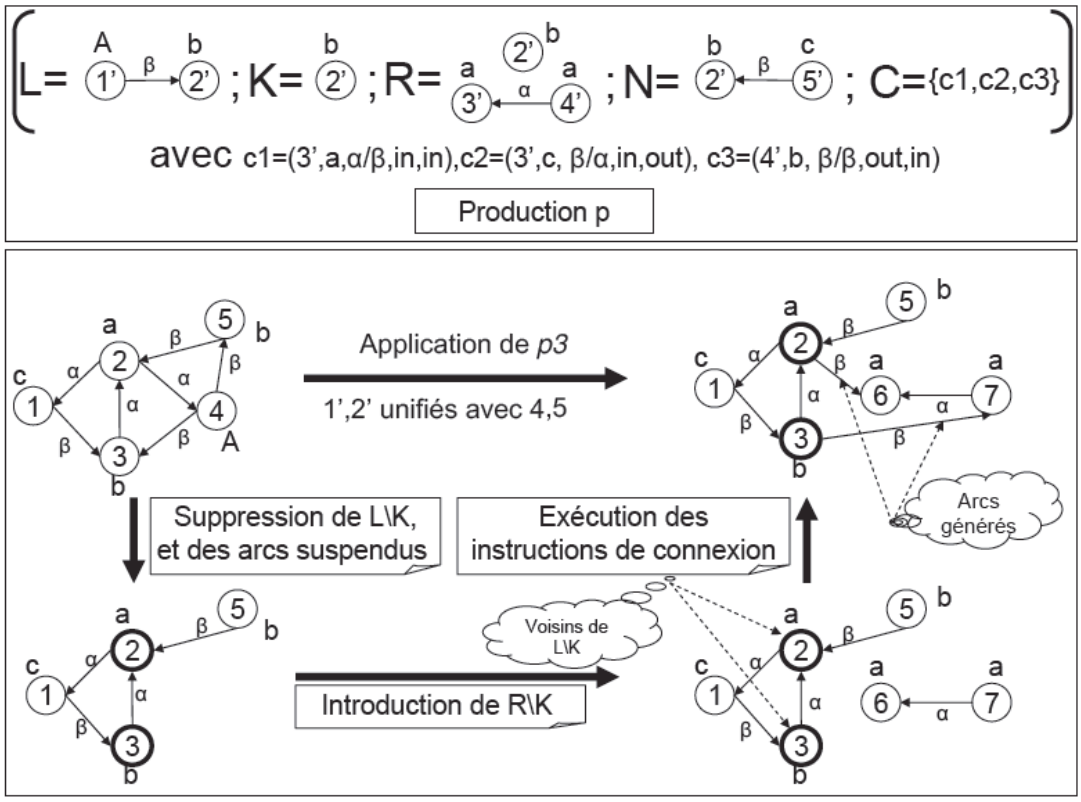


Figure 13 : Exemple combinatoire d'une production d'une grammaire de graphes étendue [Guennoun, 2007]

La production illustrée par la figure 13 est un exemple de cette combinaison d'approches. Le graphe spécifié par L\K correspond au nœud non terminal qui devrait être spécifié par X selon NCE. La partie D d'une grammaire NCE est toujours équivalente à la partie R de la production présentée.

Dans cet exemple, le graphe mère possède deux occurrences du graphe fille spécifié par L. Selon le mapping respectif partant des nœuds 1' et 2' vers les nœuds 4 et 3, la condition d'application négative exige que le nœud 3, correspondant au nœud 2' du graphe L, ne doit pas être lié par un arc entrant portant l'étiquette β, et ayant comme source un nœud labellé par c. La production n'est donc pas applicable suivant ce mapping vu la présence du nœud 1 labellé par c qui est β-voisin avec le nœud 3 par le moyen d'un arc entrant vers ce dernier.

À contrario, cette règle est bien applicable en utilisant le mapping respectif des nœuds 1' et 2' vers 4 et 5 et son application implique la suppression de la partie correspondant à L \ K, i.e., le nœud 4, et des arcs suspendus qui découlent de cette action. Ensuite une copie homomorphe de R \ K, i.e., les nœuds 6 et 7 et l'arc qui les relie, est insérée dans le graphe mère en l'attachant aux voisins du graphe correspondant à L \ K, comme décrit dans les instructions de connexion. Étant donné la présence d'un arc d'étiquette β partant de 2 vers 4, l'instruction c1 relie le nœud 6 au nœud 2 qui est labellé par a, par un arc entrant étiqueté par β. L'instruction c2 est ignorée puisqu'il n'y a aucun voisin du nœud 4, ayant le label c. De la même manière que c1, c3 relie le nœud 7 par un arc entrant d'étiquette β, partant du nœud 3 labellé par b.

2. Méta-modèle pour une adaptation structurelle automatique au sein d'une architecture orientée composants et services

2.1. Principes fondamentaux

Dans le cadre des travaux décrits dans [Guennoun, 2007] et [Bouassida *et al.,* 2008], les auteurs ont défini un méta-modèle qui spécifie « plusieurs maillons de la chaîne de description architecturale ». Un cadre formel y a été spécifié afin de modéliser leurs approches.

Ce méta-modèle englobe quatre modules basés sur les grammaires de graphes et sur des systèmes étendus des règles de réécriture de graphes. Ces modules assurent :

- la description des styles des architectures à instancier,
- la définition des systèmes de « transformation verticale » afin d'obtenir différentes facettes décrivant des niveaux d'abstraction spécifiques,
- la définition des systèmes de « transformation horizontale » qui abordent l'aspect évolutif des architectures en mettant en œuvre des protocoles de reconfiguration qui permettent le passage entre les différentes instances, et
- la définition des contraintes et propriétés architecturales qu'une instance d'architecture doit respecter [Guennoun, 2007].

Dans le cadre de notre étude, nous nous intéressons aux deux derniers modules. Nous expliquons dans ce qui suit les approches qui y sont utilisées et nous présentons leur application au sein d'un cas d'étude précis.

2.2. Gestion de l'évolution de l'architecture

Avant de traiter l'aspect de reconfiguration, nous pouvons commencer par une brève explication du système de description des instances des architectures.

Les grammaires de graphes représentent un outil théorique très puissant s'agissant de la description des styles architecturaux [Bouassida et al., 2008]. Une instance d'une architecture, qui représente une configuration structurelle d'un système à un instant particulier, est ainsi décrite à travers un graphe dont les nœuds représentent chacun un composant ou un service, et les arcs ou les arêtes décrivent les relations entre les composants/services faisant partie de ladite instance [Guennoun, 2007].

Étant donné la variation, d'une part, des « paramètres nécessaires pour la description d'un composant » et, d'autre part, des types d'interdépendances reliant ces composants, entre les différents « contextes de description » et même au sein d'une même application, les auteurs ont donc proposé un cadre dans lequel les nœuds des graphes sont « multi-labéllés ». Le nombre d'étiquettes des arcs peut également varier d'un lien de collaboration à un autre.

En ce qui concerne l'évolution dynamique des architectures, il s'agit de spécifier les actions qui permettent, suite à un changement du contexte, de produire une nouvelle instance à partir de l'instance courante. Pour ce faire, les auteurs décrivent les règles de gestion de reconfiguration de l'architecture en combinant diverses types de règles de transformation de graphes [Guennoun, 2007]

[Bouassida et al., 2008].

Les auteurs se sont inspirés de la logique des grammaires génératives de Chomsky [Bouassida et al., 2008]. Ils proposent ainsi une grammaire de graphes de la forme (AX, NT, T, P), où AX, NT et T représentent respectivement l'axiome, l'ensemble des nœuds non terminaux, et l'ensemble des nœuds terminaux. Parmi les différentes approches déployées dans la structure des productions de grammaires de graphes, les auteurs utilisent des productions de type (L, K, R, N, C) combinant l'approche :

- Double PushOut (DPO), intégrant les conditions d'application négatives (NAC), qui a comme structure (L, K, R, NAC) et qui aura comme conséquence la suppression des occurrences du « graphe mère » L privé de K dans un « graphe hôte » G, et le rajout d'une copie de R privée de K, en condition de l'absence d'une occurrence de NAC dans L, et
- les instructions de connexion étendues (edNCE) de la forme (n, p/q, δ, d, d') qui concernent les nœuds n' qui sont p-voisins du graphe mère, dans le sens d. Ils impliquent l'introduction d'un arc, de sens d' et de label q, connectant le nœud n du graphe fille à tous les nœuds n' qui sont labellés par δ.

Chaque action de reconfiguration d'une architecture est déclenchée suite à l'occurrence d'un ou de plusieurs événements appelés « événements de reconfiguration ». Les auteurs ont proposé un dispositif permettant d'identifier les événements déclencheurs d'une reconfiguration et d'indiquer les règles de transformation correspondantes, appelé « protocole de reconfiguration ». En plus de la gestion des événements de reconfiguration, ce protocole assure la combinaison et l'ordonnancement des règles de transformation à appliquer et l'instanciation des paramètres passés par l'événement de reconfiguration. Un protocole est représenté par un ensemble de triplets (T, C, I) où T comporte l'événement-type et les paramètres qui sont transportés avec cet événement, et C définit la combinaison des règles de transformation. I représente l'ensemble des règles d'instanciation permettant d'affecter la valeur des paramètres transportés par la partie T à certaines variables dans les règles de transformation décrites dans C. Formellement, un protocole de reconfiguration est décrit comme suit : PR = { (T, C, I)* }.

PR1 = { (T = E(p1, p2), C = r1(X1, X2) ; r2(X2)*, I = { (p1, X1), (p2, X2), }) }

À la réception de l'événement E, le protocole de reconfiguration PR1 affecte les valeurs des paramètres p1 et p2 respectivement aux variables X1 et X2 au niveau des deux règles r1 et r2 décrites dans C. PR1 traite ensuite la combinaison C desdites règles de transformation. Dans cet exemple, Si r1(p1, p2) n'est pas applicable, alors toute la combinaison est annulée et l'architecture n'est pas reconfigurée. Si cette règle est applicable, le protocole exécute la transformation et passe à la deuxième étape de la combinaison. r2(p2)* signifie une application multiple de r2(p2), i.e., exécuter la transformation décrite par r2 autant de fois que cette règle est applicable.

