Les interdépendances structurelles, un problème complexe traité par les graphes de coordination

Khalil Drira

▶ **To cite this version:**

Khalil Drira. Les interdépendances structurelles, un problème complexe traité par les graphes de coordination. 2009. hal-00448773

HAL Id: hal-00448773
https://hal.science/hal-00448773

Preprint submitted on 20 Jan 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Les interdépendances structurelles, un problème complexe traité par les graphes de coordination

Khalil Drira*,**

*CNRS ; LAAS ; 7 Avenue du Colonel Roche, F-31077 Toulouse, France
**Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
khalil@laas.fr
http://homepages.laas.fr/khalil/page

Résumé. Les travaux présentés traitent les problèmes relatifs à la coordination d'un ensemble de composants, laquelle inclut, en particulier, l'intégration et la distribution de ces composants sous différentes contraintes architecturales : interdépendance, dynamisme et distribution. La gestion de la dynamique de l'architecture et de ses interactions distribuées ainsi que son application pour le support des activités génériques de coopération ont été deux axes majeurs dans les problèmes traités. Cet article présente une approche basée sur les graphes pour le traitement de ces problèmes.

1 Introduction

Une activité coopérative est une activité en groupe, collaborative et coordonnée qui implique simultanément/ou à différents moments, différents “acteurs" artificiels ou humains, ayant des objectifs communs, qui produisent et échangent des données/des informations/ou des connaissances, au cours de sessions spontanées ou planifiées, et selon des règles de coordination implicites ou explicites pour la gestion des tâches et le partage de l'espace de travail.

Un environnement de support des activités en groupe (1) supporte la collaboration de ses utilisateurs, en mettant à leur disposition des outils spécialisés (dépendant du domaine d'activité) ou génériques (outils de discussion multimédia, outils d'édition partagée), et (2) intègre ces outils via des services de coordination qui introduisent la structuration, le contrôle, et gèrent la cohérence globale.

Les **interdépendances structurelles** constituent la pierre angulaire du problème de coordination qui garantit la cohérence. Nous les classons en trois catégories :

La première catégorie est celle des interdépendances relatives à l'**architecture des sites de coopération** (impliquant un ou plusieurs composants d'une ou plusieurs applications). Elles concernent la coordination des interdépendances entre composants induites par

- des contraintes du niveau communication, relatives notamment à la gestion des connexions (le serveur et le proxy doivent partager le même canal de communication et leur initialisation doit être coordonnée)
- ou des exigences du niveau coopération relatives aux règles de répartition des privilèges et d'attribution des droits (le composant central de gestion du droit de parole doit être activé d'abord sur la machine du modérateur de session).

La deuxième catégorie est celle des interdépendances relatives à la **structuration du groupe de coopération**. Elles concernent la coordination des interdépendances entre les acteurs et expriment généralement des liens de type producteur / consommateur à gérer de façon évolutive dans les structures dynamiques ou de façon figée dans les structures hiérarchiques.

La troisième catégorie est celle des interdépendances relatives à la **structure de l'espace de coopération**. Elles concernent la coordination des interdépendances entre les "objets de coopération" partagés (par ex. documents ou composantes de documents) et expriment par exemple des relations de "structuration linéaire" de type "prédécesseur (avant) / successeur (après)" ou des relations de "décomposition arborescente (ou hiérarchique)" de type "objet de coopération / composantes". On peut voir ainsi qu'un document édité en groupe (c'est ce que nous appelons objet de coopération dans ce cas) est structuré :

– de façon linéaire comme la succession de caractères, de mots, de lignes, ou de paragraphes (c'est que nous appelons composantes dans ce cas)
– ou de façon arborescente (ou hiérarchique) comme l'imbrication de parties, de chapitres, de sections (c'est que nous appelons composantes dans ce cas).

Pour traiter le problème de la **gestion des interdépendances structurelles**, nous avons conduit des études couvrant l'ensemble des catégories distinguées. Nous avons développé une technique générique de **"coordination guidée par la structure"** qui s'appuie sur des descriptions de *graphes* évolutifs étendus associés à des *règles de transformation* pouvant décrire les différentes évolutions d'une structure quelconque en général et d'une architecture logicielle plus particulièrement, leurs conditions et leurs conséquences, ceci de façon adaptée aux exigences du niveau coopération et aux contraintes et capacités du niveau communication Drira (2000). Les exigences du niveau coopération peuvent concerner, par exemple, la distribution du droit de modification des documents partagés dans un groupe d'utilisateurs avec différents critères de dynamicité et de granularité : par exemple, ne pas associer le droit obligatoirement à la totalité du contenu d'un document, et ne pas attribuer ce droit selon une répartition statique basée sur les identités ou les rôles, mais plutôt permettre de définir et d'attribuer dynamiquement le droit par partie de document et aux différents participants.

