Contribution à la conception des architectures logicielles et des protocoles de coordination pour les systèmes distribués coopératifs

Khalil Drira

Habilitation à Diriger les Recherches

présentée au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

en vue de l'obtention du
Diplôme de l'Université Paul Sabatier (Toulouse III)

Contribution à la conception des architectures logicielles et des protocoles de coordination pour les systèmes distribués coopératifs

par
Khalil Drira
Chargé de Recherche CNRS

soutenue, le 12 janvier 2005, devant le Jury :
M. Guy Juanole, Président
Me. Françoise André, rapporteur
M. Jean-Marc Geib, rapporteur
M. Hervé Guyennet, rapporteur
M. Michel Diaz, examinateur

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS). Je remercie les Directeurs successifs du LAAS, Monsieur Alain Costes, Monsieur Jean-Claude Laprie et Monsieur Malik Ghallab, pour m'avoir accueilli au sein du LAAS.

Je suis très reconnaissant à Madame Françoise André, Professeur à l'Université de Rennes I, et à Messieurs : Jean-Marc Geib, Professeur à l'Université des Sciences et Technologies de Lille, Hervé Guyennet, Professeur à l'Université de Franche-Comté, Guy Juanole, Professeur à l'Université Paul Sabatier, et Michel Diaz, Directeur de Recherche CNRS pour l'honneur qu'ils m'ont fait en participant à ce jury d'habilitation. Je remercie plus particulièrement Madame Françoise André, Monsieur Jean-Marc Geib et Monsieur Hervé Guyennet qui ont accepté la charge d'être Rapporteur de ces travaux. Je remercie très particulièrement Monsieur Michel Diaz pour avoir accepté la charge d'être le Directeur de ces travaux.

Les résultats présentés dans ce mémoire sont le fruit de la participation importante de Saùl Pomarès, de Martin Molina, d'Olga Nabuco, de Karim Guennoun et de Francisco Moo Ména dans le cadre de leurs thèses de Doctorat. Je leurs dois beaucoup et je les remercie de leurs précieuses contributions. Je dois également remercier pour leurs contributions non moins importantes : Béatrice Cambou-Bastier, Mehdi El Jed, Olfa Jenhani, Etienne Roblet et Frédéric Gouëzec.

Je remercie également tous mes collègues et doctorants du groupe OLC qui ont participé activement à l'élaboration de ces travaux. Je cite notamment Pierre Azéma, Thierry Villemur, Laura Rodriguez, Jean Fanchon, François Vernadat, Pierre Gradit, Véronique Baudin, et Said Tazi.

Mes remerciements vont également à tous mes collègues du groupe OLC avec qui je partage, depuis mon entrée au LAAS, les ambitions et les charges du métier de la recherche scientifique.

Je ne saurai oublier de remercier mes collègues, amis, et collaborateurs mexicains, bésiliens et tunisiens : Raùl Jacinto, Félix Ramos, Ernesto Lopez, João-Mauricio Rosario, Reinaldo Da Silva, Mohamed Jmaiel, Ahmed Hadj Kacem, Abedelmajid Ben Hamadou, Riadh Ben Halima, Mohamed Hadj-Kacem, Sofiene Khemakhem et Imen Loulou. Même si ce mémoire ne le détaille pas, ils sont aussi associés à mon bilan de recherche ou à ses perspectives.

Je remercie beaucoup tous les membres des différents services du LAAS qui, par leur disponibilité et leur compétence, nous offrent les meilleurs conditions de travail et permettent l'accueil de nos visiteurs dans d'aussi bonnes conditions.

Résumé

Ce mémoire résume les contributions de nos travaux de recherche, conduits après la thèse de doctorat soutenue en octobre 1992. Une première partie des travaux s'inscrit dans la continuité du sujet de recherche de la période doctorale dans le domaine des systèmes de communication. Une deuxième partie, plus importante, s'inscrit dans une évolution vers une problématique plus récente qui cible le domaine des systèmes distribués coopératifs. Ce domaine a émergé à la fin des années 80 avec l'évolution des réseaux et des technologies logicielles et fait, depuis, l'objet de nombreux projets dans le monde.

Ainsi, nos thèmes de recherche s'articulent autour des architectures et des protocoles de communication et de coordination pour les logiciels distribués coopératifs. Dans le cas de la communication, nous avons étudié un contexte d'interaction synchrone point-à-point, et dans le cas de la coordination, différents types et modes d'interaction sont considérés : synchrone/asynchrone, point-à-point ou multi-point, par échange de messages ou par invocation de méthodes à distance.

Nos contributions ont concerné la spécification, la modélisation, la conception, la validation, l'implémentation et l'expérimentation. L'adoption d'une démarche orientée modèle formel constitue la caractéristique partagée par les différents résultats de cette recherche. Cette démarche, orientée modèle, appliquée aux problèmes de conception et de validation des systèmes à architecture multi-composants peut être considérée comme le fil conducteur principal dont dérive l'ensemble de nos travaux.

L'étude du comportement des composants d'une architecture, qui peut se définir comme l'ensemble des règles qui décrivent la dynamique de leurs interactions, a constitué nos travaux de recherche initiaux.

Les travaux ultérieurs se sont ouverts pour couvrir les problèmes relatifs à la coordination d'ensembles de composants, laquelle inclut, en particulier, l'intégration et la distribution de ces composants sous différentes contraintes architecturales : interdépendance, dynamisme et distribution.

Enfin, 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.

L'édition en groupe a constitué pour nous une catégorie spécifique d'activité coopérative. La télé-ingénierie et la télé-formation ont constitué nos domaines d'expérimentation principaux.

1

Introduction

"*Le passage du paradigme de l'algorithmique à celui de l'interaction est le miroir du passage de la technologie des unités centrales à celles des stations de travail et des réseaux, du simple calcul aux systèmes embarqués et aux interfaces graphiques, et de l'orienté procédure à la programmation distribuée orientée objet.*"

Peter Wegner [Wegner, 1997]

1.1 Historique et contexte des travaux

Les activités de recherche présentées dans ce manuscrit s'inscrivent dans le contexte de la modélisation, la vérification, la conception, l'implantation et le test qui sont autant des étapes du cycle de vie d'un système que des centres d'intérêts de recherche. Plus précisément, les activités s'articulent autour de la problématique de l'étude des architectures et des protocoles qui régissent les interactions dans un système distribué multi-composants. Les composants correspondent à des processus concurrents ou à des objets logiciels répartis, les interactions à un niveau de communication, de coordination ou de coopération. Ces activités ont commencé dans le cadre conceptuel de la modélisation et de la vérification lors de notre travail de thèse et dans les deux années qui ont suivi. Elles ont particulièrement contribué, par la technique d'analyse de testabilité, aux méthodes de validation (vérification et test) des systèmes communicants sous contraintes particulières d'observabilité et de commandabilité. Nous nous sommes ensuite intéressés aux descriptions probabilisées des systèmes communicants pour l'évaluation des performances et la validation quantifiée des propriétés logiques.

Nous avons alors orienté nos activités de recherche vers le domaine des systèmes distribués coopératifs. Nous avons, surtout, étendu nos activités de recherche pour couvrir les phases de conception et d'implantation dans le cycle de vie des logiciels.

Après une synthèse de nos contributions durant la période post-doctorale dans le domaine des systèmes communicants, cette introduction et le reste de ce mémoire se consacrent à la présentation de nos activités de recherche plus récentes. Ces activités ont fait l'objets d'encadrement de PFE, de DEA et de thèses principalement en France, mais aussi à l'étranger dans le cadre de la coopération avec l'Université du Sud en Tunisie (ENIS et FSEGS) et l'Université de Campinas au Brésil (UNICAMP et CENPRA).

Ces activités portent sur des études dans le domaine des systèmes distribués coopératifs. Elles

comportent des contributions générales relatives à la conception, l'implantation et la gestion de l'évolution dans les architectures logicielles et les applications distribuées coopératives. Nos activités comportent également des contributions centrées sur le thème de la coordination dans les systèmes distribués coopératifs.

1.1.1 Travaux relatifs aux systèmes communicants

Nos travaux ont ciblé la mise en œuvre de solutions logicielles basées sur les spécifications formelles pour la vérification, l'analyse de la testabilité et la génération de séquences de test des systèmes communicants. Nos travaux se sont déroulés dans un contexte de description des propriétés logiques liées à l'ordonnancement et à la synchronisation des actions de communication. Nos travaux relatifs à la vérification se sont ensuite étendus pour tenir compte des propriétés quantitatives : temporelles et probabilistes [40Ci].

Pour la vérification, nous avons défini, pour l'équivalence de test, des procédures de décision [43Ci] basées sur la notion de bissimulation permettant de réduire un système de transitions en utilisant les algorithmes de type k-équivalence ou MRCP. L'équivalence de test a un pouvoir d'abstraction intermédiaire entre l'équivalence de trace et l'équivalence observationnelle. La première réduit fortement les systèmes de transitions mais occulte les problèmes relatifs au blocage. La deuxième peut voir son pouvoir de réduction se perturber par les différents types de non déterminisme : avant, pendant ou après une action. Ces différents types de non-déterminisme sont souvent présents de façon non significative pour le comportement du système en cours de spécification. Ils découlent souvent plus d'un style involontaire de spécification que d'une volonté de la personne à l'origine de cette spécification. L'équivalence de test permet de confondre ces différents cas de non déterminisme sans ignorer les risques de blocage. Cette caractéristique a été expérimentée pour la vérification du modèle du protocole OSI-TP [44Ci].

L'analyse de la testabilité est le résultat principal de notre recherche dans le domaine des systèmes communicants. L'originalité de notre approche réside dans l'association de son intérêt et de son utilisabilité en pratique avec son fondement théorique rigoureux. Elle est articulée autour des techniques de description formelles dérivées des algèbres de processus (LOTOS notamment) et des théories de test de Brinksma et Leduc. Elle traite le cas des spécifications multi-composants et considère le cas difficile du test d'un des composants à travers son environnement constitué par l'ensemble des autres composants. Dans un premier temps, nous nous sommes placés sous l'hypothèse de la théorie de Brinksma qui ignore la divergence [45Ci]. Ces résultats de recherche de la période doctorale ont été ensuite améliorés, durant la période post-doctorale, pour augmenter la fiabilité des résultats de l'analyse par le traitement explicite de la divergence [8J]. Nous avons enfin révisé notre approche de traitement de la divergence en la positionnant dans le contexte des nouvelles équivalences et théories de test publiées ultérieurement par Leduc [7J].