2.3. Propriétés architecturales

Le méta-modèle proposé dans le cadre de ces mêmes travaux offre une description des « propriétés architecturales » d'une application, caractérisant, entre autres, le nombre de liens de collaboration entre les différentes entités et le patron architectural de l'application [Guennoun, 2007].

Étant donné qu'il ne s'agit plus, dans ce contexte, d'une transformation, mais plutôt d'une vérification de certaines contraintes d'une architecture, projetées sur un graphe. Ces contraintes sont

décrites « sous la forme de règles de réécriture réduites » [Guennoun, 2007]. Le besoin de vérifier l'applicabilité de ces dernières est ainsi satisfait à travers les attributs L et NAC. Par conséquent, les règles sont définies avec la structure (L, N).

Un autre aspect est également défini pour donner plus de sens aux règles grammaticales :

- Le concept de « propriétés positives » : Il stipule qu'une instance d'une architecture bien déterminée doit contenir une sous-configuration, décrite par les attributs L et N, tout le long de la durée de son exécution.
- Les « propriétés négatives », à contrario, exigent que la sous-configuration décrite par la règle correspondante ne doit pas apparaître dans la configuration concernée.

2.4. Exemple d'application

Cette approche a été adoptée dans le cadre de plusieurs applications à systèmes coopératifs. Un exemple concret que nous pouvons évoquer est un cas d'études qui gère un système d'Opérations d'Interventions d'Urgence (OIU). Dans ce qui suit, les auteurs ont su appliquer les concepts théoriques qu'ils ont proposés sur un système actif et instable.

De telles activités de gestion de crises requièrent des équipes bien disposées, constituées de participants habiles et mobiles (robots, personnel militaire, etc.). Ces derniers doivent collaborer afin de réaliser une mission commune [Guennoun, 2007] [Bouassida et al., 2008].

Ainsi, une équipe d'intervention se structure en des agents déployés sur des machines distribuées. La collaboration est assurée à travers des moyens de communication divers (réseaux filaires, réseaux sans fil,…). Une telle équipe est composée d'acteurs ayant des rôles différents et hiérarchiquement organisés. Un contrôleur, responsable de la mission, gère des coordinateurs qui dirigent, chacun, un groupe d'investigateurs de terrain. Le rôle de chaque type d'acteur est expliqué comme suit :

- Le contrôleur : Il supervise la mission entière. Il a comme responsabilité de piloter les coordinateurs à sa charge. Il est installé sur une machine fixe et possède aussi bien des ressources d'énergies permanentes que des performances importantes de communication et de calcul.
- Le coordinateur : Attaché au contrôleur, un coordinateur doit gérer ses investigateurs, tout au long de la mission, et leur attribuer des tâches à exécuter. Sa fonctionnalité consiste ainsi à collecter les informations transmises par les investigateurs, les interpréter et diffuser les synthèses déduites vers le contrôleur et les investigateurs. Les coordinateurs ont eux aussi des « capacités logicielles et matérielles » importantes et sont déployés sur des machines fixes.
- L'investigateur : Installé sur une machine mobile de faibles ressources d'énergie et de calcul, un investigateur a deux fonctionnalités essentielles, à savoir :
    - l'observation et l'analyse du « champ opérationnel » en faisant parvenir des rapports de description au coordinateur concerné, et
    - l'assurance de certains services comme l'assistance, le sauvetage et le dépannage.

L'architecture de l'application est décrite sur trois niveaux d'abstraction : application, middleware et

réseau. Dans notre étude, nous nous limitons au niveau « application », le plus abstrait entre les trois. Il définit les rôles de chaque composant ainsi que l'échange de données et les contraintes architecturales.

Le scénario d'exécution comporte deux états, à savoir une phase d'exploration et une phase d'action. Les rôles des composants et la nature des échanges de données dépendent de la phase dans laquelle se situe l'activité du contexte en un instant déterminé.

En phase d'exploration, les investigateurs envoient des flux de données descriptifs fréquents (O, comme Observation) tout au long de l'examen du champ d'exécution. Des rapports d'analyse (R) sont aussi produits de temps en temps et expédiés au coordinateur responsable. Les coordinateurs envoient eux aussi des rapports de synthèse (R) au contrôleur de la mission d'une manière périodique.

L'architecture est modélisée par le moyen d'un graphe orienté, étiqueté et multi-labéllé dans lequel les différents acteurs sont représentés par des nœuds et les échanges de données sont représentés par des arcs. La figure 14 montre une instance de l'architecture en phase d'exploration.

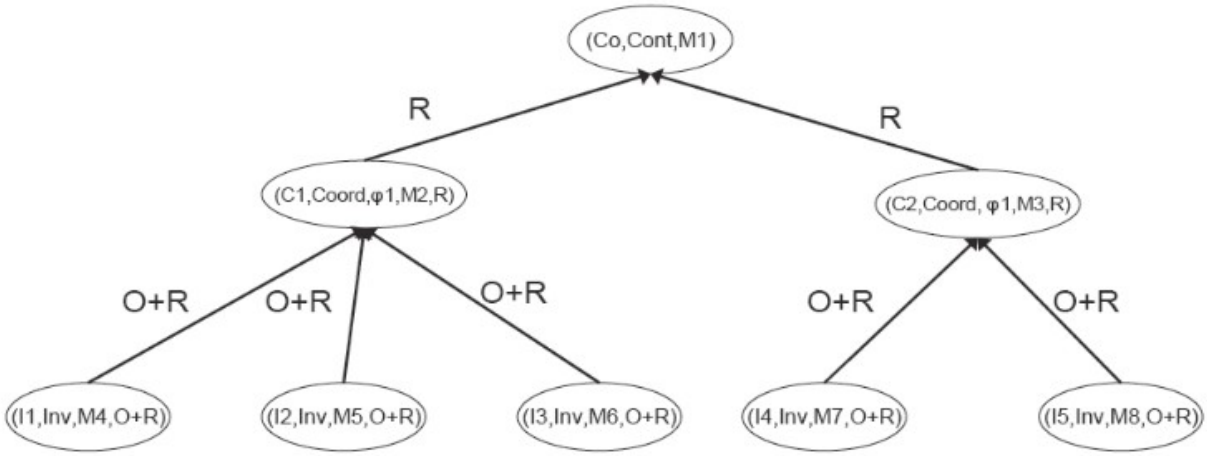


Figure 14 : Exemple d'une instance d'une architecture dans la phase d'exploration [Guennoun, 2007]

Dans cet exemple, tous les nœuds ont des attributs communs, à savoir :

- l'identifiant du composant selon son type (*Co* pour le contrôleur, *Ci* pour les coordinateurs et *Ii* pour les investigateurs),
- le type du composant (*Cont* pour le contrôleur, *Coord* pour les coordinateurs et Inv pour les investigateurs), et
- le nom de la machine sur laquelle est déployé le composant concerné (*Mi*).

Le champ *O+R* dans un investigateur signifie qu'il est en phase d'exploration et communique à son coordinateur des données de type O et de type R. L'attribut *R* dans un coordinateur indique qu'il produit des données de type R, en étant en phase d'exploration (comme indiqué par l'attribut *φ1).* Les labels des arcs représentent la nature des « feed-backs » transmis par les investigateurs et les coordinateurs.

Comme indiqué dans la section précédente, les contraintes et propriétés architecturales sont

modélisées à travers des règles de réécriture réduites de type (L,N). Comme application de cette approche, nous pouvons présenter « la caractérisation » de quelques propriétés au sein d'une architecture en phase d'exploration [Guennoun, 2007].

- La propriété : "(L = {N(X_{Cont} , "Cont" , M)} ; N = {}), Pos" [1] est une règle positive qui exige la présence d'au moins un contrôleur dans chaque instance de l'architecture de l'application.
- La propriété : "(L = {N1($X1_{Cont}$, "Cont" , M1), N2($X2_{Cont}$, "Cont" , M2)} ; N = {}), Neg" est une règle négative qui exprime l'obligation d'avoir un et un seul contrôleur.
- Une propriété plus complexe : "(L = {N1(X_I , "Inv" , M1, "O+R")} ; N = {N2(X_{Coord} , "Coord" , φx, M2, "R"), N1 $\xrightarrow{\text{"O + R"}}$ N2}), Neg" [2] implique la non applicabilité de la règle qui stipule qu'un investigateur n'est pas dirigé par un coordinateur. En d'autres mots, un investigateur est donc « forcément relié à un coordinateur » [Guennoun, 2007].

L'architecture de l'étude de cas évoquée dans le cadre de ces travaux est dynamique. Les changements qui peuvent y survenir sont classés en deux perspectives :

- La perte d'un investigateur suite à une panne, à sa destruction, ou à l'épuisement de ses réserves d'énergie.
- Le passage de la « phase d'exploration » vers la « phase d'action » suite à la découverte d'une situation critique et vice-versa (suite à la fin de la situation critique).

En premier lieu, nous considérons le cas le plus simple, s'agissant de la perte d'un investigateur en phase d'exploration. La reconfiguration consiste ainsi à ne plus le prendre en considération comme composant du système et à répartir ses tâches aux autres investigateurs de sa section. Supposant que le composant défectueux est celui qui est hébergé sur la machine M5, l'événement de reconfiguration Perte_Inv_Explo(M5), relatif à la perte de ce composant déclenche le triplet (T = Perte_Inv_Explo(m), C = R1(M1), I = { (m, M1) }) du protocole de reconfiguration qui implique l'exécution de la règle R1, décrite ci-dessous, en remplaçant M1 par M5.