Le papier est organisé comme suit. La section 2.2 positionne nos travaux par rapport aux autres approches de coordination. La section 3 décrit la descrition des interdépendances structurelles par les graphes de coordination. La section 4 décrit, de façon informelle puis formelle, l'application de notre approche pour la gestion des architectures dynamiques appariées comme des objets de coordination complexes. La dernière section est consacrée au bilan et état de nos travaux.

2 Position du problème et positionnement

2.1 Le problème général de la coordination

De nombreux travaux sur la coordination dans les systèmes de coopération font référence aux définitions de Malone et Crowston qui considèrent que la coordination est l'acte de travailler ensemble harmonieusement. ("The act of working together harmoniously" Malone et Crowston (1990)) et qu'elle consiste à gérer les dépendances entre les activités" ("coordination is the managing of dependencies between activities" Malone et Crwoston (1994)). L'identifi-

cation des dépendances[1] (les domaines de coordination, les éléments de coordination et les relations) et la définition des procédures des gestion (mécanismes, protocoles, algorithmes) sont les deux axes autour desquels s'articulent les travaux de ce domaine. C'est aussi le cas pour leurs dérivations ou spécialisations en CSCW, en DAI, et en CSCL, ainsi que leurs applications à la gestion des espaces partagés, à la coordination pour le groupware et à la gestion du workflow.

2.2 La coordination dans les logiciels de support à la coopération

Les travaux liés aux supports logiciels pour les activités de coopération distribuée développent de nouvelles architectures et de nouveaux logiciels pouvant servir à une nouvelle organisation des procédures de travail en groupes géographiquement distribués qui coopèrent par l'échange d'informations qu'ils interprètent et transforment en connaissance pour l'achèvement d'un ou plusieurs objectifs. Ces objectifs peuvent être communs justifiant l'intérêt partagé pour leur achèvement, ou individuels, et il s'agit dans ce cas de "coopération" motivée par la complémentarité des moyens et des compétences. Les membres de ces groupes se réunissent périodiquement ou selon les besoins pour coordonner leurs contributions lors de sessions de travail selon un planning prédéfini ou improvisé.

Lorsque l'on s'intéresse aux travaux traitant des systèmes multi-utilisateurs, et plus particulièrement de leur application au support des activités de coopération (CSCW), l'on distingue de fortes contributions centrées sur la coordination qui visent l'une ou l'autres des deux fonctions que sont : "la gestion de la concurrence", et la gestion de la cohérence (ou consistence)". Les deux axes principaux concernés par cette recherche appliquée au CSCW sont, d'après la taxonomie de ce domaine Mills (2003), (1) la gestion du workflow, et (2) la gestion d'un espace d'information commun partagé (Figure 1).

Dans la catégorie du workflow, les utilisateurs agissent localement selon des procédures (ou tâches) préalablement planifiées, sur des données (objets, documents, bases de données, fichiers) privées ou communes.

Les travaux de recherche dans le cadre de cette catégorie relèvent du domaine de l'automatisation des processus d'entreprise selon des théories et des approches sociologiques ou organisationnelles. Il s'agit principalement de décomposer une activité en tâches et sous tâches, d'ordonnancer les tâches (principal thème de recherche de la communauté de recherche workflow) et de gérer leur affectation aux participants. Il s'agit essentiellement de la planification à laquelle s'intéresse aussi la recherche dans le domaine de l'intelligence artificielle distribuée pour la résolution coopérative de problèmes (Figure 1).

Les activités de la deuxième catégorie sont caractérisées par des sessions de coopération synchrones au cours desquelles les participants agissent simultanément et depuis des points d'accès distribués sur des objets partagés en suivant des règles de coordination pouvant être implicites ou explicites et en utilisant un ensemble d'outils qui leur permettent de progresser de façon coordonnée. Les objets peuvent être virtuellement ou réellement centralisés dans un espace de coopération partagé. Les objets peuvent représenter des fenêtres graphiques d'une application, des documents ou des parties dans ces documents, ou tout autre support en fonction de l'activité.

[1] appelées aussi interdépendances.