La génération de séquences de test permet de compléter l'exploitation de la spécification formelle. Nous avons développé durant nos travaux de thèse, une approche permettant de générer des séquences de test [48Ci], [47Ci]. Nous avons aussi défini une approche pour sélectionner, parmi plusieurs séquences, des représentants au même pouvoir de discrimination [49Ci]. Nous avons, ensuite, durant la période post-doctorale, amélioré la procédure de génération par la prise en compte des symétries dans les spécifications [38Ci].

1.1.2 Travaux relatifs aux systèmes distribués coopératifs

Les systèmes distribués coopératifs sont des systèmes distribués multi-utilisateurs dans lesquels il y a un fort degré d'interaction. Les interactions sont liées d'une part à la distribution des différentes composantes du système ainsi que la distribution des utilisateurs. D'un autre côté, elles sont liées à la nature coopérative de ces systèmes; celle-ci suppose une forte interactivité pour la consultation et la production de données ou d'informations d'une part, et pour l'échange et le partage de ces données par les différents utilisateurs, d'autre part.

L'étude des systèmes distribués coopératifs soulève, à la fois, des problèmes relatifs aux contraintes et exigences d'ordre comportemental (que l'on pourrait aussi appeler caractéristiques fonctionnelles algorithmiques ou événementielles), et des problèmes relatifs aux contraintes et exigences d'ordre structurel (que l'on pourrait aussi appeler caractéristiques architecturales ou topologiques). Dans le premier cas concernant les aspects comportementaux, il s'agit par exemple de problèmes algorithmiques liés à la communication de groupe causalement ou totalement ordonnée. Il peut s'agir aussi de problèmes fonctionnels relatifs à la nature de l'activité supportée par les dits systèmes (gestion de session, gestion du partage, gestion du workflow). Dans le second cas concernant les aspects structurels, il s'agit par exemple de problèmes liés à la gestion d'une architecture dynamique pour optimiser la distribution des composants ou pour s'adapter à la mobilité des utilisateurs ou aux changements dans les groupes ou l'espace de coopération. On peut distinguer notamment la gestion du cycle de vie des composants, leur distribution et déploiement ainsi que la gestion des canaux de diffusion qu'ils utilisent. Il s'agit, dans l'ensemble de ces cas, de problèmes de coordination liés aux interdépendances entres les différents composants des systèmes, les différentes utilisateurs de ces systèmes et des actions qu'ils produisent.

Nos travaux principaux dans ce domaine s'articulent autour de la gestion de ces interdépendances. Ils concernent les différentes familles de problèmes liés à la coordination dans les systèmes distribués coopératifs. Nous avons aussi conduit des études qui généralisent ces résultats ou qui les complètent dans le contexte plus large qui est celui des "logiciels orientés composants distribués", et des "approches de description et de conception". De façon générale, nous avons apporté des contributions aux trois axes de recherche suivants :

- *le formalisme de description*. Dans ce contexte, les contributions ont comporté :
  - la spécialisation ou l'utilisation de formalisme de base, comme les grammaires de graphes [7O] ou la logique du premier ordre [7Ci], pour représenter les interdépendances structurelles décrites par un graphe ou par un ordre partiel;
  - l'intégration de formalismes pour représenter à la fois les contraintes structurelles et fonctionnelles. C'est le cas pour notre réflexion initiale sur l'intégration des descriptions "processus communicants" et "transformation de graphes" [36Ci]. C'est le cas aussi pour la représentation, en Z, des règles de transformations de graphe gardées par des contraintes fonctionnelles [3Ci].
- *les architectures et applications logicielles orientées composants et distribuées*. Les contributions ont concerné :
  - la définition de cadres de conception ou d'intégration avec prise en compte des exigences applicatives de type pédagogique [4O] ou multimédia [4J].

  - la classification et la recherche de composants [7Cn];
  - le déploiement dynamique des composants et des applications
  - l'administration des applications distribuées [5Cn];
  - la gestion de l'évolution et de l'adaptation dynamique des architectures [31Ci].

- *les services et applications de support pour les activités coopératives*. Les domaines couverts sont celui de la télé-ingénierie et celui du télé-enseignement. Les contributions ont porté principalement sur des points suivants :

  - la gestion des groupes de coopération : en utilisant des approches orientées rôle adaptées à la coopération planifiée [14Ci], ou des approches basées sur les ontologies [2Ci] pour traiter le cas de la coopération spontanée ou non planifiée par synthèse et comparaison des profils.
  - la gestion des sessions de coopération impliquant les outils de coopération, les sites de coopération, et les participants [15Ci], [16Ci].
  - l'édition coopérative [3J].

Nous insistons dans la suite de cette introduction et de ce mémoire sur les contributions liées à la coordination dans les systèmes distribués coopératifs. Nous respecterons, dans leur présentation, la classification en deux classes principales représentant l'une ou l'autre des propriétés structurelles et comportementales des systèmes distribués coopératifs.

1.1.2.1 Cas des contraintes et exigences comportementales

Dans cette classe de contributions, nos travaux ont ciblé deux objectifs qui se complètent pour couvrir les deux niveaux fonctionnels de l'interaction [1] (coopération et communication) gérés par les fonctions de coordination, et nous avons développé des solutions architecturales différentes du modèle centralisé adopté pour la gestion des contraintes structurelles. Les objectifs ont concerné la gestion de la causalité dite "potentielle" qui peut relier les événements de communication de façon indépendante de leur contenu, ou la causalité dite "prédéfinie" qui relie les événements de coopération et tient compte de leur contenu. Les solutions architecturales développées sont orientées vers une distribution totale de la coordination, comme il est le cas pour [13Ci], ou vers une distribution partielle [15Ci]. Chacune des études a comporté le développement de modèles pour guider l'implantation ou pour la formaliser, la mise en œuvre (sous forme de protocole ou de service) et l'application à la télé-ingénierie. La réflexion dans ce contexte s'est déroulée essentiellement dans le cadre du projet européen DSE et a été conduite dans le cadre des thèses de Saul Pomarès et de Martin Molina.

Une solution algorithmique pour la coordination basée sur la communication de groupe causalement ordonnée a été développée dans le cadre de la thèse de Saul Pomarès. L'objectif de cette solution est de maintenir la causalité (potentielle) entre les événements de communication diffusés lors des sessions de coopération. Ceci permet d'éviter l'inconsistance de l'espace de coopération qui peut avoir lieu lorsque l'ordre causal n'est pas respecté. Différentes versions de protocoles ont

1. Pour alléger la rédaction, nous parlerons, aussi, et indifféremment de "niveau d'interaction" ou de "niveau fonctionnel".

été développées : mono-canal [1J] et multi-canal [3O]. Différentes implantations ont été réalisées : sur la base de la bibliothèque MCL [Roca,] et par extension de JSDT[2]. Cette dernière version a été utilisée pour coordonner la communication dans l'outil d'édition coopérative [11Ci].

Les travaux développés dans le cadre de la thèse de Martin Molina traitent les interdépendances au niveau coopération. Ce travail a permis d'identifier les différents types d'interdépendance (inter-acteurs, inter-composants, etc...), de définir un modèle pour les représenter, et d'implanter un ensemble de services pour les gérer de façon distribuée pendant l'exécution de la session de coopération. Dans un premier temps, les différentes catégories de dépendance dans les sessions multi-outils et multi-utilisateurs ont été identifiées (inhibition, déclenchement, et précédence) et décrite dans un modèle informel [10Ci]. Après cette première réflexion, un premier modèle abstrait a été développé [7Ci] suivi d'un modèle opérationnel proche de la structure distribuée de l'architecture du service de coordination des sessions dont il modélise le fonctionnement. Ce modèle classe les interdépendances en différentes catégories : internes, locales, et distantes, entre acteurs sur différents sites de coopération ou entre composants d'une même ou de deux applications différentes [1O].

1.1.2.2 Cas des contraintes et exigences structurelles

Dans ce contexte, nous avons notamment travaillé sur le développement et l'exploitation de modèles formels pour la coordination des applications coopératives distribuées. Nous avons démontré qu'il était possible d'utiliser un modèle unique pour gérer de façon intégrée (i) les problèmes de coordination (tels que le partage de l'espace de coopération) et (ii) l'évolution de l'architecture de l'application de façon cohérente avec les contraintes de coordination. Cette évolution se fait d'une façon qui permet de favoriser la coopération (par ex. renforcer la mémoire de groupe : "awareness") tout en maintenant la consistance du point de vue coordination (par ex. éviter les conflits lors du partage). En étudiant la conception des applications distribuées coopératives, nous avons aussi mis en avant une architecture qui permet une réutilisation et une adaptation des applications d'un point de vue à la fois fonctionnel et architectural. Nous avons en effet adopté une approche multi-niveaux qui distingue la communication, la coordination et la coopération. Nous avons d'une part montré qu'il était possible de réutiliser les mêmes règles de coordination pour différents supports de coopération. Nous avons notamment prototypé deux versions pour l'édition de documents sous différents formats (ASCII et HTML) en utilisant les mêmes règles de coordination. D'un autre côté, nous avons développé différents modèles de coordination (orientés structure de groupe, ou structure de l'espace de coopération) utilisables pour différentes activités de coopération comme l'édition et la revue coopératives.

Nous avons implanté les éléments génériques de cette architecture en différentes technologies. Nous l'avons d'abord instantiée dans le cadre de la programmation par processus communicants (utilisant le langage LCS basé sur le formalisme CCS de Milner) [32Ci]. Nous l'avons ensuite instantiée dans le cadre de la programmation par objets distribués en utilisant d'abord le langage Java pour programmer les objets [30Ci], et en généralisant ensuite à C++ en utilisant l'approche Corba [31Ci]. Dans chacune de ces deux implantations, nous avons développé des services génériques permettant de générer une partie des applications (celle relative à la coordination) et de faciliter la programmation de la partie restante (celle relative à la coopération). D'une part nous

2. Java Shared Data Toolkit

avons développé un service de communication de groupe (orienté message dans le cas de LCS et par invocation de méthode à distance dans le cas de Corba). D'autre part, nous avons développé un deuxième service qui gère la coordination en se basant sur des règles de fonctionnement décrites comme des transformations de graphes. La dernière partie générique est relative au niveau coopération. Il s'agit d'un ensemble de classes pour Java et Corba (respectivement des fonctions pour LCS) que le programmeur étend pour implanter le comportement des objets Java et Corba (respectivement des processus pour LCS). Ces deux implantations nous ont permis de prototyper un environnement de support pour la programmation des applications distribuées coopératives. Mais cette démarche concerne les applications que nous développons entièrement, et ne constitue pas une solution pour l'intégration d'outils de coopération existants. Pour ce dernier problème, nous avons travaillé sur la définition d'une architecture d'intégration multi-niveau respectant la même décomposition fonctionnelle que la précédente [4J]. Ces travaux ont débuté, d'abord dans le cadre des projets internes de l'équipe OLC du LAAS vers 1995, et se sont ensuite poursuivis dans le cadre d'un projet de l'appel d'offre Télécoms du CNRS entre 1997 et 1999.