R1 = (L = { N1(X_{I1}, "Inv", M1, "O+R"), N2(X_{coord} , "Coord" , "φ1", M2),
N3(X_{I2} , "Inv", M3, "O+R"), N1 $\xrightarrow{\text{"O + R"}}$ N2, N3 $\xrightarrow{\text{"O + R"}}$ N2 } ;
K = { N2(*Xcoord*, "Coord", "φ1", M2), N3(X_{I2} , "Inv", M3, "O+R"),
N3 $\xrightarrow{\text{"O + R"}}$ N2 } ;
R = { } ; N = { } ; C = { })

Cette règle de transformation de type (L, K, R, N, C) engendre la suppression du nœud N1 relatif au composant perdu, étant donné qu'il n'est pas présent dans la partie K. Cependant, R1 n'est applicable que s'il existe au moins un autre investigateur dans la section concernée par la reconfiguration.

Dans le cas où un investigateur découvre une situation critique, l'architecture de l'équipe doit être reconfigurée afin de passer à la phase d'action. L'investigateur « critique », désigné désormais par « α », continue à envoyer ses observations O et ses rapports R à son coordinateur, et diffuse, outre cela, les données O aux investigateurs dites « β », pilotés par le même coordinateur. Ces derniers ne

1 *Les attributs écrits entre deux guillemets représentent des constantes et les autres sont des variables. Par exemple, dans la première propriété, le nœud concerné par la partie L doit avoir trois attributs. X_{cont} et M sont des variables pouvant accepter n'importe quelle valeur. Cependant, le deuxième attribut doit obligatoirement avoir la valeur « Cont ».*

2 *La flèche $\xrightarrow{\text{"O + R"}}$ est une représentation simplifiée d'un arc ayant l'étiquette O+R liant le nœud N1 au nœud N2. Ce format sera utilisé dans la suite du document afin de désigner un arc étiqueté.*

communiquent maintenant que les données R à leur coordinateur.

Suite à ses changements, la nouvelle grammaire spécifie que l'attribut *φ1* du coordinateur touché par la reconfiguration prend la valeur *φ2*. L'attribut indiquant le type de l'investigateur prend la valeur « Inv_α » dans l'investigateur critique et reste intact dans les autres investigateurs. L'attribut « O+R »

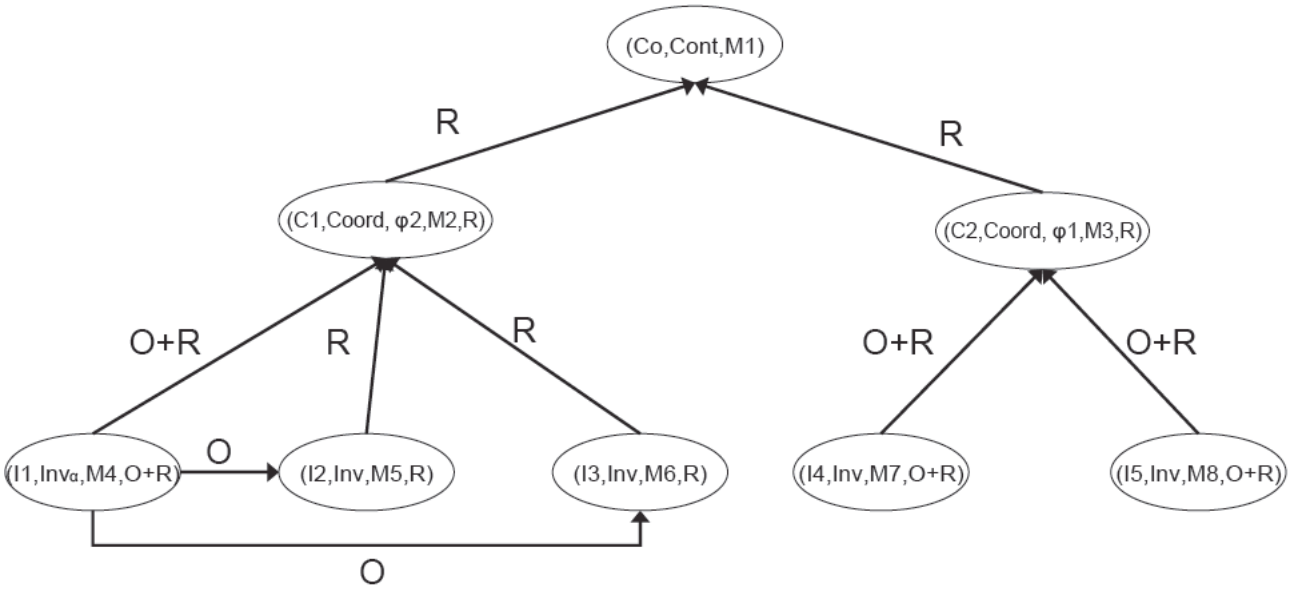


Figure 15 : Exemple d'une instance d'une architecture dans la phase d'action [Guennoun, 2007]

dans ces derniers porte désormais la valeur « R ».

Dans la figure 15, l'investigateur I1, hébergé sur la machine M4, est celui qui découvre une situation critique, ce qui engendre le passage de son coordinateur *C1* et des autres investigateurs de son groupe à la phase d'action.

Par conséquent, l'événement Situation_Critique(M4) déclenche le triplet (T =Situation_Critique(m), C = ($R2_a$(M1), $R2_b$(M1)*), I = { (m, M1) }) du protocole de reconfiguration qui implique l'exécution de la combinaison C des règles de transformation décrites ci-dessous, en remplaçant M1 par M4.

$R2_a$ = (L = { N1(X_{I1} , "Inv", M1,"O+R"), N2(X_{I2}, "Inv", M2, "O+R"),
N3(X_{coord}, "Coord", "φ1", M3), N1 $\xrightarrow{\text{"O + R"}}$ N3, N2 $\xrightarrow{\text{"O + R"}}$ N3 } ;
K = { N2(X_{I2} ,"Inv", M2, "O+R") } ;
R = { N4(X_{I1} , "Inv_α", M1, "O+R"), N5(X_{coord} , "Coord", "φ2", M3),
N4 $\xrightarrow{\text{"O + R"}}$ N5 } ;
N = { } ; C = {ic})
avec ic = (N5, (*XI*,"Inv",M,"O+R"), "O+R"/"O+R", in, in)

$R2_b$ = (L = { N6(X_{I1} , "Inv_α", M1, "O+R"), N7(X_{coord} , "Coord", "φ2", M2),
N8(X_{I2} , "Inv", M3, "O+R"), N8 $\xrightarrow{\text{"O + R"}}$ N7 } ;
K = { N6(X_{I1} , "Inv_α", M1, "O+R"), N7(X_{coord} , "Coord", "φ2", M2) } ;
R = { N9(X_{I2} , "Inv", M3, "R"), N6 $\xrightarrow{\text{"O"}}$ N9, N9 $\xrightarrow{\text{"R"}}$ N7 } ;
N = { } ; C = { })

La « combinaison » ($R2_a$, $R2_b$*) permet d'exécuter les actions suivantes :

- Le remplacement du nœud N3 du coordinateur en question par un nœud N5 relatif au même coordinateur, mais en phase d'action (avec l'attribut de phase $\varphi 2$).
- Le remplacement du nœud N1 de l'investigateur critique par un nœud N4 relatif au même investigateur, mais ayant le rôle α.
- Le transfert des liens reliant les autres investigateurs à l'ancien nœud vers le nouveau nœud N5 représentant le coordinateur en phase d'action grâce à l'instruction de connexion « ic ».
- Le remplacement, à chaque application de $R2_b$, d'un nœud N8 relatif à un investigateur en phase d'exploration, par un nœud N9 du même composant, mais ayant le rôle β.
- L'adjonction, à chaque application de $R2_b$, d'un flux de données O partant de l'investigateur α vers l'investigateur β courant, et le remplacement des flux de données O+R partant de l'investigateur β courant vers le coordinateur concerné par des flux de données de type R.

3. Propriétés et contraintes temporelles

Dans les deux sections précédentes, nous avons évoqué les principes de base des grammaires de graphes, ainsi que diverses extensions servant à les enrichir en offrant des options et des contraintes supplémentaires. Cependant, toutes ses extensions ne s'intéressent qu'aux propriétés relatives à la structure des graphes et des grammaires de graphes. Malgré la diversité de ces travaux, aucune extension des grammaires de graphes par les propriétés et les contraintes temporelles n'a encore vu le jour.

« Le temps est intimement lié à notre vie biologique (cycles circadiens et annuels, vieillissement), affective (passé, présent, futur) et sociale (emplois du temps, agendas, calendriers). Il intervient dans l'expression des lois physiques. Il joue également un rôle essentiel dans de nombreux systèmes techniques. » [André, 2011]. Ainsi, nous présentons, dans ce qui suit, quelques travaux abordant le concept du temps dans le cadre de quelques domaines de recherche, à savoir, la logique temporelle [Harel et al., 1990] et les automates temporisées [Alur et Dill, 1994].

En ce qui concerne la modélisation du temps, les travaux abordés dans [Harel et al., 1990] proposent un modèle de temps discret, dans le cadre de la logique temporelle. Ce modèle énonce le temps en étant une séquence infinie d'entiers naturels qui s'incrémentent d'une manière monotone.

Les auteurs définissent le temps d'une manière sommaire par une variable t_i restreinte par les contraintes suivantes :

- Le temps ne décroît jamais : $t_i \leq t_{i+1}$ pour tout $i \in N$.
- La valeur du temps augmente : Pour tout entier n, il existe un autre entier i tel que $t_i > n$.
- La valeur du temps n'avance pas par plus qu'une unité à chaque pas (i.e., la complétude) : $t_{i+1} < t_i + 1$.

Cependant, les événements dans un processus physique n'arrivent pas toujours à un temps de valeur entière. La valeur de temps sera ainsi arrondie à un nombre entier, ce qui diminue la précision, voire l'exactitude du temps. Ceci aura un effet plus ou moins remarquable sur la fidélité du temps dans les systèmes physiques [Alur et Dill, 1994].