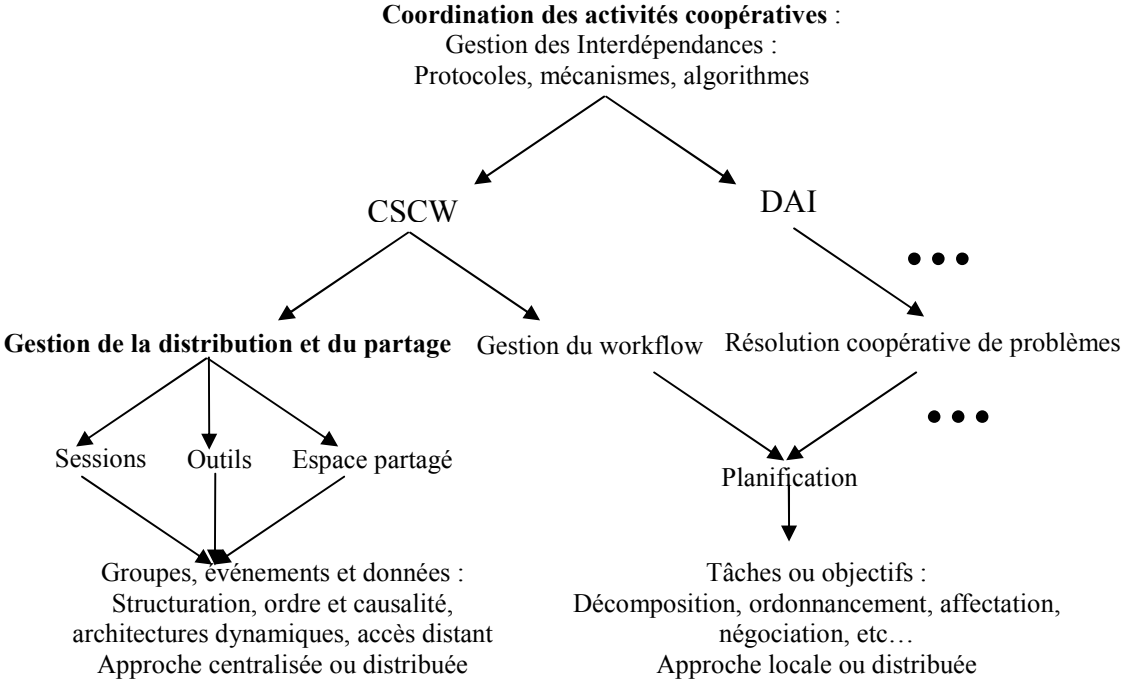

FIG. 1 – *Les différents domaines de recherche sur la coordination*

L'ensemble de nos contributions se situent dans cette deuxième catégorie. Elles portent sur la gestion de la cohérence au niveau de "l'espace d'information commun partagé" (vu comme une collection "d'objets de coopération") et au niveau de la structure logicielle qui soutient les utilisateurs dans le partage de cet espace et dans leur interaction.

Ces contributions concernent, dans une première partie, des solutions pour la gestion de la causalité entre les événements émis ou reçus par les participants et les outils de coopération qu'ils utilisent.

Dans une deuxième partie, nos contributions portent sur des solutions pour initialiser et adapter la structure afin de permettre à l'ensemble du système de fonctionner sous l'hypothèse de limitation ou d'absence de capacité de traitement de la coordination par les outils de coopération et par les composants qu'ils intègrent.

3 La description des interdépendances structurelles par les graphes de coordination

Notre approche se base sur les graphes de coordination définis pour représenter l'architecture des applications coopératives distribuées. L'étude de différentes familles d'applications coopératives a montré en effet qu'il était possible de les concevoir selon une architecture sy-

métrique d'un site à l'autre. Les particularités de chaque site sont représentées en paramètre de l'instance des composants qui constituent ce site. Le graphe de coordination est un graphe dont les nœuds sont associés à 3 types d'information : un type, une étiquette, et un ensemble de propriétés (appelées aussi paramètres) définies par le programmeur pour les besoins de son protocole. A ces informations, s'ajoutent : un identifiant géré par le système et une association $\langle$*type de nœud*, *classe de comportement*$\rangle$ choisie par le programmeur et utilisée par le système pour créer des instances de comportements.

L'évolution du graphe de coordination se fait selon un protocole de coordination qui utilise un ensemble de règles de transformation. De façon similaire aux règles de production des grammaires de graphes, l'application d'une règle de transformation exige la présence d'un certain motif dans le graphe de coordination. Le motif recherché fait partie d'un schéma de transformation plus général appelé règle de coordination et qui se décline en trois parties :

- le motif "Recherché" : R,
- la partie du motif à effacer ("Détruire") sur le graphe de coordination initial : $D \subseteq R$
- une partie supplémentaire à insérer ("Ajouter") au graphe nouvellement obtenu : A.

L'ensemble de ces trois parties, $\langle (R \setminus D) \cup D \cup A \rangle$, doit former un graphe, appelé graphe de la règle. Le graphe de la règle déclare des nœuds et des arcs pouvant être identiques à ceux du graphe de coordination. La recherche d'un motif dans le graphe initial applique alors des règles d'unification considérant les types des nœuds : deux nœuds sont unifiables s'ils ont le même type. Lorsque l'unification des types est insuffisante pour le problème en cours de description, le modèle offre la possibilité d'unifications des labels qui étiquettent les nœuds : deux nœuds sont unifiables s'ils sont unifiable par le type et s'ils ont les mêmes étiquettes. Et pour augmenter la puissance d'expression des règles de coordination, il est possible d'utiliser des nœuds dont le type et/ou l'étiquette sont génériques et unifiables avec des types différents ou des étiquettes différentes. Lorsque ces 2 contraintes d'unification sont insuffisantes, nous offrons la possibilité de programmer des fonctions booléennes sur des paramètres attachés aux nœuds du graphe. Le concepteur est libre de choisir les paramètres et les fonctions qui conviennent à son application. Dans le cadre de la gestion du partage de document, ces paramètres ont été représentés par une structure contenant un couple d'entiers et un attribut booléen indiquant les coordonnées de la zone de texte dans le document et son état (voir fig. 3).