Pour compléter notre réflexion sur les architectures, nous avons coopéré avec d'autres chercheurs pour l'application de nos solutions dans un contexte de coopération avec un contenu riche (multimédia [4O]) ou avec un comportement intelligent (partage de connaissance et agents : [6Ci], [2Ci]). Nos travaux dans le premier cas se sont déroulés dans le cadre du projet "Ingénierie coopérative" de l'équipe OLC du LAAS. Dans le second cas, nos travaux ont commencé en 1999 dans le cadre d'une coopération informelle avec l'Université de Campinas au Brésil et se poursuivent actuellement dans le cadre d'un projet de collaboration CAPES/COFECUB. Nous travaillons actuellement sur l'intégration des deux contextes de coopération (contenu riche et comportement intelligent) et sur l'expérimentation de leurs apports, séparément et après intégration, dans le cadre du projet "Virtual Manet" [4Ci], [1Ci].

1.2 Démarche commune à toutes nos contributions

La **démarche orientée modèle formel** de toutes nos contributions, durant les différentes périodes des activités de recherche, constitue une caractéristique importante de nos travaux.

Différents modèles ont été et sont développés et évalués en termes de structures de représentation des architectures et des comportements et en termes de règles de leur modification ou de leur interaction. L'élaboration des modèles est réalisée de façon à leur permettre d'être adaptés au niveau d'interaction ciblé (communication, coordination, ou coopération), et ce pour chacun des travaux présentés.

L'exploitation de ces modèles est conduite en tenant compte de l'objectif de la modélisation : validation (test ou vérification) analyse et évaluation (testabilité ou performances); conception de protocoles de niveau communication, ou de niveau coordination et coopération. En fonction de l'objectif et de la fonction sous étude, l'effort a consisté à modéliser l'architecture de la partie logicielle du systèmes étudié ou le comportement de ses composants, de façon à permettre une utilisation appropriée des modèles développés.

Nous avons manipulé ou introduit différents types de formalisme : opérationnel comme les systèmes de transitions (ST) ou les graphes de refus (GR), structurel comme les graphes de coordination (GC), logique comme la logique du premier ordre (LPO), fonctionnel comme le langage

de spécification Z ou les langages de programmation ML et son extension par les processus communicants en LCS.

Pour les contributions de niveau communication, nos modèles se sont basés sur les formalismes opérationnels ST et GR [42Ci] lors des travaux initiaux relatifs à l'analyse de la testabilité. Nos modèles se sont basés ensuite sur la relation de dépendance causale (déduite de la relation de dépendance immédiate) pour la coordination des événements [3O].

Pour le niveau coordination, nous avons introduit et utilisé les graphes de coordination dans [30Ci]. Nous avons ensuite enrichi les règles de transformation de graphe par des contraintes exprimées en pré et post-conditions. Nous avons défini une approche pour traduire l'ensemble des règles dans le langage Z [3Ci].

Pour le niveau coopération, et concernant la coordination des événements de coopération, nous avons utilisé la logique du premier ordre pour spécifier les différentes catégories d'interdépendance [1O]. Le tableau suivant résume cette classification.

niveau	formalisme/type	objectif
communication	STE,GR/opérationnel	analyse de testabilité, vérification
coordination	GC/structurel	synthèse des protocoles de gestion de l'architecture dynamique et du partage
	Z/fonctionnel	vérification des protocoles de gestion de l'architecture dynamique et du partage
coopération	LPO/logique	coordination des sessions de coopération

1.3 Organisation du rapport

Ce rapport est organisé de la manière suivante :

– le premier chapitre constitué par cette introduction a présenté l'historique et le contexte de nos travaux. Une première synthèse générale des contributions a été développée, dans cette introduction. Elle fournit une première vue générale de nos contributions. Toutes les contributions ne feront pas l'objet de développement dans les autres chapitres.

– un deuxième chapitre, après cette introduction, présentera nos contributions dans le domaine des systèmes communicants. Nous ne détaillerons pas les aspects techniques de nos travaux dans ce chapitre. Nous fournirons en annexe, une publication, en français, qui introduit les principaux aspects techniques de nos contributions dans ce domaine.

– En troisième chapitre, nous présenterons la problématique scientifique générale de la recherche sur les systèmes distribués coopératifs. Nous mettrons l'accent dans ce chapitre sur les problématiques liées à la recherche dans le domaine des logiciels de support des activités de coopération. Nous ciblerons plus particulièrement le problème de la coordination, et nous situerons nos contributions dans ce contexte.

– En quatrième chapitre, nous présenterons une synthèse des résultats de nos travaux sur la coordination et le support logiciel pour la coopération. Nous introduisons pour cela, une classification des différents types d'interdépendances en coordination. Nous utiliserons cette classification pour présenter, de façon structurée, nos propres contributions dans ce domaine.

– En cinquième chapitre, nous conclurons cette synthèse en dressant son bilan et sa prospective.

– En sixième chapitre, nous présenterons la liste de nos publications et travaux de recherche, d'encadrement d'enseignement et de formation.

– En septième chapitre, nous présenterons, une annexe contenant les détails techniques des implémentations logicielles de la gestion des architectures dynamiques utilisant la transformation des graphes de coordination.

– En huitième chapitre, nous présenterons, une annexe contenant les détails techniques de notre approche d'analyse de la testabilité avec traitement de la divergence.

1.4 Conventions bibliographiques

Nous utilisons la version française de la convention bibliographique "apalike" pour les travaux de la littérature. Le nom de l'auteur et l'année apparaissent entre crochets dans la citation.

Pour classer et distinguer nos publications, nous les citons et présentons sous différentes catégories. Nos publications apparaissent sous la forme

"[⟨*numéro dans la catégorie*⟩⟨*abréviation de la catégorie*⟩]".

Par exemple [9Ci] désigne l'article numéro 9 de la catégorie "Conférences internationales avec actes et comité de lecture".

Les catégories et leurs abréviations sont données dans la table suivante :

Catégorie	**Abréviation**
Thèse	T
Articles et préfaces dans des revues scientifiques avec comité de lecture	J
Contributions à ouvrages	O
Conférences internationales avec actes et comité de lecture	Ci
Conférences nationales avec actes et comités de lecture	Cn
Manifestations internationales sans actes et avec comité de lecture	M
Rapports de contrat	Rc
Rapports de recherche	Rr
Revues de vulgarisation	V
Edition d'ouvrage (Livre, Revue, Actes de Conférence)	E
Exposés sur invitation	Xi
Exposés sur proposition acceptée	Xp

2

Contributions dans le domaine des systèmes communicants

Cette partie présente nos activités de recherche dans le domaine des systèmes communicants. Ces activités ont porté notamment sur l'analyse et la validation, d'abord dans un contexte de description formelle purement logique, ensuite, dans un contexte de description probabilisée et temporisée du comportement logique.

2.1 Introduction

La validation des systèmes communicants est une étape importante du processus de conception. Elle précède la mise en exploitation de ces systèmes et permet de vérifier si leur implantation est conforme à leur spécification. Les approches utilisées diffèrent selon la nature du système à valider et selon le formalisme utilisé pour le modéliser ou pour le réaliser.

Pour les systèmes de communication, on peut distinguer principalement deux approches :

vérification de la conformité : lorsque l'implantation peut être manipulée sans contraintes (c'est à dire au même titre que sa spécification), la validation est habituellement appelée *vérification* de la conformité ; on parle alors d'approche *boîte blanche*.

test de la conformité : dans le cas contraire, l'implantation est considérée comme une *boîte noire* que l'on manipule via ses points d'interaction avec l'extérieur. Sous cette contrainte, la validation est habituellement appelée *test* d'une implantation.

Nos contributions ont concerné chacune de ces deux approches. Elles se sont déroulées durant la période doctorale et durant le début de la période post-doctorale.

Nous rappelons, dans la suite de cette section, les principales directions en rapport avec cette recherche et nous résumons les principaux résultats de la période post-doctorale.

2.1.1 Sujet des travaux de la période doctorale

Notre recherche, durant la période doctorale, a porté plus particulièrement sur la validation par le test ou la vérification des systèmes communicants. Un cadre formel unifiant des approches

issues des théories de test et des approches issues des théories de vérification a été défini. L'objectif a été l'exploitation des descriptions formelles pour : (1) la vérification par réduction, (2) la génération de séquences de test et (3) l'analyse de la testabilité des protocoles à architecture multi-composants, à la fois sous des contraintes d'accessibilité, d'observabilité et de commandabilité de ces composants et sous des contraintes de non déterminisme dans la description de leur comportement.

2.1.2 Contributions des travaux de la période post-doctorale

Nos contributions à ce domaine, durant la période post-doctorale, ont porté principalement sur l'*exploitation* des résultats obtenus pendant la thèse de doctorat et sur leur *extension*.

L'application des résultats de nos travaux initiaux a concerné l'utilisation du pouvoir d'abstraction des équivalences de test pour vérifier que le service fourni par le modèle d'une implantation d'un système de communication est conforme à sa spécification.

L'extension des résultats de nos travaux initiaux a concerné, d'une part, le traitement de la divergence, qui reflète la présence de comportements en boucle non observables dans la technique d'analyse de testabilité. D'autre part, l'extension a inclus la prise en compte des informations stochastiques dans la technique de vérification par réduction, allant ainsi d'une approche qualitative vers une approche, à la fois, qualitative et quantitative.

Le traitement de la divergence a été, d'abord, développé [8J] selon la théorie de test de Brinksma [Brinksma et al., 1987] et ensuite révisé [7J] selon la nouvelle théorie de test de Leduc. La génération sélective des séquences de test par exploitation des symétries a aussi constitué une extension de nos travaux sur la génération de test [38Ci]. Ce travail a été développé en collaboration avec François Michel dans le cadre de ses travaux de thèse [Michel, 1996] sous la Direction de Pierre Azéma.