Nous pouvons également remarquer que la description du temps dans le cadre de ce modèle n'est pas bien formalisée. Il s'agit plutôt d'une description brève et très simpliste.

Dans les travaux présentés dans [Alur et Dill, 1994], les auteurs utilisent un modèle de temps dense dans le cadre des automates temporisés. Le temps est représenté par un ensemble dense et les événements arrivent à un temps de valeur réelle. « Les horloges de ce type sont censées suivre fidèlement le temps physique » [André, 2011].

Un automate temporisé est un automate fini, muni d'un ensemble d'horloges de valeurs réelles. À chaque transition, une réinitialisation d'une horloge peut être faite d'une manière complètement indépendante des autres horloges de l'automate. Des contraintes temporelles peuvent être ainsi affectées à la valeur d'une horloge au niveau d'une transition donnée. Ces contraintes peuvent mettre en œuvre certains besoins quantitatifs des systèmes à temps réel, à savoir, la « périodicité », le « temps limite d'une réponse », etc. [Alur et Dill, 1994].

Les auteurs présentent une description du temps plus exhaustive avec un modèle de temps dense dans lequel le domaine des valeurs du temps est décrit par un ensemble continu et infini de nombres réels positifs. Le temps est ainsi représenté par une séquence « τ » définie comme suit :

Une séquence $\tau = \tau_1\tau_2...$ est une séquence infinie de valeurs de temps strictement positives $\tau_i \in R$, respectant les contraintes suivantes :

- La monotonicité : τ croît d'une manière strictement monotone : $\tau_i < \tau_{i+1}$ pour tout $i \geq 1$.
- La progression : pour tout $t \in R$, il existe un entier $i > 0$ tel que $\tau_i > t$.

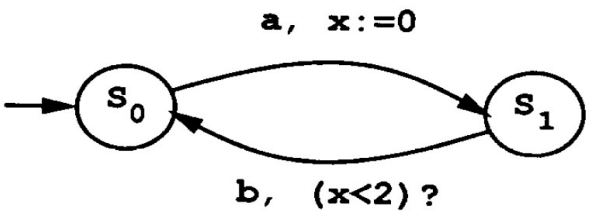


Figure 16 : Une table de transitions d'un automate temporisé

La figure 16 présente un exemple de transitions d'un automate fini temporisé dans lequel une horloge x est mise à 0 au niveau de la transition partant du premier état s_0 vers l'état s_1 suite à la lecture du caractère « a ». Cette action est décrite par « x:=0 ». La transition partant de l'état s_1 vers s_0 à travers le caractère « b » est dotée d'une contrainte temporelle « (x<2) ? » exigeant que cette transition ne peut être effectuée qu'à partir de deux unités de temps de l'occurrence de la première transition.

Dans l'exemple précédent, une seule horloge a été utilisée. Dans un automate temporisé, plusieurs horloges peuvent êtres employées et manipulées indépendamment les unes des autres. Dans l'exemple illustré par la figure 17, deux horloges x et y sont remises à 0 respectivement au niveau des transitions partant de s_0 vers s_1, et de s_1 vers s_2. La transition partant de s_2 vers s_3 ne dépend que de la valeur de l'horloge x, et celle partant de s_3 vers s_0 ne dépend que de la valeur de l'horloge y.

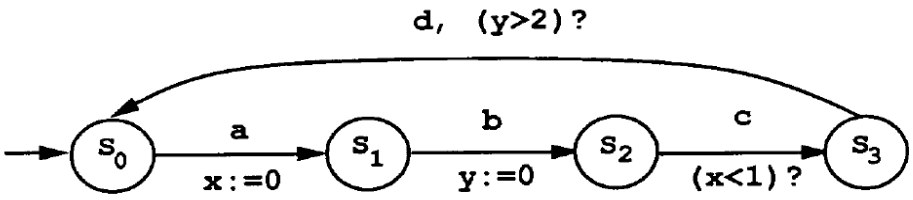


Figure 17 : Une table de transitions d'un automate à deux horloges

Une contrainte temporelle consiste ainsi en une condition booléenne comparant la valeur d'une horloge avec une constante positive apprenant à l'ensemble Q des nombres rationnels. Elle est définie comme suit :

> Pour un ensemble X d'horloges, Φ(X) représente l'ensemble de contraintes temporelles δ définies sur X, tel que :
>
> $\delta := x \leq c \mid x \geq c \mid x < c \mid x > c$, x étant une horloge dans X, et c une constante dans Q.

Bien que la prise en compte des propriétés temporelles n'ait pas été étudiée dans le cadre des grammaires de graphes, nous avons présenté, dans cette section, quelques travaux antérieurs traitant de cette problématique dans d'autres domaines de recherche.

Chapitre 2 : ***Reconfiguration temporisée des architectures***

Dans le chapitre précédent, nous avons montré certaines extensions sur les grammaires de graphes, abordées dans [Guennoun, 2007] et [Bouassida *et al.,* 2008]. Les auteurs ont recouru à ces notions afin de pouvoir modéliser l'aspect évolutif dans les systèmes collaboratifs dynamiques. Nous avons vu que certains systèmes, tel que celui des Opérations d'Interventions d'Urgence (OIU), subissent des mutations plus ou moins fréquentes et peuvent ainsi avoir un degré de stabilité faible. Cette faible stabilité est due, entre autres, aux « changements des objectifs applicatifs », aux pannes que les composants du système peuvent subir, et aux exigences des qualités de services [Guennoun, 2007].

Les architectures traitées par ces travaux évoluent donc à travers le temps. L'auto-reconfiguration de ces architectures en représente ainsi la problématique primordiale. Cependant, ce méta-modèle traitant de l'adaptabilité structurelle des architectures ne s'intéresse pas aux contraintes temporelles que peut avoir un système collaboratif. En effet, l'orchestration de la collaboration entre les composants d'un tel système peut dépendre de certaines contraintes relatives au temps. Le système OIU, en l'occurrence, requiert par exemple des délais nécessaires à l'installation et à la mise en marche de certains composants, un temps de vérification de l'état d'une connexion entre deux composants, ainsi qu'une fréquence maximale des passages d'une configuration à une autre. Par conséquent, le fait d'introduire ou de supprimer un composant, et celui d'activer ou de désactiver un lien de collaboration, doivent être restreints à certaines conditions relatives aux temps.

Ces contraintes ont pour but d'assurer la stabilité du système et d'échapper aux anomalies engendrées par les actions de reconfiguration occurrentes à des instants considérés comme critiques pour ledit système. Ces contraintes permettent, outre cela, de garantir la consistance de la journalisation des événements de reconfiguration et des actions de reconfiguration subies par l'architecture à un instant donné.

Dans ce chapitre, nous présentons une extension du méta-modèle décrit par [Guennoun, 2007] et [Bouassida *et al.,* 2008], prenant en compte les propriétés et les contraintes temporelles spécifiques à la reconfiguration des architectures dynamiques. Nous définissons ainsi un cadre formel traitant de la reconfiguration temporisée, se basant sur les concepts des grammaires de graphes et des règles de transformation, et s'inspirant des aspects temporisés présentés dans le chapitre précédent, tout en argumentant certains choix abordés entre diverses propositions. Nous projetons au fur et à mesure les nouveaux concepts sur les besoins temporels requis au sein de l'architecture du système OIU.

1. Modélisation du temps

Dans le chapitre précédent, nous avons évoqué deux formalismes de représentation du temps. Le premier, proposé par [Harel et al., 1990], aborde un modèle de temps discret qui présente le temps par une séquence infinie de nombres entiers positifs. Quant au deuxième, présenté par [Alur et Dill, 1994], il adopte un modèle de temps dense décrit par un ensemble continu et infini de nombres réels positifs. Nous optons pour la deuxième représentation, étant donné qu'elle assure l'exactitude des valeurs temporelles envisagées dans uns système physique réel.

Afin d'obéir auxdits besoins, une valeur temporelle est portée par une horloge qui suit fidèlement le temps physique. Une horloge est définie dans un domaine séquentiel de temps (DST) décrit comme suit :

> $d(\tau) = [\tau_0, \tau_\infty]$, $d(\tau)$ est un DST. τ représente une séquence infinie de valeurs temporelles réelles et positives τ_i, et respecte les conditions suivantes :

- C1 : La « monotonicité » : τ croît d'une manière strictement monotone : $\tau_i < \tau_{i+\varepsilon}$ pour tout $i \geq 0$.
- C2 : La « progression » : Pour tout réel positif t, il existe un autre réel positif τ_i, tel que $\tau_i > t$.

Afin de modéliser la vérification des propriétés temporelles, il est nécessaire de concevoir les contraintes temporelles sous une représentation formelle. Ainsi une contrainte δ représente une condition booléenne correspondant à une comparaison entre la valeur d'une horloge x avec une constante positive c appartenant à l'ensemble R des nombres réels. δ est ainsi définie comme suit :

$$\delta (x, c) := x \leq c \mid x \geq c \mid x < c \mid x > c \mid x = c$$

Ce cadre formel sera adopté tout le long des concepts que nous abordons dans les sections suivantes. En effet, toutes les horloges d'un système collaboratif suivent ainsi la définition formelle du temps dense, et les contraintes temporelles obéissent désormais au format décrit ci-dessus.

2. Reconfiguration temporisée

2.1. Propriétés architecturales temporelles

Nous avons présenté dans le chapitre précédent, le concept de « propriétés architecturales » relatives à un système collaboratif. Nous avons aussi montré l'avantage de ce concept dans la définition des contraintes globales d'une architecture, telles que l'interdépendance requise entre deux composants et la participation obligatoire d'un composant-type à une collaboration-type, etc.

Les propriétés architecturales, comme définies dans [Guennoun, 2007], sont modélisées à travers un ensemble de règles de transformation réduites de type (L, N). Nous avons également évoqué le concept de propriétés positives et de propriétés négatives qui permettent de donner plus de sens à ces règles, et nous en avons cité des exemples illustratifs expliquant ces concepts (cf. Chapitre 1, section 2.3).