Des formats visuels ou textuels peuvent être utilisés pour décrire une règle. Nous présentons, dans la figure 3, un exemple de représentation de chaque catégorie.

Nous présentons, dans la figure 2, les conventions de la notation que nous utilisons pour décrire visuellement de façon simplifiée une règle de transformation de graphe. La description complète tient compte des types des nœuds de leurs paramètres ainsi que des classes de comportement Java (ou corba dans d'autres exemples) qui leurs sont associées.

La notation visuelle (figure 2) décompose le graphe de la règle en trois parties correspondant respectivement :

- au motif recherché R représenté par tout le fragment se trouvant :
  - à gauche du symbole "parenthèse ouvrante", "**(**"
- à la partie de ce motif à effacer D représenté par tout le fragment se trouvant :
  - à gauche du symbole "parenthèse fermante", "**)**",
- et le motif à insérer A représenté par tout le fragment se trouvant :
  - à droite du symbole "parenthèse ouvrante", "**(**".

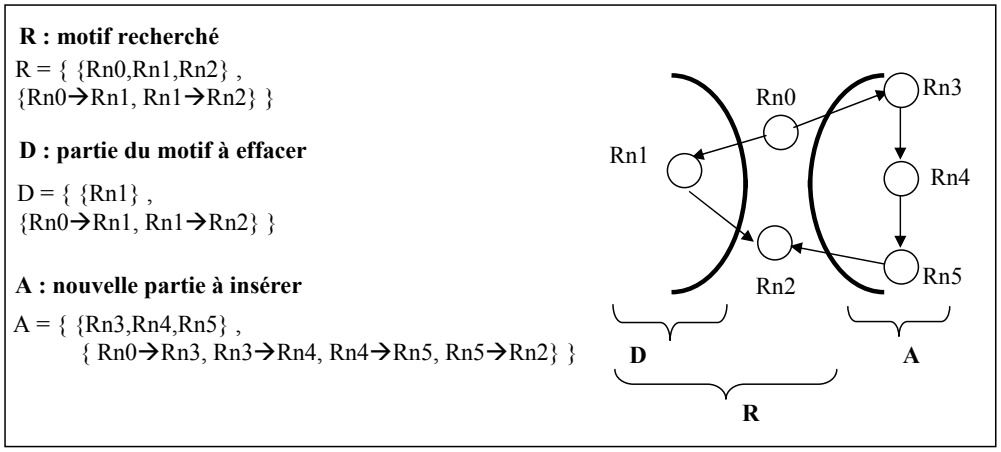

FIG. 2 – *Notation visuelle pour décrire une règle simplifiée de transformation de graphe*

D'après les lois de transformation, une règle peut être applicable plusieurs fois à plusieurs parties du graphe de coordination. En effet celui-ci peut contenir à plusieurs endroits le même motif exigé pour l'application de la règle. Lorsque l'on désire qu'une règle applicable soit appliquée à toutes ces parties, on a la possibilité, par l'ajout d'un attribut à la description de cette règle, d'exiger son application partout où il est possible de le faire dans le graphe de coordination. Ce type de règle est utilisé dans le modèle de coordination linéaire (défini pour la coordination de l'édition coopérative) pour autoriser l'évolution de l'espace non seulement en terme de nombre de zones mais aussi en terme de contenu de chaque zone. En effet, les zones étant dynamiquement définies par leurs positions les unes par rapport aux autres, la mise à jour du contenu d'une zone entraîne la modification des coordonnées des nœuds qui lui succèdent dans le graphe de coordination.

4 Application à la gestion de l'architecture dynamique des sites de coopération

Dans cette section, nous décrivons le principe de la gestion de l'architecture des sites de coopération et indirectement des applications qu'ils hébergent. Nous commençons par caractériser les applications distribuées coopératives et nous détaillons ensuite l'utilisation des graphes de coordination pour la description de l'architecture d'un site de coopération et la gestion de sa dynamique.