Pour étendre les approches de validation qualitative, nous avons défini en collaboration avec des spécialistes des spécifications stochastiques, et implanté, une nouvelle équivalence observationnelle stochastique [40Ci]. Nous avons mis les sources des logiciels développés pour la vérification et l'analyse de testabilité à la disposition des doctorants qui ont continué à développer ces axes de recherche au LAAS et au LABRI.

La suite de ce chapitre est consacrée à la présentations des résultats de recherche de la période post-doctorale dans le domaine des systèmes communicants.

2.2 Analyse de la testabilité

Telle que définie en introduction de ce chapitre, la validation par le test est plus générale mais moins discriminante pour les erreurs d'implantation qu'une vérification boîte blanche. Ceci s'explique par les contraintes d'accès et de manipulation que suppose la réalisation de la procédure de test.

Des travaux de recherche se sont penchés sur la capacité théorique du test de conformité à détecter les implémentations erronées. Dans le domaine des systèmes communicants, ces travaux se sont exclusivement basés sur une représentation par machines à états finies et ont considéré des descriptions monolithiques représentant des systèmes entièrement accessibles (voir notamment la synthèse de ces travaux dans [Vuong et al., 1993]).

Il s'avère en pratique que le système sous test ne peut pas toujours être librement commandé et observé. On est alors dans le cas d'un système partiellement accessible (en observation ou en commande). Ceci est notamment le cas lors du test de conformité de certains éléments d'un système multi-composants. Ce contexte constitue les motivations principales du travail présenté dans cette section. Nous visons plus particulièrement à formaliser la diminution du pouvoir de détection de la non-conformité par le test -ce que nous appelons « dégradation de testabilité »- et à évaluer cette dégradation pour un système et un environnement donnés.

Les originalités de notre approche sont multiples. D'abord nous ne nous limitons pas à une analyse purement structurelle [1] comme il était le cas dans la plupart des travaux existants (voir notamment [Dssouli et Fournier, 1990, Ghedamsi et al., 1992, Petrenko et al., 1993, Vuong et al., 1993]). Nous ne nous limitons pas non plus à une analyse purement comportementale concernant le système sous test uniquement ou le système sous test et son environnement. Les résultats d'analyse de notre approche tiennent compte simultanément de l'architecture du système sous test, du comportement de ce dernier ainsi que du comportement de son environnement. L'approche développée utilise le contexte de description formelle du comportement et les opérations algébriques de composition à la LOTOS [2]. Ceci offre l'avantage de pouvoir appliquer notre approche à tout formalisme qui utilise la sémantique de composition de rendez-vous, et ce dès les premiers stades de la conception. En effet nous admettons la possibilité de non déterminisme dans les descriptions du comportement attendu, ce qui est utile lorsque l'on désire faire abstraction de détails et de choix d'implémentation. Enfin, même si notre approche repose implicitement sur un modèle de fautes, celles-ci ne sont pas exhaustivement analysées. Un ordonnancement partiel de ces fautes (ordonnancement qui est compatible avec la définition de conformité utilisée) permet de considérer une analyse hiérarchique qui s'arrête sur des comportements erronés « limites » qui, ainsi que toute implémentation plus erronée, sont détectables alors que toutes les implémentations strictement moins erronées ne le sont pas.

Nous avons, pour cela, proposé de formaliser le *test d'un système communicant à travers son environnement* [14Ci], [8J], [7J].

2.2.1 Traitement de la divergence

Notre première approche d'analyse de testabilité a reposé initialement [14Ci] sur une définition de conformité qui suppose que le modèle de l'implantation ne peut pas indéfiniment évoluer sans communiquer avec son environnement de test (la divergence n'était pas admise). En effet, lorsque la restriction, c'est à dire le choix des actions observées, crée une divergence après une séquence d'actions, il n'est plus possible de calculer les ensembles de refus associés à cette séquence dans la restriction d'un graphe de refus à partir des ensembles de refus initiaux. En pratique ceci se traduit par le fait que le calcul de l'ensemble de refus d'un groupe d'états du graphe de refus

1. C'est à dire une analyse qui ne considère que des critères de testabilité « a priori » basés sur l'architecture du système sous test.

2. « Language Of Temporal Ordering Specification » est un langage de description algébrique normalisé par l'OSI, l'Organisation de Standardisation Internationale. Il constitue la base de plusieurs travaux sur la validation et fait l'objet actuellement de projets d'extensions pour considérer entres autres, les aspects temps réel de certaines applications dans le domaine des télécommunications. Une introduction à ce langage est fournie dans [Bolognesi et Brinksma, 1988]. Ce langage s'inspire de CCS de Milner [Milner, 1980], et de CSP de Hoare [Hoare, 1985] les formalismes algébriques les plus utilisés actuellement.

restreint n'est pas forcément correct lorsqu'il y a divergence à l'intérieur de ce groupe, c'est à dire lorsqu'il y a entre ces états, un cycle étiqueté par des actions rendues non observables.

Pour que les résultats de l'analyse de testabilité soient valides même lorsque le comportement analysé contient des états de divergence, nous nous sommes intéressés au traitement de la divergence dans l'approche d'analyse de testabilité. Plusieurs solutions sont possibles en théorie pour tenir compte de la divergence dans la représentation des comportements.

La solution que nous avons développée permet d'assurer la préservation de la conformité en présence de divergence, et supporte le calcul de la composition et surtout de la restriction d'un graphe de refus, ce qui est essentiel pour l'algorithme d'analyse de testabilité qui est basé sur ces deux opérations ([7J]). Notre interprétation de la divergence est différente de l'interprétation "catastrophique" utilisée dans la sémantique de certaines théories de concurrence.

L'interprétation catastrophique de la divergence selon De Nicola Hennessy et Hoare [Nicola et Hennessy, 1984, Hoare, 1985] suppose qu'un processus qui diverge après la trace σ (en présentant un cycle d'actions non observables), sera considéré comme divergent après toutes les traces qui contiennent σ. Cette interprétation n'est pas réaliste dans le cas de notre représentation états/transitions, car un même état peut être atteint par différentes séquences et pourrait ainsi être jugé divergent même s'il ne représente pas de cycle d'évolution internes. Ainsi dans le système de transitions de la figure 2.1, l'état initial, bien que ne présentant pas de possibilité d'évolution infinie par les transitions internes τ, sera considéré divergent selon l'interprétation catastrophique car il est atteint par la séquence $a.b$ qui passe par l'état divergent 2.

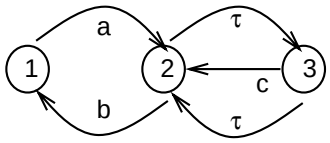


FIG. 2.1 – *Divergence dans un système de transitions. Les actions a,b et c dénotent des transitions externes. L'action τ dénote les transitions internes.*

Comme Leduc [Leduc, 1991], nous avons choisi une définition moins restrictive qui considère que la divergence est associée à l'arrivée dans un état divergent et non à la possibilité de passer par un état divergent. Ainsi pour le système de transitions de la figure 2.1, seuls les états 2 et 3 sont considérés divergents.

Contrairement à Leduc, nous ne distinguons pas et pour des raisons purement pratiques, entre blocage[3] et divergence. En effet dans les deux cas, le système ne communique pas avec son environnement. Nous considérons alors qu'une divergence dans la spécification permet de juger conformes les implantations qui bloquent après l'exécution de la séquence qui aboutit à cette divergence.

Dans [8J] nous avons étendu notre approche formelle d'analyse de testabilité pour tenir compte de la présence éventuelle d'exécutions divergentes[4] qui aboutissent à un état où le système n'est pas bloqué alors qu'aucune communication avec ce système n'est possible.

3. Le blocage relatif à un ensemble d'actions à partir d'un état donné désigne l'absence d'actions de cet ensemble à partir de cet état. Le blocage désigne l'absence de toute action, et donc toute transition, la transitions interne comprise.

4. Le terme « divergence » est utilisé pour désigner l'état d'un système qui évolue indéfiniment sans que cette évolution ne soit perceptible par l'utilisateur. Lorsqu'on représente le comportement d'un système communicant par

Dans [7J], nous avons révisé notre approche formelle d'analyse de testabilité dans le cadre de la théorie de test de Leduc [Leduc, 1995] qui étend la théorie de test de Brinskma pour tenir compte de la divergence. Nous avons montré que notre définition de la conformité avec divergence pouvait se ramener, sous l'hypothèse que seules les spécifications peuvent contenir des traces divergentes et que les implémentations n'en contiennent pas, à l'une des définitions que Leduc a introduit pour généraliser la relation de conformité de Brinksma qui ignorait les spécifications divergentes [Brinksma, 1988, Brinksma et al., 1987].

2.3 Vérification par réduction

Cette partie résume nos contributions concernant la vérification par réduction. Il s'agit de l'application de nos résultats initiaux développés durant la période doctorale, de leur extension quantitative et de leur extension logique.

2.3.1 Validation et analyse des propriétés logiques

Nos travaux initiaux de recherche ont concerné les techniques de vérification et d'analyse dans le cadre des techniques algébriques notamment CCS et LOTOS. Nous avons défini et proposé les graphes de refus [42Ci], [47Ci], [9J] (que l'on peut voir comme des systèmes de transition déterministes doublement étiquetés), pour :

– fournir des procédures de calcul qui permettent de ramener la vérification d'une relation d'implémentation, de conformité, ou de test à un calcul de bissimulation. Le concept de bissimulation de Park et Milner est utilisé pour définir et calculer l'équivalence de deux ou plusieurs modèles d'un même système [42Ci].

– transformer et réduire les descriptions par systèmes de transitions tout en préservant les propriétés caractérisées par la relation d'équivalence de test. Ceci permet à un concepteur d'avoir une vue simplifiée de son modèle qui tient compte de toutes les séquences d'exécution possibles ainsi que des éventuels blocages relatifs à ces séquences. Ces blocages sont dus à des évolutions internes au système modélisé et qui sont donc non observables et non commandables par l'utilisateur [43Ci], [44Ci].

2.3.2 Validation des propriétés logiques probabilisées

Dans le cadre de nos activités de recherche concernant la validation des propriétés liées aux performances dans les systèmes communicants, nos travaux ont concerné l'extension des méthodes d'agrégation dans les domaines de description formelle «classique» [5] et les méthodes stochastiques, et notamment probabilistes. Le changement d'état dans les systèmes modélisés est alors dicté par une interaction entre ce système et son environnement (réception/émission d'un

un graphe étiqueté fini, une évolution infinie se traduit généralement par la présence de cycles de transitions dites non observables. Ces transitions sont désignées souvent par τ dans les formalismes dérivés des algèbres de processus.