Nous comptons, dans ce qui suit, étendre ce formalisme afin de pouvoir définir les propriétés temporelles architecturales. Ces propriétés permettent de spécifier une horloge globale initialisée dès le début du fonctionnement du système, servant comme repère pour certaines règles de reconfiguration, ainsi qu'une contrainte relative à la fréquence des actions de reconfiguration d'une architecture qui sont déclenchées suite à l'occurrence d'un événement de reconfiguration spécifique.

2.1.1. Définition des propriétés architecturales

Notre proposition consiste à faire intégrer les propriétés temporelles globales au sein du concept des propriétés architecturales. Pour ce faire, ces dernières sont désormais représentées par un ensemble PA = { PS, PT }. PS correspond à l'ensemble des propriétés architecturales, précédemment rapportées, relatives à la structure de l'architecture à instancier. PT représente l'ensemble des propriétés temporelles architecturales abordées dans le cadre de l'extension proposée.

PT est représentée par un triplet (H, T, x_T), où :

- H correspond à l'horloge globale du système ; elle est déclenchée au début du fonctionnement de ce dernier,
- T représente la « période » qui permet de mettre une contrainte sur la fréquence des actions de reconfiguration du système, et

- x_T est une « horloge périodique » déclenchée en même temps que H, et qui est remise à zéro à chaque reconfiguration du système.

Dans le cadre du système d'opérations d'interventions d'urgence, la collecte des premiers rapports de synthèse établis par les coordinateurs attachés à l'unique contrôleur requiert un temps minimal estimé à six secondes au maximum. En effet, ce temps est estimé en fonction du temps de transmission des données à travers le réseau, et du temps de calcul nécessaire à l'élaboration des premiers rapports de synthèse par les coordinateurs. Pour ce faire, nous avons attribué aux propriétés architecturales temporelles une période T de valeur égale à six secondes. La propriété PT = (H := 0, T = 6, x_T := 0) caractérise ce besoin. Elle déclare l'initialisation d'une horloge globale déclenchée au début de la mise en marche du système, c-à-d, lors de la création du contrôleur unique, la définition d'une période T = 6 exigeant qu'une action de reconfiguration de l'architecture ne peut être effective qu'après l'écoulement de 6 secondes à partir de l'instant de la dernière reconfiguration, et l'utilisation d'une horloge périodique x_T déclenchée au début du système et réinitialisée à chaque reconfiguration. Cette horloge est vérifiée à chaque reconfiguration du système afin de garantir la fréquence conçue.

Le concepteur de l'architecture dynamique d'un système a le choix de spécifier, ou non, une contrainte sur la fréquence des actions de reconfiguration. Si une telle condition n'est pas requise, nous pouvons attribuer la valeur zéro à la période T. Aucune contrainte n'est ainsi posée sur la fréquence.

2.1.2. Vérification des propriétés temporelles architecturales

Afin de pouvoir respecter les contraintes temporelles architecturales, il faut concevoir un dispositif permettant de vérifier les propriétés temporelles déclarées au niveau des propriétés architecturales du système collaboratif. Pour ce faire, nous avons proposé deux options. La première consiste à étendre le concept de protocoles de reconfiguration (cf. Chapitre 1, section 2.2). La deuxième solution est inspirée de l'extension des règles de transformation par l'approche NCE (cf. Chapitre 1, section 1.3.2). Nous comparons, dans ce qui suit, les deux possibilités, à travers des exemples d'application simples, afin d'argumenter le choix de l'approche abordée. Nous en définissons ensuite le cadre formel correspondant.

Les protocoles de reconfiguration sont représentés par un ensemble de triplets (T, C, I). Son extension repose sur l'adjonction d'une partie C_T qui regroupe des contraintes posées sur les propriétés temporelles architecturales définies par PT.

$$PRT_1 = \{ (T = E(p1, p2),\ C_T = (H < 10, ø)\ ,\ C = r1(X1, X2)\ ;\ r2(X2),\ I = \{ (p1, X1), (p2, X2), \}) \}$$

Le protocole étendu PRT_1 requiert que l'événement de reconfiguration E ne pourra engendrer les transformations décrites dans la partie C que si la valeur de l'horloge globale H est inférieure à 10 unités de temps. Si cette condition est vérifiée, toutes les règles de transformation de la combinaison C sont exécutées selon le formalisme correspondant.

La deuxième option consiste à étendre la structure des règles de transformation d'une manière indépendante des événements qui peuvent déclencher leur exécution. Une règle de transformation de base étendue par le concept des contraintes temporelles est ainsi de la forme ((L, R), C_T), où L et R gardent la même signification précédemment évoquée (cf. Chapitre 1, section 1.2.1).

Les règles r1 et r2, décrites ci-dessous, suivent cette approche :

$r1 = \big((L = \{ N1(X1), N2(X2), N1 \longrightarrow N2 \}, R = \{ N2(X2) \}), (H < 10, ø) \big)$

$r2 = \big((L = \{ N2(X2) \}, R = \{ N2(X2), N3(X3), N3 \longrightarrow N2 \}), (H < 10, ø) \big)$

L'application du protocole de reconfiguration PR_1 = { (T = E(p1, p2), C = r1(X1, X2) ; r2(X2), I = { (p1, X1), (p2, X2), }) }, avec les règles de transformation étendues, implique la vérification de la contrainte temporelle relative à la valeur de l'horloge globale avant l'exécution de chacune de ces règles. Cependant, dans ce cas, une anomalie pourra avoir lieu. En fait, chaque règle ne peut être exécutée que si l'horloge H n'a pas encore atteint la valeur 10, et la combinaison C stipule que l'exécution de la règle r2 n'est effective qu'après l'exécution de r1. Par conséquent, si un événement de type E survient par exemple lorsque la valeur de H est égale à 9, et que l'exécution de r1 dure 2 unités de temps, r2 ne sera pas donc exécutée. Ce genre d'anomalies pourra avoir des effets considérables au niveau de la fidélité de l'architecture d'un système.

Partant de cette analyse comparative, nous optons pour la première solution, s'agissant d'étendre le concept de protocole de reconfiguration. Une présentation formelle d'un protocole de reconfiguration temporisé est décrite comme suit :

$PRT = \{ (T, C_T, C, I)^* \}$.
$C_T = \big((\delta(H, c) \mid ø), (\delta(x_T, T) \mid ø) \big) \mid ø$, H, T et x_T étant respectivement l'horloge globale, la période de reconfiguration et l'horloge périodique, qui ont été décrites dans la section précédente, et c une constante.

Ce protocole est ainsi constitué par un ensemble de quadruplets de la forme (T, C_T, C, I) qui assurent :

- la gestion des événements de reconfiguration et la déclaration des paramètres à introduire, à travers la partie T,
- la vérification des propriétés et des contraintes temporelles architecturales et la réinitialisation de l'horloge périodique x_T, à travers l'élément C_T,
- la combinaison et l'ordonnancement des règles de transformation à exécuter, à travers la partie C, et
- la gestion des règles d'instanciation permettant d'affecter la valeur des paramètres transportés par l'événement de reconfiguration à certaines variables dans les règles de transformation décrites dans C, à travers la partie I.

Suivant cette définition, le concepteur de l'architecture garde la possibilité de spécifier des actions de reconfiguration qui ne dépendent pas de la durée du fonctionnement du système ou même de la fréquence des actions de reconfiguration. La contrainte $C_T = (H < 10, ø)$ du protocole PRT_1, par exemple, ne pose aucune contrainte sur la fréquence des actions de reconfiguration. Une autre contrainte de la forme $C_T = (ø, x_T > T)$ obéit à la fréquence requise mais ne s'intéresse pas à la durée de fonctionnement du système à l'instant de l'occurrence de l'événement de reconfiguration correspondant. Quant à la contrainte $C_T = ø$, elle est relative à une action de reconfiguration totalement indépendante des contraintes temporelles.

Nous avons pu exprimer, dans la section précédente, la déclaration des propriétés temporelles architecturales au sein du cas d'étude des Opérations d'Interventions d'Urgence. Cependant, Les quadruplets du protocole de reconfiguration temporisée sont ceux qui sont à l'origine de l'exécution

des règles de transformation. Nous présentons ainsi, dans ce qui suit, le concept des règles de transformation temporisées, avant de continuer dans le même cadre d'application abordant ces deux concepts.

2.2. Règles de transformation temporisées

Dans le premier chapitre, nous avons évoqué l'état de l'art des différentes approches de transformation de graphes abordant une variété de structures de règles de transformation. Une combinaison de ces approches a été utilisée dans le cadre du méta-modèle de l'adaptation structurelle automatique (cf. Chapitre 1, section 2). Nous comptons, dans ce qui suit, présenter un formalisme servant comme extension des règles de transformation proposées dans le cadre de ce méta-modèle. Il s'agit en fait, de manipuler une horloge locale à un nœud ou à un arc, afin de traiter les différentes contraintes temporelles relatives à un nœud ou à un arc spécifique, dans le cadre des règles de transformation de graphes.

Dans les travaux abordés dans [Guennoun, 2007] et [Bouassida *et al.,* 2008], la reconfiguration d'une architecture est modélisée par le moyen de règles de transformation de la forme (L, K, R, N, C) combinant l'approche DPO avec les conditions d'application négative (NAC), et les instructions de connexion étendues (edNCE). Dans ces règles, les labels d'un nœud sont énumérés, après son nom, entre deux parenthèses. La représentation "N1($X1_{Cont}$, "Cont" , M1)", par exemple, désigne un nœud ayant comme nom N1, et comme label la liste des attributs situés entre deux parenthèses. En revanche, un arc est représenté par une flèche au-dessus de laquelle sont énumérées les étiquettes correspondantes. Par exemple, la représentation "N1 $\xrightarrow{\text{"O + R"}}$ N2" désigne un arc partant de N1 vers N2 et ayant "O+R" comme étiquette.