4.1 Le contexte général de la description des architectures logicielles

Les deux buts principaux de la description des architectures sont d'après IEEE-S2ESC Ellis et al. (1996) :

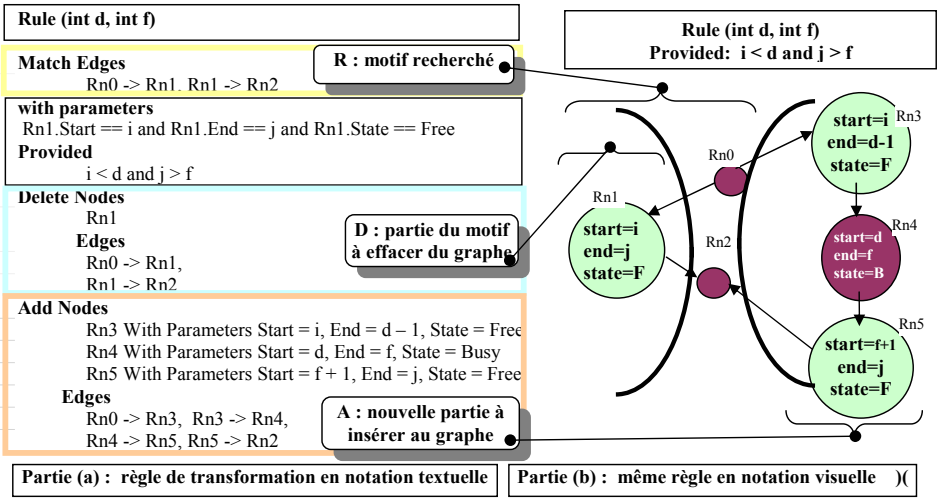

FIG. 3 – *Exemple de règle simplifiée au format textuel et visuel*

– "La description pour la conception" ("architecture as design") utilisée comme un moyen d'exprimer des caractéristiques architecturales de haut niveau du système et permettant de définir et organiser ses éléments et leurs interactions.
– "La description de type style" ("architecture as type") permettant de raisonner sur l'architecture, par exemple en terme de compatibilité, d'interopérabilité, et d'interchangeabilité de composants. Un style d'architecture est alors défini comme un ensemble de modèles ou de règles pour créer une ou plusieurs architectures de manière consistante.

Nos contributions ont couvert chacun des deux objectifs identifiés ci-dessus. Elles ont principalement concerné "la description pour la conception" que nous allons décrire dans la suite. D'autres travaux ont porté sur la "description de type style".

4.2 Le contexte des applications distribuées coopératives

Une *application distribuée coopérative* est définie comme l'intégration :
– de '*n*' *composants coopératifs* (élémentaires ou composites) symétriquement distribués sur 'n' *sites de coopération*,
– et d'un ensemble de services de communication et de coordination pouvant être :
  – regroupés sur un site (ce site peut faire partie de l'ensemble des sites de coopération ou non)
  – ou distribués (symétriquement ou non) sur l'ensemble des sites de coopération.

Considérant la symétrie de la distribution, nous associons le graphe de coordination et ses règles de transformation à la gestion de la dynamique de l'architecture logicielle interne aux sites de coopération. Un site est identifié en pratique à la machine sur laquelle travaille le participant, mais aucune corrélation n'existe entre site et machine en général : un site peut

correspondre à plusieurs machines, et une machine peut exécuter les composants associés à plusieurs sites. Des problèmes de portabilité peuvent ainsi être résolus quand c'est nécessaire.

Lorsqu'une règle de transformation est applicable et est appliquée, un ensemble d'actions est exécuté pour modifier l'architecture logicielle de chaque site. Ces actions ont lieu lors de l'application de la règle et selon la loi suivante :

- l'ajout d'un nouveau nœud au graphe de coordination entraîne la création, sur tous les sites (machine ou groupe de machines) connectés, d'une instance du composant (objet Corba ou Java dans notre implantation) qui lui a été associé lors de la déclaration de son type.
- L'effacement d'un nœud du graphe de coordination entraîne la destruction, sur tous les sites connectés, du composant qui lui a été associé. Un modèle de gestion plus affiné a été aussi défini. Il introduit des actions supplémentaires permettant l'activation et la désactivation sans détruire et recréer les composants. Aussi les conditions d'application des règles tiennent compte de l'état des composants (actif/inactif) et permettent de le modifier.
- Les composants créés sont paramétrés par une copie des paramètres (dits utilisateurs) qui ont été associés aux nœuds de coordination lors de leur déclaration. Ceci permet de concevoir une architecture symétrique tout en permettant des comportements différents de l'instance du même composant d'un site à l'autre.
- Certaines règles permettent de modifier le comportement d'un composant sur un ou plusieurs sites sans modifier le graphe de coordination. Ces règles particulières agissent sur les paramètres des composants sans effacer les nœuds correspondants. Ce type de règle est utilisé par exemple pour gérer le droit de parole dans un groupe de coopération en associant un privilège aux composants qui soutiennent les actions des participants. La circulation de ces privilèges se fait par l'application de ces règles.