5. Ces techniques considèrent des propriétés relevant purement la logique des interactions entre le système et son environnement : séquencement, causalité, parallélisme, etc...

message) ou par l'exécution d'une action en interne [6] qui constitue donc le premier attribut des transitions qui représentent le changement d'état. L'extension probabiliste introduit un attribut supplémentaire de nature quantitative qui représente une probabilité d'occurence de cette transition. Enfin un autre attribut de nature quantitative est aussi considéré. Il représente le temps moyen de séjour dans l'état de départ avant que la transition ne puisse avoir lieu.

Nous avons alors unifié le concept d'équivalence observationnelle avec la notion d'agrégation Markovienne et défini une nouvelle relation d'équivalence appelée Markovienne. Sur la base de cette nouvelle définition, nous avons proposé une technique permettant de calculer la plus petite partition [7] qui en même temps vérifie la propriété d'agrégation Markovienne et préserve les propriétés logiques liées aux séquencement des actions [40Ci].

2.3.3 Application au protocole OSI-TP

L'application des résultats de nos travaux sur la vérification a concerné l'utilisation du pouvoir d'abstraction des équivalences de test pour vérifier que le service fourni par le modèle d'une implantation d'un système de communication est conforme à sa spécification. Il s'agit notamment des travaux de thèse de Raul Jacinto Montès [Jacinto, 1995] sous la direction de Guy Juanole et concernant la modélisation du protocole OSI-TP [44Ci].

2.4 Génération de séquences de test

La génération sélective des séquences de test par exploitation des symétries [38Ci] a constitué une extension de nos travaux initiaux sur la génération de test [47Ci], [48Ci]. Ce travail a été développé en collaboration avec François Michel dans le cadre de ses travaux de thèse sous la Direction de Pierre Azéma. Les contributions ont porté sur l'exploitation des symétries de données et d'architecture pour l'optimisation de la phase de génération de séquences de test. Il s'agit de fournir, à la génération, une séquence de test représentative d'un ensemble de séquences équivalentes qui ont le même pouvoir de discrimination pour les même erreurs d'implantation (à la symétrie près).

2.5 Conclusion

Nos contributions dans le domaine de la vérification et du test de conformité des systèmes communicants ont porté sur les modèles du comportement des composants. Nous avons exploré les différents problèmes liés au manque d'accessibilité et d'observabilité en considérant tant les problèmes inhérents à la méthode de validation (contraintes de test en approche boite noire) que les problèmes supplémentaires introduits par la conception (contraintes d'accessibilité et d'observabilité d'un composant particulier).

Nos travaux ultérieurs complètent ces contributions par des modèles de conception pour les logiciels distribués coopératifs. La complexité de ce contexte réside dans le passage des architec-

6. L'admission de cette notion d'action interne exige de notre approche le traitement du non-déterminisme dans les modèles.

7. définie par une relation d'équivalence sur l'espace d'états d'un graphe probabilisé que l'on obtient généralement à partir d'un langage de description stochastique.

tures statiques aux architectures dynamiques, de la communication point à point à la coopération en groupe et de la synchronisation à la coordination des composants. Cette recherche a introduit de nouveaux problèmes liés à l'intégration et à la distribution des composants sous différentes contraintes architecturales : interdépendance, dynamisme et distribution.

3

Les problématiques de recherche sur les systèmes distribués coopératifs

3.1 Introduction

Les études organisationnelles et les analyses des processus d'ingénierie distribuée et des autres activités coopératives dans les domaines de la formation, de l'expertise ou de l'enseignement à distance montrent les besoins des entreprises en solutions pour l'organisation de leurs activités autour d'un espace de travail que l'on peut appeler virtuel partagé ou distribué. Dans cette ère de compétitivité à l'échelle mondiale, l'espace économique témoigne d'un mouvement de réorganisation sans cesse, dans lequel les entreprises se globalisent et s'internationalisent [1] [Mol et Koppius, 2002]. Et ceci aboutit dans les deux cas à une organisation en équipes, unités ou groupes de travail de plus en plus géographiquement distribués.

Les procédures de conception et d'ingénierie sont en train d'être redéfinies pour s'adapter à la nouvelle organisation en anticipant des solutions permettant d'éviter les inconvénients de cette nouvelle organisation, d'un côté, et pour tirer bénéfice de ses avantages d'un autre côté. Ceci a entraîné un besoin de recherche portant sur une nouvelle organisation des activités de groupe, de nouveaux logiciels génériques utilisés dans différents domaines d'activités et de nouvelles architectures pour les logiciels spécifiques à un domaine d'activité particulier.

Différents domaines de recherche sont concernés par cette problématique et sont classés sous le thème pluridisciplinaire du travail coopératif assisté par ordinateur, CSCW. Nous proposons tout au long de ce chapitre de faire la synthèse de leurs spécificités, similitudes et différences, en les situant entre eux et en les situant par rapport au domaine voisin que constitue la branche coopérative de l'Intelligence Artificielle Distribuée (IAD ou DAI en anglais) .

En conclusion de ce chapitre, et avant de présenter nos contributions dans le chapitre suivant, nous proposons de les situer, d'abord en adoptant des classifications reconnues pour les travaux concernant les logiciels de support à la coopération en général, et pour les travaux relatifs à la coordination plus particulièrement. Nous proposons ensuite une classification plus fine des différentes catégories d'interdépendances permettant de situer l'ensemble de nos contributions dans le domaine de la coordination dans les systèmes distribués coopératifs.

1. En moins de deux ans, l'aérospatiale s'est restructurée deux fois, en se regroupant d'abord avec son partenaire français Matra et ensuite avec de nouveaux partenaires européens pour former le nouveau groupe EADS.

3.2 Les problématiques visées dans les différents domaines de recherche

Plusieurs équipes de recherche des mondes académiques et industriels se sont penchées sur les nouvelles problématiques engendrées par la distribution des activités d'ingénierie traditionnelles et par la création de nouvelles procédures de travail en groupe. La multiplicité des besoins, tant du point de vue organisationnel que logiciel, pour ces réflexions, en fait un domaine de recherche multidisciplinaire dans lequel nous pouvons distinguer différentes disciplines, différentes applications et différents points de vues pour chaque discipline et pour chaque application.

Les principales disciplines concernées pourraient se classer sous deux principales catégories : **l'informatique des logiciels de support**, et l'**organisation du travail en groupe**. Le **CSCW** est un domaine multi-disciplinaire qui regroupe des contributions de ces deux catégories sous le thème du **travail coopératif assisté par ordinateur**. L'organisation du processus de coopération, la sociologie de la coopération de groupe, et l'ergonomie des systèmes de support sont des branches du CSCW à dominante organisationnelle ou ergonomique. Les branches de l'informatique qui sont directement concernées par, ou liées à l'aspect "distribution'" et/ ou "coopération" sont : l'intelligence artificielle distribuée, les interfaces homme/machine, l'architecture des logiciels distribués, et plus généralement les systèmes informatiques distribués.

3.2.1 La branche organisationnelle

Les problèmes étudiés par les travaux de la discipline de l'organisationnel concernent la coopération homme/homme et s'intéressent à l'étude des activités assistées par ordinateur. La théorie de structuration est à la base de nombreux résultats dans ce domaine. Les problèmes étudiés appartiennent généralement à l'une des trois classes suivantes :

- le partage de la connaissance,
- la décision de groupe,
- la communication.

3.2.2 La branche de l'informatique

Les principales spécialités de l'informatique qui contribuent au domaine des systèmes informatiques distribués et coopératifs sont l'intelligence artificielle, les systèmes d'information et l'ingénierie du logiciel.

3.2.2.1 Cas de l'intelligence artificielle

Les principaux problèmes étudiés en intelligence artificielle, dans ce contexte, concernent principalement la résolution de problèmes ("problem solving"). Le premier axe étudié dans ces travaux concerne les protocoles de coordination pour maintenir la cohérence de la connaissance distribuée et garantir la convergence [Durfee et al., 1985a]. Le 2ème point étudié dans ces travaux concerne le langage utilisé par les agents coopérant pour s'échanger les informations. Le standard KQML a été défini pour normaliser ces échanges [Finin et al., 1993].

3.2.2.2 Cas de l'ingénierie du logiciel

Les problèmes étudiés par les travaux de la discipline du logiciel, dans le contexte coopératif, couvrent principalement les domaines suivants :

- la communication multimédia
- l'algorithmique pour la gestion de la distribution des composants du système logiciel et la coordination des événements générés par les interactions de niveau système-système ou utilisateur-système,
- la gestion des architectures distribuées. Et notamment les problèmes liés à :
  - l'adaptabilité : tant au niveau de la mobilité des utilisateurs, des différents rôles et profils des utilisateurs et à leur évolution, des changements des règles de coordination, des changements des objectifs de coopération, aux changements des supports de communication, que des changements de l'échelle du groupe (”scalability”)
  - le déploiement des services et des outils : nécessitant par exemple, l'utilisation du web et des navigateurs comme clients universels.

Différents domaines d'application sont concernés, par exemple :

- l'ingénierie coopérative : conception ou production (projet DSE)
- la télémaintenance (projet Proteus [IST-Proteus, 2003], [Garcia et al., 2004])
- l'enseignement distribué coopératif (projet Topase)
- la télémédecine.

Différentes catégories d'activités sont génériques et communes à certains ou tous ces domaines d'application :

- la production coopérative de documents,
- le contrôle des applications à distance,
- la gestion des connaissances de groupe.

Différents logiciels de support sont utiles à certaines voire toutes ces catégories d'activités :

- la discussion de groupe : conférence textuelle, ou audiovisuelle.
- la rédaction et la révision de documents : les 2 fonctions principales sont l'édition et l'annotation
- l'exécution d'applications en groupe : ces applications peuvent être des logiciels de groupe ou des logiciels mono-utilisateur virtuellement distribués par des logiciels de partage d'application.

Deux types d'interaction sont distingués pour chacun de ces outils :

- interaction synchrone : présence simultanée des membres du groupe, espace de travail partagé, mise à jour immédiate des objets distribués,
- interaction asynchrone : présence différée des membres du groupe, espaces de travail individuels, mise à jour périodique des objets distribués.

3.3 Les niveaux fonctionnels : consensus intra et inter domaines

Les études menées dans le cadre des différents domaines de recherche amènent à la distinction de trois niveaux fonctionnels (appelés aussi espaces fonctionnels dans le modèle du trèfle élaboré par les travaux de recherche relatifs au collecticiel de [Salber et al., 1995])

- le niveau coopération (appelé aussi espace de production dans [Salber et al., 1995] ou collaboration dans d'autres travaux)
- le niveau coordination
- le niveau communication.