Notre approche consiste à étendre ces représentations au sein des règles de transformation de façon à adopter les propriétés et les contraintes temporelles spécifiques à chaque nœud ou arc. Ainsi, un nœud est représenté par un couple (S, T), où S décrit la liste des labels structuraux classiques, et T représente la propriété temporelle correspondante. Les étiquettes des arcs ont la même structure que celle des labels des nœuds. Pour bénéficier de la représentation souple évoquée ci-dessus, un arc reliant N1 à N2 est ainsi de la forme N1 $\xrightarrow[T]{S}$ N2, où S et T gardent la même signification que dans les nœuds.

Pour ce qui est des instructions de connexion edNCE, qui sont de la forme (n, p/q, δ, d, d'), les arcs insérés suite à l'application de ces instructions doivent être, eux aussi, représentés par la structure conventionnelle abordée ci-dessus. Pour ce faire, la structure des instructions de connexion doit comporter également l'information relative à la propriété temporelle correspondant à l'arc inséré. Ainsi, une instruction de connexion est désormais représentée par un couple (S, T), où S regroupe les éléments classiques d'une instruction edNCE suivant la forme (n, p/q, δ, d, d'), et T représente la propriété temporelle correspondante.

Jusqu'ici, nous avons présenté la structure globale des nœuds, des arcs et des instructions de connexion dans les règles de transformation temporisées. La partie S regroupe, dans tous les éléments de ce type de règles, les concepts classiques précédemment définies. Nous abordons, dans ce qui suit, les différentes formes que peut porter la partie T ainsi que la signification relative à chaque contexte. En effet, selon la partie de la règle de transformation, T aura comme fonctionnalité, ou bien de vérifier une condition relative à une contrainte temporelle locale (i.e., dans les parties L, K et N), ou bien d'initialiser une horloge locale x relative à un nœud ou à un arc (i.e., dans la partie R et dans les instructions de connexion de la partie C).

- Dans la partie L : Le concepteur a la possibilité de poser, ou non, une contrainte temporelle. T est ainsi représentée comme suit :

  $T := (\delta\ (x, c)^*) \mid \text{ø}, c \in R^+$

- Dans la partie K : Cette partie ne contient que des éléments qui existent dans la partie L. Les propriétés temporelles des nœuds et des arcs de K sont donc déjà vérifiées. Pour échapper à toute ambiguïté, T est définie dans K comme suit :

  $T := \text{ø}$

- Dans la partie R : Le concepteur a le choix d'affecter, ou non, une horloge déclenchée à la création du nœud ou de l'arc inséré. Dans ce cas, T est définie comme suit :

  $T := (x:=0) \mid \text{ø}$

- Dans la partie N : A l'instar de la partie L, le concepteur a le choix de poser, ou non, une contrainte temporelle sur un nœud ou un arc bien déterminé. Dans ces derniers, T est représentée d'une manière identique à ce qui est fait dans la partie L :

  $T := (\delta\ (x, c)^*) \mid \text{ø}, c \in R^+$

- Dans les instructions de connexion de la partie C : Il s'agit de l'introduction de nouveaux arcs. Comme dans la partie R, le concepteur peut attribuer, ou non, une horloge aux arcs concernés. T est ainsi définie comme suit :

  $T := (x:=0) \mid \text{ø}$

Dans la section 2.1.1, nous avons défini les propriétés temporelles architecturales PT = (H := 0, T = 6, $x_T := 0$) limitant la fréquence des actions de reconfiguration au sein du système d'Opérations d'Interventions d'Urgence. Nous continuons dans ce qui suit à présenter l'application des nouveaux concepts (i.e., les protocoles de reconfiguration temporisés et les règles de transformation temporisées) au sein du même cas d'étude. Nous présentons ainsi un scénario de la découverte d'une situation critique engendrant le passage de l'architecture de la phase d'exploration à la phase d'action.

Prenons l'instance architecturale du système OIU illustrée dans la figure 14 (cf. Chapitre 1, section 2.4). Étant en phase d'exploration, l'investigateur I1 découvre la situation critique et sa section, qui regroupe son coordinateur C1 et les autres investigateurs supervisés par C1, passe à la phase d'action. Ce type de reconfiguration exige que le système soit parfaitement installé et qu'il ait été bien en marche depuis plus de 40 secondes. Par conséquent, la valeur de l'horloge globale doit être supérieure à 40. Cependant, considérant l'aspect urgent de l'opération, la fréquence des actions de reconfiguration n'est pas prise en compte, afin de pouvoir passer immédiatement en phase d'action. Par conséquent, l'événement Situation_Critique(M4) déclenche le quadruplet (T =Situation_Critique(m), C_T = (H > 40, ø), C = ($R2_a$(M1), $R2_b$(M1)*), I = { (m, M1) }) du protocole de reconfiguration permettant la vérification des contraintes temporelles relatives au passage à une situation critique, engendré par l'investigateur I1, avant d'exécuter les transformations décrites par la partie C.

Par ailleurs le coordinateur C1 et l'investigateur I1 doivent être fonctionnels, respectivement depuis 25 et 10 secondes au minimum, pour vérifier qu'ils sont considérés comme aptes à parfaire une tâche aussi importante.

Les règles exécutées suite à l'occurrence de cet événement de reconfiguration sont expliquées dans

la section 2.4 du premier chapitre. La règle $R2_a$ consiste à substituer le nœud relatif à l'investigateur critique par un nœud représentant le même composant et ayant comme rôle α, à remplacer le nœud relatif au coordinateur critique par un nœud représentant le même composant et qui est en phase d'action, et à réinsérer les liens de collaboration d'une manière identique à l'ancienne description. Quant à la règle $R2_b$, son application multiple implique le remplacement des nœuds des autres investigateurs de la section par des nœuds relatifs aux mêmes composants et ayant comme rôle β, et le remplacement des arcs représentant les anciens flux de données par des flux relatifs à une section en phase d'action. Ainsi, ces deux règles sont décrites comme suit :

$R2_a$ = (L = { N1((X_{I1}, "Inv", M1,"O+R"), (x > 10)), N2((X_{I2}, "Inv", M2, "O+R"), ø), N3((X_{coord}, "Coord", "φ1", M3), (x > 25)), N1 $\xrightarrow[ø]{\text{"O + R"}}$ N3, N2 $\xrightarrow[ø]{\text{"O + R"}}$ N3 } ;
K = { N2((X_{I2},"Inv", M2, "O+R"), ø)} ;
R = { N4((X_{I1}, "Inv$_\alpha$", M1, "O+R"), (x := 0)), N5((X_{coord}, "Coord", "φ2", M3), (x := 0)), N4 $\xrightarrow[x:=0]{\text{"O + R"}}$ N5 } ;
N = { } ; C = {ic})
avec ic = ((N5, (XI,"Inv",M,"O+R"), "O+R"/"O+R", in, in), (x := 0))

$R2_b$ = (L = { N6((X_{I1}, "Inv$_\alpha$", M1, "O+R"), ø), N7((X_{coord}, "Coord", "φ2", M2), ø), N8((X_{I2}, "Inv", M3, "O+R"), ø), N8 $\xrightarrow[ø]{\text{"O + R"}}$ N7 } ;
K = { N6((X_{I1}, "Inv$_\alpha$", M1, "O+R"), ø), N7((X_{coord}, "Coord", "φ2", M2), ø) } ;
R = { N9((X_{I2}, "Inv", M3, "R"), (x := 0)), N6 $\xrightarrow[x:=0]{\text{"O"}}$ N9, N9 $\xrightarrow[x:=0]{\text{"R"}}$ N7 } ;
N = { } ; C = { })

Au niveau de la règle $R2_a$, l'horloge locale du nœud N1, correspondant à l'investigateur critique, doit avoir une valeur supérieure à 10 secondes. Quant au coordinateur de la section, son horloge doit avoir une valeur supérieure à 25 secondes. Aucune contrainte temporelle n'est posée sur le reste des nœuds et des arcs. Par ailleurs, des horloges locales seront affectées aux nouveaux nœuds et arcs insérés suivant les parties R et C des deux règles.

Chapitre 3 : ***Outil de transformation temporisée des graphes***

Nous avons présenté, dans le chapitre précédent, notre approche concernant l'extension d'un méta-modèle, traitant de l'adaptation structurelle des architectures dynamiques, par les propriétés et contraintes temporelles que peuvent requérir les systèmes collaboratifs, tel que le système OIU. Nous comptons ainsi, dans ce qui suit, présenter aussi bien un tour d'horizon des principaux outils de transformation de graphes, que l'application des concepts formels de notre approche au sein de l'outil GMTE (Graph Matching and Transformation Engine).

1. Outils de transformation de graphes

Plusieurs outils de transformation de graphes ont vu le jour et divers travaux d'amélioration et d'optimisation sont également en cours. Parmi ces outils, nous pouvons citer l'outil appelé AGG (Attributed Graph Grammar system) [Taentzer, 2004], considéré comme « un langage visuel basé sur les règles ». Il offre la possibilité de modéliser et de mettre en œuvre les données complexes des architectures à travers les graphes [Site 1]. La modélisation de l'aspect évolutif des systèmes est traitée par une approche algébrique de transformation de graphes assurée par des règles de transformation adoptant les conditions d'application négative (NAC). AGG offre à la fois des éditeurs graphiques pour la représentation des graphes et des règles de transformation, et des éditeurs de texte pour la spécification de ces concepts par le moyen d'un code écrit en JAVA. En définissant un graphe de départ et un scénario d'actions de transformations décrit par un ensemble ordonné de règles de transformation, l'utilisateur peut simuler de diverses combinaisons des architectures évolutives. En outre, grâce au fait que cet outil est implémenté en langage JAVA, AGG peut être facilement être incorporé au sein des applications nécessitant l'intégration de méthodes relatives à un moteur de transformation de graphes [Baresi et Heckel, 2002]. Les développeurs de AGG visent également à faire intégrer cet outil dans le domaine de l'intelligence artificielle vu son approche orientée vers les règles [Site 1].