4.2.1 Formalisation

Dans cette section, nous utilisons les notations qui se basent sur la structure abstraite de graphe appelée ACG (Graphe Abstrait de Composants) définie initialement dans Guennoun et al. (2003) . De cette structure sont dérivées deux autres structures de graphe par instatiation partielle ou totale des attributs des nœuds : le "graphe d'architecture" et le "graphe de règles". Ces deux structures permettent de formaliser respectivement le "graphe de coordination" et une "règle de transformation".

La structure ACG

Le graphe abstrait de composants (ACG) est une structure marquée et générique. Elle permet de définir les graphes d'architecture comme des ACG totalement instantiés, et les graphes de règles comme des ACG partiellement instantiés. La première structure décrit une architecture comme un ensemble de composants associés aux nœuds du graphe et un ensemble d'arcs dénotant les relations d'interdépendance entre ces composants.

$$\textbf{\textit{ACG}} : \mathcal{P}(Nodes) \times \mathcal{P}(Edges)$$

Dans une structure ACG, les nœuds décrivent des composants logiciels et sont marqués par les champs suivants : la classe, le comportement, l'état du composant (actif/inactif), les

facettes de son comportement, sa localisation (le site sur lequel il est exécuté), et une liste supplémentaire de paramètres concernant le niveau applicatif. Nous distinguons deux catégories de nœuds : les nœuds de règles, et les nœuds de graphes. La différence entre ces deux types de nœuds est que le premier est une abstraction d'un type de composants et peut avoir des champs variables [2], alors que le second correspond à un composant instantié de l'architecture et ne peut donc avoir que des champs totalement instantiés.

$$\textbf{\textit{Node}} : Class \times State \times Facets \times Location \times Parameters$$

Les arcs sont orientés et sont définis par le couple de nœuds qu'ils relient. Ils constituent le deuxième moyen élémentaire de la description d'architecture. Ils peuvent modéliser un large panel de relations comme les dépendances de niveau communication, ou des dépendances de niveau applicatif entre les composants.

$$\textbf{\textit{Edge}} : Node \times Node$$

Structure des règles de transformation
Nous définissons une *règle de transformation* par une partition d'un *graphe de règle* permettant de décrire les contraintes qui conditionnent l'évolution de l'architecture et les changements qui se produisent quand une règle est applicable. Au niveau supérieur de l'abstraction, une règle de transformation peut être vue comme un triplet, $RT \equiv$ <Partition, constraints, Substitutions>, où :

- **Partition** : est une décomposition du graphe de la règle de transformation en quatre zones :
  - La zone *R :* Un fragment du graphe de la règle qui devrait être identifié (par homomorphisme) dans le graphe d'architecture. Ce fragment du graphe restera inchangé après l'application de la règle.
  - La zone *D :* Un fragment du graphe de la règle qui doit être identifié (par homomorphisme) dans le graphe de l'architecture. Le fragment du graphe qui lui a été associé par l'homomorphisme est supprimé après l'application de la règle.
  - La zone[3] *Abs :* Un fragment du graphe de la règle qui ne doit pas être identifié (par homomorphisme) dans le graphe de l'architecture pour que la règle de transformation soit applicable.
  - La zone *A :* Le fragment du graphe de la règle qui sera ajouté après l'application de la règle.
- **Constraints** : Décrit des contraintes sur les champs des nœuds. La règle n'est applicable que si toutes ses contraintes sont satisfaites par les nœuds du graphe de l'architecture qui sont unifiés avec les nœuds de la règle. Une contrainte est un couple dont le premier champ est la fonction d'évaluation de la contrainte δ (une fonction prenant en paramètre un ensemble de nœuds et renvoyant un booléen), et le deuxième est l'ensemble des nœuds qui sera évalué par δ.

$$\textbf{Const} : \mathcal{P}(\delta(\ :\mathcal{P}(\text{Nodes}) \rightarrow \text{boolean}) \times \mathcal{P}(\text{Nodes})))$$

[2] Les variables seront préfixées par le symbole "_". Par exemple, la notation <E,_x,F,Ad1> dénote un nœud de la classe E avec une facette F, situé dans le site Ad1. La variable _x indique que le nœud peut être dans un état actif ou inactif.

[3] Cette zone correspond à la "restriction", une contrainte non décrite dans les section précédentes. Elle permet d'alléger certains modèles, mais n'est pas indispensable pour la modélisation.