Cette décomposition fonctionnelle en trois niveaux est présente :

- dans les travaux relatifs à la résolution de problèmes par des agents intelligents développés par les travaux de recherche du domaine de l'intelligence artificielle [Durfee et al., 1985b].
- dans les travaux relatifs à la description des interactions dans les collecticiels (groupware) développés par les travaux de recherche du domaine de l'ingénierie de l'interaction (nouveau domaine de recherche multidisciplinaire qui englobe les domaines de l'ergonomie et de l'ingénierie logicielle des Interfaces Homme Machine) [Salber et al., 1995];
- dans les travaux relatifs à l'organisation, à l'ergonomie et à la sociologie développés par les travaux de recherche du domaine du CSCW ("Computer Supported Cooperative Work" ou Travail Coopératif Assisté par Ordinateur en Français) [Conen et Neumann, 1998];
- dans les travaux d'application à des domaines particuliers comme l'ingénierie coopérative du logiciel [H.Krasner et al., 1991].

Certains travaux du domaine du CSCW [Coleman, 2002] distinguent la communication (où les messages ont une sémantique simple et vont d'un émetteur vers un récepteur avec éventuellement acquittement), l'interaction (qui résulte d'une situation de communication où chaque participant peut jouer le rôle d'émetteur et de récepteur et où la sémantique de l'information est complexe) et la collaboration (définie comme des interactions multiples entre 2 ou plusieurs participants qui s'échangent des informations complexes pendant une durée de temps donnée). Certains autres travaux de recherche issus du domaine industriel, proposent d'autres décompositions. [McCormack, 2001] définit une approche plus détaillée qui part du niveau "insulaire" (isolé) jusqu'au niveau "coopératif interactif" en passant par le niveau "transactionnel". Le niveau transactionnel distingue à

son tour, 3 sous niveaux intermédiaires: “le transactionnel” , “le transactionnel coordonné” et le “transactionnel coopératif”.

Deux niveaux d’information sont manipulés par les logiciel coopératifs : le premier correspond à celles que nous appelons “information de coopération” [31Ci] (appelées “subject-matter information” dans [Mills, 2003]) et le deuxième correspond à celles que nous appelons “informations de coordination” [31Ci] (appelées “collaboration-support information” dans [Mills, 2003]).

Dans sa synthèse des travaux du domaine du CSCW, [Mills, 2003] distingue les domaines listés dans le tableau 3.1.

Communication	Asynchrone, audio, données, privée, partagée, structurée, synchrone, texte, non structurée, vidéo
Configuration	Adaptation, composition, évolution, extension
Coordination	Contrôle d’accès, concurrence, consistance, délégation, planification, gestion de versions accès à l’ information, distribution, filtrage, retrait, structure
Interaction	“Attention management”, awareness, gestion de contexte, maintient et établissement des relations
Utilisabilité	Traverser les frontières (Boundary crossing) (cyberespace, espace physique, espace logique), interaction entre différents dispositifs en différents modes

TAB. 3.1 – *Cinq domaines de conception pour le CSCW et les concepts clefs associés (source : [Mills, 2003])*

Dans leur article de synthèse [Ellis et al., 1991, page 40], qui constitue un des premiers articles de référence dans le domaine de Groupware , Ellis, Gibbs et Rein soulignent l’importance de la communication, de la collaboration et de la coordination. Ils soutiennent que “l’interopérabilité (ou la fusion) entre les systèmes de télécommunication et les systèmes informatiques constituerait l’efficacité de la communication pour le groupware. Les auteurs qualifient la collaboration de pièce maîtresse pour les activités de groupe. Les auteurs précisent qu’une collaboration effective exige un partage d’information entre les personnes impliquées, que la coordination permet d’améliorer l’efficacité de la collaboration et de la communication.

D’après [Ellis et al., 1991], l’objectif du groupware est d’assister les groupes dans leur communication, dans leur collaboration, et dans la coordination de leurs activités. Le groupware y est défini comme un système informatique qui assiste des groupes de personnes engagées dans une tâche commune (ou objectif commun) et qui fournit une interface à un environnement partagé. Ellis et al mettent en avant les notions de “tâche commune” et “d’environnement partagé” dans leur définition et insistent sur l’importance de ces deux notions.

3.4 La coopération et la collaboration

3.4.1 Coopération synchrone vs coopération asynchrone

La présence simultanée ou différée des participants à une activité de coopération permet de qualifier la coopération, respectivement de synchrone ou d’asynchrone. Cette distinction est ré-

percutée sur les logiciels de coopération utilisés. Ellis, Gibs et Rein [Ellis et al., 1991, page45] proposent deux taxonomies pour les différentes catégories de logiciels de coopération. La première taxonomie, la plus connue, est basée sur les notions de temps et d'espace. La 2ème taxonomie est fonctionnelle et se base sur le type de partage : "orienté tâche" ou "orienté environnement de travail".

Les logiciels de coopération peuvent appartenir à l'une des quatre catégories de la taxonomie spatio-temporelle, à savoir :

- activité de type "même place/même moment" soutenue par l'interaction de type face-à-face.
- activité de type "même place/différents moments" soutenue par l'interaction asynchrone.
- activité de type "différentes places/même moment" soutenue par l'interaction synchrone distribuée.
- activité de type "différentes places/différents moments" soutenue par l'interaction asynchrone distribuée.

Pour la classification fonctionnelle, Ellis et al. distinguent :

- les activités à faible besoin de partage de tâche, comme celles soutenues par les systèmes temps partagé.
- les activités à fort besoin de partage de tâche, comme celles soutenues par le systèmes de revue de logiciel.
- les activités à faible besoin de partage d'environnement, comme celles soutenues par messagerie électronique (e-mail).
- les activités à fort besoin de partage d'environnement, comme celles soutenues par les salles de réunion virtuelle.

3.4.2 Collaboration vs Coopération

Les définitions des termes "collaboration" et "coopération" changent d'un domaine à l'autre et d'un auteur à l'autre au sein d'un même domaine. Il ressort néanmoins une différenciation autour du partage des objectifs : objectif global partagé ou objectifs individuels, et autour du partage des tâches ou leur regroupement. Un consensus (parfois inversé par certains auteurs) semble se former autour de la distinction entre les activités en groupe organisées et coordonnées, auquel cas il s'agit de coopération, et les activités de groupe spontanées et autogérées par des règles implicites, auquel cas, il s'agit de collaboration. C'est ce que l'on peut appeler respectivement coopération guidée par des structures "fonctionnelles" ou coopération guidée par des structures "interprétées" selon les deux grandes familles de théories organisationnelles citées, dans [Poole et DeSanctis, 2003], par Marshall Scott Poole, l'auteur de la théorie de structuration utilisée et étendue par de nombreux travaux.

Dans le domaine de l'enseignement coopératif assisté par ordinateur (CSCL), [Roschelle et Teasley, 1995], cité par [Brna, 1998], considère que le travail coopératif est accompli par le partage de l'effort entre les participants, et qu'il s'agit d'une activité où chaque personne est responsable

d'une partie de la résolution du problème. D'un autre côté, il considère que le travail collaboratif implique l'engagement mutuel des participants dans un effort coordonné pour résoudre le problème ensemble.

Dans le même domaine, et d'après [Dillenbourg et al., 1995], cité dans [Sinia, 2002], la coopération et la collaboration ne différent pas en termes de distribution ou de non distribution de la tâche, mais en vertu de la façon dont elles sont décomposées. En coopération, la tâche est décomposée (hiérarchiquement) en sous tâches indépendantes ; en collaboration, les processus cognitifs peuvent être (hiérarchiquement) décomposés en couches entrelacées. En coopération, la coordination est nécessaire seulement lors de l'assemblage des résultats partiels alors que la collaboration est une activité synchrone coordonnée qui est le résultat d'une tentative permanente pour construire et pour maintenir une conception partagée du problème.

Dans [Panitz, 1996], l'apprentissage coopératif est défini par un ensemble de processus qui aident les gens à interagir entre eux dans le but d'accomplir un but spécifique ou de développer un produit final qui est généralement spécifique au contenu de la formation. L'auteur soutient que ce mode d'apprentissage est plus directif qu'un système collaboratif car il est étroitement contrôlé par l'enseignant. Il y existe différents mécanismes pour l'analyse et l'introspection en groupe, mais dans son fondement, l'approche coopérative reste centrée autour de l'enseignant alors que l'approche collaborative est centré autour de l'apprenant. D'après [Panitz, 1996], la collaboration est une philosophie d'interaction et un style de vie personnel où les individus sont responsables de leurs actions (y compris l'apprentissage) et respectent les compétences et les contributions de leurs partenaires. La coopération est une structure d'interaction conçue pour faciliter la réalisation d'un produit ou d'un but final à travers le travail en groupe de différents individus.

Dans le domaine de l'intelligence artificielle distribuée, nous pouvons citer [Tonino et al., 2002] qui définit la coopération comme une fusion où les agents ont décidé de partager leurs ressources et objectifs : les agents ne coopèrent au sein de cette fusion que si les intérêts communs ne diminuent pas. D'un autre côté, la collaboration est un cas de fusion particulier dans lequel les agents ne coopèrent que si leurs intérêts individuels ne diminuent pas.

Nos contributions au support des activités en groupe se partagent entre les deux catégories que distinguent les définitions de la coopération et de la collaboration. En considérant que la coopération est une activité en groupe organisée et coordonnée, nous pouvons classer nos contributions autour de la coordination sous cette catégorie. Sous la deuxième catégorie, la collaboration, qui caractérise les activités de groupe spontanées et autogérées, se classent nos contributions par les différentes versions d'outils d'édition partagée.

3.4.3 Les théories de coopération

Les travaux sur la coopération font référence à des théories d'aide à la décision comme la "théorie du choix rationnel" (Rational Choice Theory) , "la théorie de la capacité limitée de traitement" (Bounded Rationality Theory) et la "théorie de la prospective" (Prospect Theory).

Dans [Lesser et Corkill, 1981, Bird et Kasper, 1995], les auteurs font référence à la théorie de la "capacité limitée de traitement" élaborée par H.A. Simon. Cette théorie reflète la limitation des capacités humaines dans le traitement et l'analyse des informations, limitation qui a de sévères implications sur la qualité des décisions prises en présence d'une large quantité d'incertitudes [Lesser et Corkill, 1981, p. 89].