Un autre outil de transformation de graphes, appelé PROGRES (PROgrammed Graph REwriting Systems) [Shürr et al., 1999], est plus orienté vers le traitement des problématiques du génie logiciel. En fait, PROGRES est considéré comme un environnement de modélisation d'architectures, intégrant plusieurs outils, et permettant aux utilisateurs de concevoir leurs artefacts. Cet environnement fournit des éditeurs graphiques et des éditeurs textuels munis d'un interpréteur, aussi bien syntaxique que sémantique, conforme à un langage spécifique de description des graphes attribués, de la transformation des graphes, et d'autres aspects de description des architectures évolutives [Baresi et Heckel, 2002]. Outre la combinaison de diverses approches de transformation des graphes, un des principaux apports de cet environnement est son caractère muti-langage. Les spécifications peuvent en effet être traduites en langage C, Java, Modula 2, etc.

Quant à GMTE, c'est un moteur de transformation de graphes implémenté en C++. Sa particularité la plus importante réside dans le fait qu'il utilise une large combinaison d'approches théoriques de transformation de graphes, ainsi que de nouvelles extensions et de diverses variétés abordées sur la description, la comparaison et la transformation des graphes. Cet outil se distingue également par une performance remarquable au niveau des coûts de la recherche d'homomorphisme (ou *matching*), même dans les cas les plus extrêmes. GMTE peut être intégré dans des programmes écrits en C++ comme étant une bibliothèque C++ fournissant une interface d'invocation.

Étant donné les avantages que garantit ce dernier outil, nous avons choisi de mettre en application notre approche en l'étendant par l'intégration des propriétés et des contraintes temporelles. Ainsi, nous exposons dans ce qui suit, une présentation plus détaillée sur l'outil GMTE, ainsi qu'une vue

conceptuelle de l'extension que nous comptons lui apporter.

2. Présentation de l'outil GMTE

Cet outil manipule des graphes décrits dans des fichiers XML sous le format conventionnel GraphML (Graph Markup Language). Les règles de transformation sont également représentées dans un fichier XML sous un format spécifique. Quant aux diverses fonctionnalités qu'assure cet outil, elles sont hiérarchiquement organisées sous forme de couches. Chaque couche comprend un ou plusieurs API qui fournissent des méthodes spécifiques au niveau hiérarchique correspondant et à la nature du service visé. La figure 18 schématise les différentes couches interagissant au sein de GMTE.

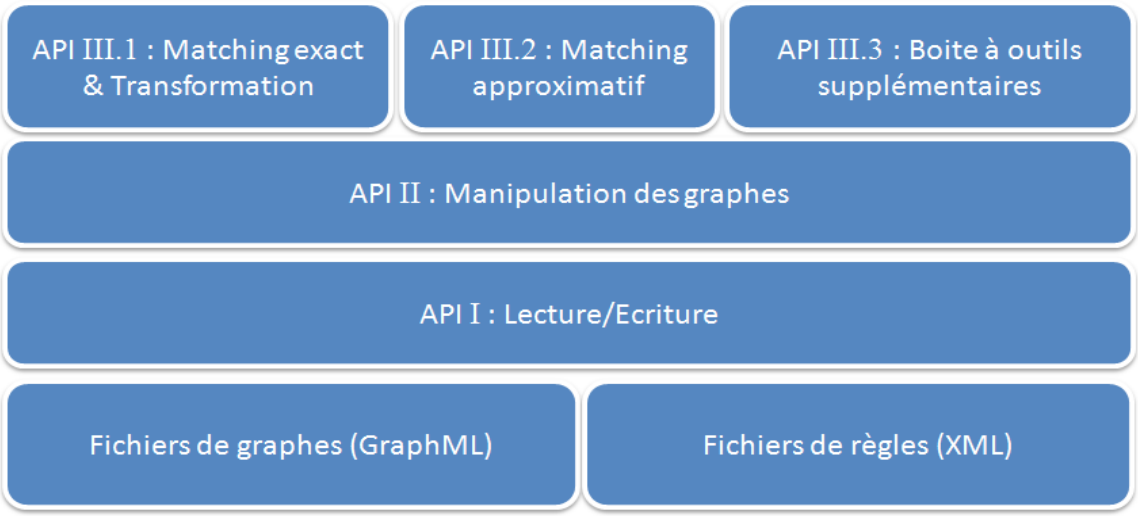
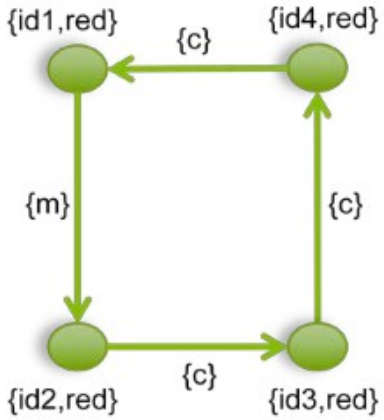


Figure 18 : La structure en couches de l'outil GMTE

Nous pouvons remarquer à travers la figure précédente que GMTE ne se limite pas, contrairement aux autres outils et environnements de transformation de graphes, à l'application de règles de transformations sous une approche spécifique, mais offre aussi une variété de fonctionnalités supplémentaires. Dans ce qui suit, nous nous limitons à présenter les composants nécessaires à l'intégration de notre approche au sein de cet outil.

2.1. Fichiers de graphes

Le standard de présentation GraphML a été adopté dans la modélisation des graphes. GraphML est un format de présentation conventionnel basé sur le format de description des données XML. Ce langage sert à décrire d'une manière simple les propriétés structurelles des graphes. Un document GraphML est ainsi constitué d'une balise GraphML unique décrivant un graphe à travers une liste de balises représentant les nœuds et d'autres représentant les arcs. La figure 19 illustre un graphe orienté, étiqueté et multi-labellé ainsi que sa description sous le standard GraphML.



a. Représentation schématique du graphe G1

```
<?xml version="1.0" ?>
<graphml>
    <graph id="0">
        <!-- GraphML -->
        <node id="n0">
            <data>id1</data>
            <data>red</data>
        </node>
        <node id="n1">
            <data>id2</data>
            <data>red</data>
        </node>
        <node id="n2">
            <data>id3</data>
            <data>red</data>
        </node>
        <node id="n3">
            <data>id4</data>
            <data>red</data>
        </node>
        <edge source="n0" target="n1">
            <data>m</data>
        </edge>
        <edge source="n1" target="n2">
            <data>c</data>
        </edge>
        <edge source="n2" target="n3">
            <data>c</data>
        </edge>
        <edge source="n3" target="n0">
            <data>c</data>
        </edge>
    </graph>
</graphml>
```

b. Représentation du graphe G1 sous GraphML

Figure 19 : Représentation des graphes dans GMTE [GMTE R.M.]

2.2. Fichiers de règles

En ce qui concerne les règles de transformation de graphes, GMTE utilise une combinaison des approches DPO et NAC ainsi que des instructions de connexion de type edNCE. Ces règles, qui sont de la forme (L, K, R, NAC, C) sont stockées dans des fichiers XML. Une instruction de connexion appartenant à C est de la forme (m, p/q, δ, x, d, d'). Cette structure est identique à celle décrite dans le premier chapitre (i.e. la forme (n, p/q, δ, d, d')), sauf pour l'élément x qui correspond désormais à l'élément n de la description classique, et pour l'élément m qui a été ajouté. En effet, une instruction edNCE classique (cf. Chapitre 1, Section 1.4) correspond à une règle de transformation de la forme (X, D) où X représente un nœud non terminal unique. Étant donné que GMTE ne manipule que des nœuds terminaux, le sous-graphe correspondant à (L \ K) peut contenir plusieurs nœuds. D'où l'utilité de l'élément m, qui sert à identifier, pour une instruction de connexion spécifique, le nœud, appartenant au sous-graphe à supprimer, dont les nœuds voisins seront rattachés au nœud n, suivant les paramètres de l'instruction de connexion concernée. Une règle simple, i.e., sans prise en compte de l'extension edNCE, est représentée par un une structure composée de quatre zones définies comme suit :

- La zone *Inv* : Cette zone caractérise la partie K de la règle de transformation. Si plusieurs occurrences du graphe spécifié par cette zone existent dans le graphe de départ, une de ces occurrences est choisie d'une manière arbitraire. Lors de l'application de la règle, le sous-graphe correspondant ne subit aucune modification.
- La zone *Abs* : Cette zone est relative à la partie NAC de la règle de transformation. L'applicabilité de la règle requiert ainsi l'absence d'une occurrence du graphe spécifié par cette zone dans le graphe de départ.
- La zone *Del* : Cette zone correspond au graphe spécifié par (L \ K) qui sera supprimé suite à l'application de la règle.
- La zone *Add* : Cette zone spécifie les nœuds et les arcs à insérer. Elle correspond au graphe spécifié par (R \ K) [GMTE R.M.].

En adoptant l'extension edNCE, la représentation de la règle de transformation est étendue par des balises représentant chacune une instruction de connexion.

2.3. Couche de lecture/écriture des fichiers XML

Cette couche fournit les services de plus bas niveau, à travers une interface permettant la manipulation des fichiers des graphes et des fichiers des règles de transformation qui seront utilisés par les couches supérieures. Les fonctionnalités principales assurées par cette API sont :

- la lecture et la conversion des graphes de départ décrits sous le format GraphML, en un objet de type *Graphe*.
- la lecture des règles de transformation écrits en XML et leur conversion en un objet de type *Règle, et*
- la sauvegarde des graphes résultant de l'application des règles de transformation dans un fichier XML suivant la description GraphML [GMTE Ref 2.0].

2.4. Couche de manipulation des graphes

Cette couche fournit des méthodes permettant la manipulation directe des graphes chargés par la couche de l'API I. Ces manipulations concernent l'adjonction, la modification et la suppression des nœuds et/ou des arcs, que ce soit lors de l'étape de conception initiale des graphes et des règles de transformation, ou suite à l'application d'une règle de transformation [GMTE Ref 2.0].