– **Substitutions** : Modélise les différentes substitutions que devraient subir les champs de certains nœuds du graphe après l'application de la règle. Les substitutions sont modélisées par la procédure de substitution σ qui prend en paramètres un ensemble de nœuds et permet de substituer à certains de leurs champs des nouvelles valeurs. Ceci permet de spécifier l'évolution dynamique à l'échelle du composant (migration, changement de comportement, . . .etc).

$$\textbf{\textit{Sub}} : \mathcal{P}\ (\ \sigma\ (\ : \mathcal{P}\ (Nodes) \rightarrow void) \times \mathcal{P}(Nodes))$$

Ainsi, avec les définitions précédentes, une règle possède la structure suivante :

$$\textbf{\textit{Rule}} : \underbrace{R \times D \times A \times Abs}_{Partition} \times Const \times Sub$$

Le protocole de coordination

Le protocole de coordination est en charge de gérer et de décrire l'évolution dynamique de l'architecture du système. Ce protocole manipule, le graphe courant, les règles de coordination et les événements à traiter, et associe à chaque type d'événements les règles de transformation correspondantes. Il associe, aussi, pour chaque type d'événements et pour chacune des règles lui correspondant, la procédure de transformation qui doit être appliquée à chaque règle (au niveau de son graphe, son champ *constraints*, et son champ *substitutions*) avant l'unification avec le graphe. Le protocole de coordination peut introduire aussi des événements spéciaux traduisant, par exemple, des vérifications de propriétés telles que des propriétés de *sûreté* ou de *complétude*. Ces propriétés seront décrites sous la forme d'une ou de plusieurs règles de coordination.

$$\textbf{\textit{Protocol}} : Graph \times \mathcal{P}(Rules) \times \mathcal{P}(EventType \times Trans \times \mathcal{P}(Rules))$$

L'événement décrit l'action de déclenchement menant à l'application d'un ensemble de règles de transformation. Il peut être produit par le système, ou par son environnement et est décrit comme un couple contenant le type de l'événement, et les paramètres qu'il transporte.

$$\textbf{\textit{Event}} : EventType \times EventParameters$$

Le champ *Trans* permet de répercuter les paramètres des événements sur les règles de transformation. Les règles sont ainsi instantiées en affectant des valeurs à certaines de leurs variables.

$$\textbf{\textit{Trans}} : \mathcal{P}(\alpha\ (\ : \mathcal{P}(Nodes) \times Event \rightarrow void) \times \mathcal{P}(Nodes))$$

Application des règles de transformation

Nous définissons la fonction d'unification comme une fonction récursive qui établit un homomorphisme entre la partition du graphe de la règle de transformation et le graphe de l'architecture (en tenant compte du champ *constraints* de la règle). Elle est basée sur les quatre définitions suivantes qui décrivent l'unification d'un ensemble de nœuds du graphe de règle avec un ensemble de nœuds du graphe de l'architecture.

Définition 1 (Unification de champs de nœuds)

$$\textbf{\textit{Unifiable}}(champ_i, champ_j) \equiv \begin{cases} \exists_X \in Variables, \textbf{\textit{Tel que}}\ champ_i = _X, \ \textbf{\textit{Ou}} \\ champ_i = champ_j \end{cases}$$

Définition 2 (unification de deux nœuds) *Soit,*
$N_1 = (N_1.champ_1, \ldots, N_1.champ_n)$ *un nœud d'un graphe de règle (r), et*
$N_2 = (N_2.champ_1, \ldots, N_2.champ_m)$ *un nœud d'un graphe d'architecture. Alors,*

$$\textbf{\textit{Unifiable}}(N_1, N_2) \equiv \begin{cases} (n = m),\ \textbf{\textit{Et}} \\ \forall i \in [1, .., n], \textbf{\textit{Unifiable}}(N_1.champ_i, N_2.champ_i),\ \textbf{\textit{Et}} \\ \textbf{\textit{Unifiable}}(N_1.successeurs, N_2.successeurs),\ \textbf{\textit{Et}} \\ Const(r),\ \textbf{\textit{Et}} \\ \textbf{\textit{Pas d'inconsistence dans les unifications}} \end{cases}$$

Définition 3 (Unification de deux ensembles ordonnés de nœuds) *Soit EO_1 un ensemble ordonné de nœuds d'un graphe de règle, (r), tel que $EO_1 = [N_{1,1}, \ldots, N_{1,n}]$. Soit EO_2 un ensemble ordonné de nœuds d'un graphe d'architecture tel que $EO_2 = [N_{2,1}, \ldots, N_{2,m}]$. Alors,*

$$\textbf{\textit{S_Unifiable}}(EO_1, EO_2) \equiv \begin{cases} (n = m),\ \textbf{\textit{Et}} \\ \forall i \in [1, .., n], \textbf{\textit{Unifiable}}(N_{1,i}, N_{2,i})),\ \textbf{\textit{Et}} \\ Const(r),\ \textbf{\textit{Et}} \\ \textbf{\textit{Pas d'inconsistence dans les unifications}} \end{cases}$$

Définition 4 (Unification de deux ensembles de nœuds) *Soit NR un ensemble de nœuds d'un graphe de règle tel que $NR = \{N_{1,1}, \ldots, N_{1,n}\}$ et NG un ensemble de nœuds de graphe d'architecture tel que $NG = \{N_{2,1}, \ldots, N_{2,m}\}$. Alors,*

$$\textbf{\textit{Unifiable}}(NR, NG) \equiv \begin{cases} n < m,\ \textbf{\textit{Et}} \\ \exists \{N_{2,i1}, \ldots, N_{2,in}\} \subset \{N_{2,1}, \ldots, N_{2,m}\},\ \textbf{\textit{Tel que}} \\ \textbf{\textit{S_Unifiable}}([N_{1,1}, \ldots, N_{1,n}], [N_{2,i1}, \ldots, N_{2,in}]) \end{cases}$$