De nombreux autres travaux font référence à des théories organisationnelles, et notamment la "théorie de structuration" (Structuration Theory). Selon [Poole et DeSanctis, 2003], l'argument principal de la "théorie de structuration" réside dans le constat que la structure sociale existe dans les actions des agents humains puisqu'ils utilisent des structures existantes et en inventent d'autres dans leur vie de tous les jours. Selon les mêmes auteurs, la théorie de structuration recherche la complémentarité entre deux extrêmes philosophiques qui traditionnellement ont été considérées comme incompatibles : "le fonctionnalisme" et "l'interprétivisme".

En effet, dans les "approches orientée fonctionnalisme", la coopération se base fortement sur des structures définies par avance. L'effort est alors dans la définition des structures génériques et adéquates. **C'est l'approche que nous avons adoptée pour soutenir la coopération.** Alors que dans les "approches orientées interprétivisme", la coopération se base fortement sur les capacités des participants pour inventer une structuration en utilisant des connaissances ou des règles sociales (social scripts). Les structures sont formées, dans la tête des participants, de façon cognitive à travers l'expérience de travail en commun. C'est généralement l'approche adoptée dans les travaux de l'intelligence artificielle. C'est effectivement aussi la conclusion de la synthèse de Ellis, Gibs et Rein [Ellis et al., 1991, page45] qui considèrent que l'approche de l'intelligence artificielle est généralement heuristique ou augmentative, et permet le raffinement et la précision des informations à travers des interactions utilisateur/machine plutôt qu'un départ à partir d'information structurée et complète.

3.4.4 Les logiciels de support à la coopération

Les travaux présentés dans [Bird et Kasper, 1995], décrivent les résultats d'une étude sur la "formalisation des problèmes de coopération dans les systèmes coopératifs". Les deux auteurs de cette étude sont des spécialistes en organisation du travail pour l'un et en informatique pour l'autre. Contrairement à d'autres études, celle-ci s'efforce de traiter de façon uniforme les problèmes de coopération étudiés dans les domaines CSCW et DAI. Cette étude forme ainsi une base de formalisation que l'on peut appliquer systématiquement que les entités coopérantes soient des agents artificiels ou des êtres humains.

D'après cette étude, il est commun de définir la coopération comme "l'interaction entre de multiples entités intelligentes pour la résolution de problèmes complexes". Selon les même auteurs, la "résolution de problèmes en coopération" est, par définition, le but commun des systèmes CSCW et DAI. Elle n'est pas seulement une activité partagée, mais une activité partagée qui exige une décomposition effective des problèmes et une bonne distribution sur les différents intervenants selon leurs compétences et leurs moyens.

Dans les travaux de [Lesser et Corkill, 1981], les auteurs se placent sous l'hypothèse de la "capacité de traitement limitée" et prévoient de compenser cette limitation par des variations dans les structures organisationnelles (en terme de type, de fréquence et de schémas (patterns) de connectivité pour le flux d'information)" [Lesser et Corkill, 1981, p. 89]. Dans [Bird et Kasper, 1995], les auteurs adoptent, de leur côté, le même raisonnement et proposent la décomposition du problème à résoudre en différents sous-problèmes. Chaque sous-problème est spécifique à un domaine bien défini. Les sous-problèmes sont caractérisés par différents degrés de dépendance et de séparation. Un certain nombre d'agents humains et artificiels - chacun ayant sa propre connaissance, objectifs, stratégies, et styles cognitifs - développent, alors, une sorte de fabrique d'interaction coopérative

afin de résoudre d'abord les différents sous-problèmes et ensuite le problème global lui-même... [Bird et Kasper, 1995, p 231].

L'analyse faite par [Bird et Kasper, 1995] propose le terme "coopératif" pour décrire des systèmes composés de différentes combinaisons d'agents humains et artificiels appliquées à différentes situations de résolution de problèmes. Ces situations peuvent correspondre à différents domaines d'application comme les trois domaines couverts par [Lesser et Corkill, 1981] et qui sont : l'interprétation distribuée, les réseaux de contrôle et la planification distribuée.

Cette analyse des systèmes coopératifs est partagée par d'autres travaux dans d'autres domaines. Nous pouvons aussi citer les travaux de [Roschelle, 1992] dans le domaine du CSCL autour de l'étude des technologies de coopération. Ce travail distingue les outils utilisés pour préparer le travail de groupe de ceux qui sont constamment utilisés par le groupe lors de son activité de coopération. D'après les définitions et exemples donnés par ce travail, on peut dire que les technologies (ou outils) de collaboration sont ceux qui sont utilisés en ligne pour la coordination de l'activité.

[Roschelle, 1992] considère que de nombreux exemples de logiciel de groupe sont de bonnes technologies personnelles (ou individuelles) qui fonctionnent comme une fondation pour le travail collaboratif. Par exemple un outil de planification de sessions peut faciliter l'organisation de travail collaboratif. Mais un tel logiciel ne constitue pas une technologie collaborative, car il ne participe pas dans la transformation collective de l'expérience liée au problème traité par le groupe. C'est le cas, par exemple pour nous, pour l'outil de planification des session RMS [2] [14Ci]. A l'opposé de ces bonnes technologies personnelles, les bonnes technologies collaboratives fonctionnent en devenant une partie hautement visible de l'expérience partagée par le groupe. Par exemple, un outil de partage de documents est une technologie collaborative et peut supporter une activité de réflexion libre en groupe (" brainsorming") qui peut être réalisée par la coordination des idées en l'air via le texte partagé sur les différents écrans des membres du groupe. C'est le cas pour nous, pour les outils de partage de documents [30Ci], [3J], [11Ci] et pour les services de coordination des sessions [19Ci], [15Ci], [10Ci], [9Ci], [7Ci], [1O].

3.5 La coordination

3.5.1 Les théories de coordination

La coordination est étudiée, par Malone et Crowston, en tant que science multi disciplinaire. Ce point de vue adopte une définition très générale (ou abstraite) qui s'applique à l'organisation des activités en général et des activités coopérative en particulier, qui s'applique à l'informatique en général et aux systèmes informatiques de support à la coopération en particulier.

La coordination est aussi étudiée en tant que spécialité pour certaines activités.

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'identification des

2. Responsibility Management System

dépendances[3] (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.

Dans [Malone et Crwoston, 1994], Malone et Crowston identifient quatre grandes classes de relations de dépendances :

- Les relations de dépendances liées au "partage des ressources", ou à "l'affectation des tâches", avec comme exemple de règles de coordination : premier arrivé/premier servi, ordre de priorité, et d'autres règles spécifiques à certains domaines d'activité, parmi lesquelles : les règles et contraintes budgétaires, les décisions de gestion, et les offres de marché.

- Les relations de dépendances induites par les "relations producteur/consommateur". Cette classe se décompose en trois sous-classes dont la principale concerne "les contraintes pré-requises". Les autres classes sont le transfert et l'utilisabilité. Pour la classe principale, les mécanismes sous-jacents les règles de coordination sont : la notification, l'ordonnancement, et le cheminement et suivi (" tracking").

- Les relations de dépendances liées aux contraintes de simultanéité avec comme exemple de mécanismes de coordination, la planification et la synchronisation.

- Les relations de dépendances de type "dépendances entre les tâches et les sous-tâches", avec comme mécanismes de coordination, la sélection et la décomposition.

Les dépendances liées aux contraintes de partage des ressources et aux contraintes de simultanéité sont traitées dans les travaux relatifs à la gestion des sessions coopératives synchrones. **C'est le cas de nos contributions résumées dans ce mémoire.** Les dépendances de type tâche/sous-tâche sont essentiellement étudiées par les travaux liés au domaine de la gestion du Workflow.

La coordination dans le domaine du Groupware se base aussi sur ces concepts de partage et d'interdépendance. Dans leur article de synthèse sur le "Groupware", Ellis, Gibs et Rein concluent que l'efficacité de la communication et de la collaboration peut être améliorée si les activités du groupe sont coordonnées. D'après cette synthèse, la coordination permet d'éviter les actions conflictuelles ou répétitives qu'une équipe de collaborateurs (comme des programmeurs ou des rédacteurs) pourrait engager.

Pour définir la coordination, la notion d'interdépendance de Malone a été aussi reprise ou étendue pour traiter les interdépendances entre les participants par d'autres chercheurs des domaines du CSCW ou du CSCL. Dans le cas du CSCL, et selon l'étude récente de [Stenning et al., 2003], la coordination peut être vue comme le développement des pratiques de représentation qui ont lieu dans le discours et qui engagent les apprenants dans l'interaction sociale.

Différents travaux ont été aussi menés dans le domaine du CSCL [Bourdeau et Wasson, 1997], et du CSCW [Raposo et al., 2000] pour l'identification et la description des rôles et des interdépendances.

3. appelées aussi interdépendances.

Dans le domaine du CSCL, les auteurs de [Bourdeau et Wasson, 1997] expliquent la nécessité de traiter ce type de dépendances par le raisonnement qui stipule que les acteurs doivent obligatoirement partager les objectifs et les ressources pour achever les activités, et vise-versa.

Dans [Bourdeau et Wasson, 1997], les auteurs classent les interdépendances entre participants en deux familles : "l'interdépendance de collaboration" et "l'interdépendance de concurrence". Dans la première famille, ils identifient deux classes d'interdépendance : celles liées à "l'objectif partagé" et celles liées à "l'activité partagée". Les mécanismes de support identifiés pour le support des interdépendances sont : "la formation d'équipe" pour les deux classes d'interdépendances, et pour la deuxième classe uniquement, l'affectation des tâches et des rôles, la décomposition de l'effort, le partage d'information, la communication, la planification, la synchronisation. Pour la famille d'interdépendance de concurrence, les auteurs distinguent deux classes d'interdépendances : celles liées à "l'activité partagée" (planification, communication, et monitoring) et celles liées aux "ressources partagées" de Malone et Crowston. Ces travaux ont été à la base de l'environnement de coordination de [Wasson, 1999].

Dans [Spector, 2001], l'auteur rappelle les théories de coordination à la base des études dans le domaine du CSCL. Il identifie la théorie de Salomon (Salomon, 1992,1993) comme une bonne théorie de coordination qui se base sur les théories éducatives (”learning theories”) et sur l'observation. Cette théorie accorde une grande importance à la médiation et la considère comme un concept fondamental pour l'enseignement coopératif. Ces travaux introduisent aussi le principe de l'interdépendance authentique (” genuine interdependency”) caractérisé par :

– la nécessité de partager l'information, les explications, les conceptions et les conclusions ; une décomposition de l'effort où les rôles des membres de l'équipe se complémentent les uns les autres dans un effort conjoint et le produit final requiert ce groupage des différents rôles ; et

– le besoin d'une réflexion commune en termes explicites qui peuvent être examinés, changés et élaborés par les partenaires.