2.5. API de matching exact et de transformation

Cette API est responsable de l'application des règles de transformation. Une fois une règle est chargée et convertie vers un objet *Règle*, l'API assure la recherche d'un homomorphisme entre le graphe spécifié par ladite règle et le graphe de départ. Vérifiant l'applicabilité de cette règle, cette interface applique, sur le graphe de départ, la transformation décrite en invoquant les méthodes de manipulation nécessaires. Le graphe résultant de la transformation sera enregistré dans un nouveau fichier XML. Cette API fournit également d'autres options telles que l'application multiple des règles, i.e., l'application d'une règle autant de fois qu'elle est applicable, et l'application d'une séquence de règles de transformation décrites dans un seul fichier de règles [GMTE Ref 2.0].

3. Extension de GMTE

Afin de mettre en œuvre les concepts théoriques de notre contribution, nous comptons, dans cette section, présenter une vue globale sur l'extension potentielle que nous comptons apporter à l'outil de transformation de graphes GMTE. Ce travail, qui sera repris à la deuxième phase du stage, sera réparti suivant les tâches suivantes :

- L'extension de la structure des règles de transformation décrites dans les fichiers de règles et de la structure de description statique des graphes dans les fichiers de graphes, afin de supporter la notion de contraintes temporelles et l'affectation des horloges.
- L'extension des couches préexistantes, traitant de la lecture/écriture des graphes et des règles de transformation, de la manipulation des graphes chargés en mémoire, et de l'application des règles de transformation, afin de les adapter aux nouvelles structures des graphes et des règles.
- Le développement d'une couche de haut niveau permettant la définition des propriétés temporelles architecturales ainsi que la vérification de ces dernières à travers les protocoles de reconfiguration temporisés.
- Le développement d'une couche de plus haut niveau permettant, d'une part, la description d'un scénario d'exécution en spécifiant le graphe de départ et une séquence temporisée d'événements de reconfiguration, et, d'autre part, la définition des services nécessaires à une simulation en temps réel du scénario spécifié.

Enfin, pour illustrer ces différentes extensions, nous présentons dans la figure 20 une vue architecturale de l'évolution potentielle de l'outil de transformation GMTE.

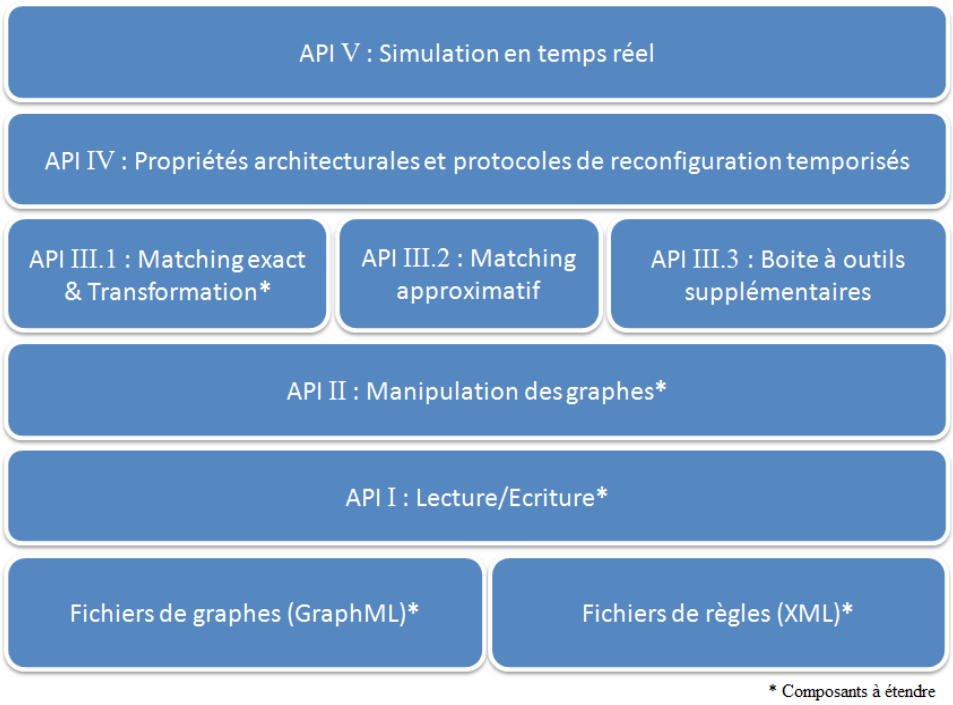


Figure 20 : Structure en couches de l'outil GMTE mettant en œuvre les contraintes temporisées

Conclusion et perspectives

Nous avons présenté dans ce mémoire l'ensemble des travaux réalisés pendant mon stage, s'intéressant à la reconfiguration dynamique des architectures au sein des systèmes collaboratifs, prenant en compte les propriétés et les contraintes temporelles.

Nous avons fait dans le premier chapitre un tour d'horizon des différentes thématiques de base de notre sujet. En premier lieu, nous avons évoqué une étude théorique des concepts relatives aux différentes approches de transformation de graphes et nous avons présenté le concept des grammaires de graphes ainsi qu'une variété de ses extensions. Ces concepts formels sont considérés comme la base des travaux, qui ont permis de construire un méta-modèle traitant de l'adaptabilité structurelle des architectures dynamiques. Enfin, vu l'absence de travaux s'intéressant à la problématique de la reconfiguration temporisée, nous avons eu recours à une exploration d'autres domaines de recherche abordant la prise en compte des contraintes temporelles qui ont servi comme source d'inspiration de notre contribution.

Dans le deuxième chapitre, nous avons présenté un cadre formel relatif à la reconfiguration temporisée. Ce travail représente une extension du méta-modèle évoqué dans le premier chapitre, pour que ce méta-modèle puisse prendre en compte les contraintes temporelles.

Dans le troisième chapitre, nous avons présenté sommairement trois outils de transformation de graphes : AGG, PROGRES et GMTE. Ce dernier, développé au LAAS, est celui qui répond le mieux aux exigences d'implémentation de l'extension visée par notre projet. Il représente l'un des outils les plus performants, traitant de la transformation des graphes ainsi que d'un ensemble de variétés et de fonctionnalités supplémentaires. Ce chapitre comprend également une vue conceptuelle d'une extension que nous comptons apporter à cet outil afin de mettre en application le concept de reconfiguration temporisée. L'implémentation de cette extension fera l'objet de la deuxième phase de mon stage.

Comme nous l'avons cité dans l'introduction de ce mémoire, l'adaptation des architectures est classée en deux catégories, à savoir l'adaptation structurelle et l'adaptation comportementale. Dans nos travaux, nous nous sommes limités au premier type d'adaptation. En complément du travail réalisé, une perspective s'agissant de faire intégrer l'adaptation comportementale avec prise en compte des propriétés et des contraintes temporelles est bien envisageable.

Bibliographie & Webographie

[Alur et Dill, 1994] R. Alur, D. L. Dill. A theory of timed automata, Theoretical Computer Science 126, 1994.

[André, 2011] C. André. Modèles de temps et de contraintes temporelles de MARTE et leurs applications. Equipe-Projet Aoste, Rapport de recherche n° 7788 – Novembre 2011.

[Baresi et Heckel, 2002] L. Baresi, R. Heckel. Tutorial Introduction to Graph Transformation : A Software Engineering Perspective, Graph Transformation, Springer Berlin Heidelberg, 2002.

[Bouassida et al., 2008] I. Bouassida, K. Guennoun, K. Drira, C. Chassot, M. Jmaiel. Implementing a rule-driven approach for architectural self configuration in collaborative activities using a graph rewriting formalism, CSTST 2008.

[Chomsky, 1956] N. Chomsky. Three models for the description of language. IRE Transactions on Information Theory, 2 :113-124, 1956.

[Ehrig et al., 2006] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. Fundamentals of Algebraic Graph Transformation, Springer-Verlag Berlin Heidelberg, 2006

[Eichler et al., 2012] C. Eichler, I. Bouassida, T. Monteil, P. Stolf, K. Drira. Generic approach for graph-based description of dynamically reconfigurable architecture, 2012.

[GMTE R.M.] GMTE Rule Manual, http://homepages.laas.fr/khalil/GMTE/

[GMTE Ref 2.0] GMTE Reference, Release 2.0, http://homepages.laas.fr/khalil/GMTE/

[Guennoun, 2007] M. K. Guennoun. Architectures dynamiques dans le contexte des applications à base de composants et orientées service, Thèse de doctorat, Université TOULOUSE III – PAUL SABATIER, 2007

[Harel et al., 1990] E. Harel, O. Lichtenstein, A. Pnueli. Explicit Clock Temporal Logic, Fifth Annual IEEE Symposium, 1990.

[Rozenberg, 1997] G. Rozenberg. Handbook of Graph Grammars and Computing by Graph Transformation. World Scientic Publishing, 1997.

[Shürr et al., 1999] A. Schürr, A.J. Winter, A. Zündorf. The PROGRES approach : Language and environment. World Scientific, 1999.

[Taentzer, 2004] G. Taentzer. AGG : A graph transformation environment for modeling and validation of software. In Applications of Graph Transformations with Industrial Relevance : Second International Workshop, AGTIVE 2003, Springer-Verlag Berlin Heidelberg, 2004.

[Site 1] http://user.cs.tu-berlin.de/~gragra/agg/agg-docu.html

[Site 2] http://www.laas.fr/1-30644-Le-LAAS-aujourd-hui.php

[Site 3] http://sciences.blogs.liberation.fr/home/2009/11/le-cnrs-premi

%C3%A8re-institution-scientifique-mondiale-.html

[Site 4] http://www.cnrs.fr/fr/organisme/presentation.htm

[Site 5] http://www.laas.fr/1-27971-Equipes-de-Recherche.php

[Site 6] http://www.laas.fr/SARA/