Définition 5 (Unification d'une règle avec un graphe) *Soit r une règle et g un graphe, soit précondition(r) l'ensemble des nœuds et des arcs appartenant à ($R(r) \cup D(r)$) et soit restriction(r) l'ensemble des nœuds et des arcs appartenant à ($R(r) \cup D(r) \cup Abs(r)$), alors,*

$$\textbf{\textit{Unifiable}}(r, g) \equiv \begin{cases} \exists s_g = (\{n_0, \ldots, n_i\}, \{e_0, \ldots, e_j\}) \in g,\ \textbf{\textit{Tel que}} \\ \textbf{\textit{S_Unifiable}}(precondition(r), s_g),\ \textbf{\textit{Et}} \\ Const(r),\ \textbf{\textit{Et}} \\ \{(\forall s'_g = (\{n_0, \ldots, n_i, \ldots, n_{i+k}\}, \{e_0, \ldots, e_j, \ldots, e_{j+l}\}) \subset g) \\ \quad \Longrightarrow \neg\ (\textbf{\textit{S_Unifiable}}(restriction(r), s'_g))\} \end{cases}$$

L'unification d'une règle r avec un graphe g suivra la démarche suivante :

- Si r n'est pas unifiable avec g alors g reste inchangé.
- Sinon : - On rajoute dans le graphe g des copies des nœuds et des arcs contenus dans le champs *A(r)*. On détruit les nœuds et les arcs du graphe qui ont été unifiés avec des nœuds et des arcs du champs *D(r)*. On modifie les champs des nœuds de g qui ont été unifiés avec les nœuds figurant dans le champs *Sub(r)*.

5 Conclusion

Nos travaux ont abouti à la conception d'un modèle formel pour la coordination, constituant une approche originale de synthèse de services de coordination. Nous avons validé et mis

en œuvre cette approche selon une architecture multi-niveaux qui étend le modèle composants et services distribués. L'architecture distingue 3 niveaux fonctionnels : (1) La communication, (2) la coordination, et (3) la coopération. Le niveau coordination implante les fonctions relatives à la coordination des sites (instantiation, activation/désactivation des objets internes aux sites) et des activités (gestion de l'espace de travail partagé) pour un accès cohérent aux objets partagés.

Un outil efficace a été développé en C++ pour la comparaison et la transformation de graphes `http://homepages.laas.fr/khalil/GTE`, et utilisé pour valider nos modèles. Son utilisation est actuellement facilité par son implantation en librairie C++ utilisable de façon interactive via une interface textuelle ou graphique, ou bien depuis un programme extérieur (Java ou C++). Une analyse de la complexité de l'algorithme et des mesures expérimentales ont démontré l'efficacité de notre implémentation.

Références

Drira, K. (2000). A coordination middleware for collaborative component-oriented distributed applications. *Special Issue on information and communication middleware. NETNOMICS (Economic Research and Electronic Networking) 2*(2000), 85–99. (Baltzer Science Publishers journal).

Ellis, W., R. Hilliard, P. Poon, D. Rayford, T. Saunders, B. Sherlund, et R. Wade (1996). Toward a recommended practice for architectural description. In *Proceedings of 2nd IEEE International Conference on Engineering of Complex Computer Systems, Montreal, Quebec, Canada, October 21-25, 1996.*

Guennoun, K., K. Drira, et M. Diaz (2003). A proved component-oriented approach for managing dynamic software architectures. In *7th IASTED International Conference on Software Engineering and Applications (SEA 2003)*, Marina del Rey, CA, USA. ACTA PRESS. ISBN 0889863946.

Malone, T. et K. Crowston (1990). What is coordination theory and how can it help design cooperative systems ? In *Proceedings of CSCW '90*, New York. ACM.

Malone, T. et K. Crwoston (1994). The interdisciplinary Study of Coordination. *ACM computing Surveys 26*(1), 87–119.

Mills, K. L. (2003). Computer-Supported Cooperative Work. In *Encyclopedia of Library and Information Sciences (2nd Edition), Marcel Dekker (eds), New York DOI : 10.1081/E-ELIS 120008706*.

Summary

The work presented addresses the problems of coordinating a set of components, which includes, in particular, the integration and distribution of these components under various architectural constraints: interdependence, dynamism and distribution. The management of the dynamics of architecture and its distributed interactions as well as its application for the support of the generic cooperative activities were two major axes within the addressed problems. This paper presents an approach based on graphs for handling these problems.