Le travail de [Schmidt, 2002] relatif à la gestion du "workflow" définit la coordination comme le développement des technologies numériques pour aider des agents coopérants à coordonner et à intégrer leur travail. Dans ce domaine et selon la définition de [Schmid et Simone, 1996], le travail coopératif est constitué de l'interdépendance de plusieurs acteurs qui, dans leur activités individuelles, en changeant l'état de leurs champs de travail individuels, changent aussi l'état du champ de travail des autres acteurs et qui ainsi interagissent à travers le changement d'état d'un champ de travail partagé.

Le cas général de la coordination dans les groupwares a été traité dans nos travaux sur la coordination où des rôles spécifiques à la coordination sont spécifiés et où les interdépendances entre ces rôles sont identifiées [14Ci], [1O], [9Ci].

3.5.2 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 3.1).

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 3.1).

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.

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.

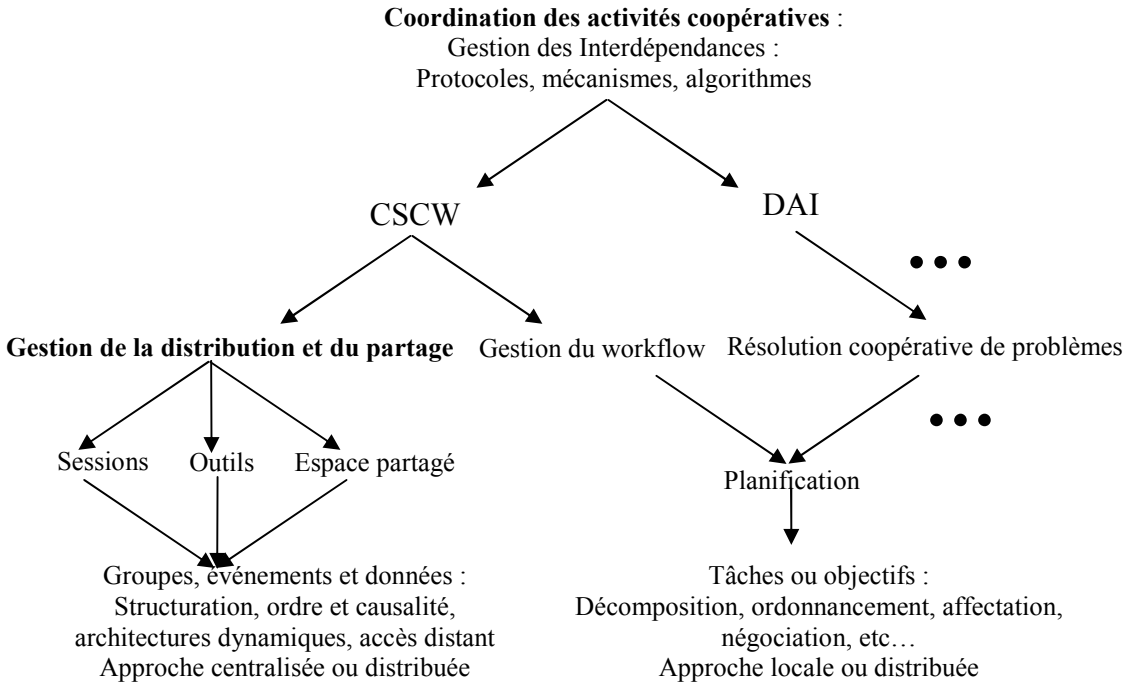


FIG. 3.1 – *Les différents domaines de recherche sur la coordination*

3.6 Synthèse

En synthèse des différentes définitions, nous considérons que :

– 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.

3.7 Conclusion

La coopération distribuée assistée par ordinateur implique différentes personnes géographiquement réparties sur différents sites de coopération. L’aide aux utilisateurs se base d’une part sur l’utilisation d’un espace de travail partagé sujet à des modifications concurrentes pouvant concerner un ou plusieurs objets de coopération, et conduites depuis différents points d’accès par des acteurs humains (participants) ou artificiels (composants ou applications). D’un autre côté, l’aide

se base sur un ensemble de supports logiciels pour l'échange d'information (données, ou actions à exécuter à distance) sujets à des interactions fréquentes dans un environnement de communication asynchrone, pour des groupes à structures et à attributs qui varient, et pour des participants dont les rôles et les privilèges différent d'un site à l'autre et changent au cours du temps.

L'incohérence des points de vues de l'espace partagé par la communication asynchrone et l'incohérence de son contenu par les actualisations concurrentes pourraient amener à un dysfonctionnement de l'ensemble du système de coopération. Cela entraînerait une mauvaise transformation de l'information en connaissance et par conséquent une mauvaise décision de groupe sur la base de cette connaissance.

La cohérence des vues et du contenu de l'espace de coopération est une exigence centrale pour le succès de la coopération.

La gestion des dépendances dans le "partage de ressources" et dans les "relations de type producteur/consommateur" sont deux "processus de coordination" [4] qui peuvent être développés, implantés et appliqués à la gestion de la cohérence de la coopération.

La *gestion de la concurrence* est un mécanisme de mise en œuvre pour le processus de coordination qui gère les dépendances lors de partage de ressources. Le contrôle d'accès est une fonction importante qui implante ce mécanisme. Favoriser le partage et la simultanéité est une exigence forte pour le succès du processus de coordination du partage dans les activités de coopération. Elle doit être respectée par le contrôle d'accès. Une solution appropriée pour le cas de l'assistance à la coopération doit se baser sur un contrôle à granularité fine et conscient de la structure de l'espace partagé, de l'architecture des applications qui sont utilisées pour le modifier, de la structure du groupe qui relie les participant, et des rôles et fonctions de ces participants.

La gestion de la causalité (réelle [5] ou supposée [6]) est un mécanisme de mise en œuvre pour le processus de coordination qui préserve les dépendances entre producteurs et consommateurs. Le bon fonctionnement de l'ensemble du système est conditionné par une cohérence dans l'ordre de diffusion des données, dans l'ordre d'exécution des actions, et une cohérence entre ces deux. Le contrôle *de l'ordre d'exécution des "actions"* (locales à un site ou distribuées) et le contrôle de *l'ordre de livraison des "données"* (informations, événements) sont des fonctions qui implantent ce mécanisme. Favoriser la simultanéité des actions et la diffusion des données est une exigence forte pour le succès du processus de coordination des dépendances entre producteur et consommateurs dans les activités de coopération. Le contrôle de l'ordre doit les respecter tout en tenant compte des contraintes d'hétérogénéité et d'interactivité des acteurs (producteurs et consommateur humain ou artificiel aux différents niveaux du système de support : communication, middleware, application) et de leur spécificité (concurrence interne possible ou non) et d'asynchronisme de la communication.

Les fonctions de coordination peuvent être implantées au niveau des interfaces, des architectures ou des protocoles, et ce tant pour les applications, pour le middleware que pour les services de communication ou de coopération.

Pour les interfaces, le contrôle peut être implanté au niveau des API par un dialogue à distance avec une application ou un service. Ce fut le cas pour configuer et contrôler l'application de simulation HLA et le service de conférence multi-point MCU durant le projet DSE. Le contrôle

4. selon la terminologie et la décomposition de la théorie de coordination de Malone.
5. appelée prédéfinie dans la suite.
6. appelée potentielle dans la suite.

peut aussi s'implanter au niveau des interfaces graphiques des applications (éditeurs partagés) ou des services de coopération (gestionnaires de session) par une personnalisation et adaptation selon l'état de l'activité assistée, ou selon les rôles et les privilèges des participants. Ceci est le cas pour les services de préparation et de gestion de session que nous avons développés également dans le cadre du projet DSE.

Pour les protocoles, le contrôle peut être implanté au niveau communication ou au niveau coopération. Dans le premier cas, il s'agit de préserver la causalité potentielle entre les événements diffusés dans le groupe de coopération. Dans le second cas, le contrôle au niveau coopération agit sur l'ordre des actions exécutées par chaque composant en tenant compte des actions déjà exécutées dans le reste du système de coopération.

La programmation orientée aspect peut constituer une solution pour implanter la coordination de façon générique au niveau protocole et interfaces. AspectJava permet de la mettre en œuvre pour les logiciels programmés en Java. L'interception des requêtes peut constituer une solution pour implanter les fonctions de contrôle au niveau middleware. Les intercepteurs du broker Orbacus permettent de la mettre en œuvre pour le middleware CORBA.[7]

L'implantation d'un processus de coordination au niveau architecture permet de gérer les dépendances producteur/consommateur de façon générique par une adaptation dynamique. Cette adaptation peut être guidée par les changements imposés pour le partage des ressources.

L'adaptation passe par la mise en œuvre de mécanismes de configuration dynamique. Celle-ci peut être réalisée par une distribution, une configuration et un déploiement[8] dynamiques des composants. L'adaptation peut passer aussi par une configuration dynamique des canaux de communication et la modification dynamique de leurs caractéristiques.[7]

L'adaptabilité basée sur l'évolution des structures, ou l'adaptabilité des structures, est une technique qui permet la mise en œuvre de la coordination. Elle concerne l'architecture des sites de coopération et la structure du groupe de coopération dans le cadre du domaine d'application qui nous intéresse. La reflexivité peut constituer une solution pour l'adaptabilité au niveau middleware.[7]

Trois catégories d'implantation peuvent être distinguées pour la coordination : la coordination centralisée, la coordination distribuée et la coordination hybride. Ce sont respectivement les catégories de nos solutions pour la gestion des architectures dynamiques, la gestion de la causalité au niveau communication, et la gestion de la causalité au niveau coopération.

La coordination guidée par un modèle formel ou semi-formel peut constituer un approche adaptée pour traiter le problèmes à différents niveaux fonctionnels de façon cohérente.

D'autres exigences de coopération, d'autre processus de coordination, d'autres solutions et approches de mise en œuvre peuvent être considérés. Nous nous sommes focalisés dans cette conclusion sur les caractéristiques de nos contributions dans ce domaine, et sur leurs extensions et alternatives potentielles.

7. Ce sont des extensions ou alternatives possibles pour les solutions présentées dans ce manuscrit.
8. Le déploiement est la mise en œuvre d'une distribution.

4

Contributions dans le domaine des systèmes distribués coopératifs

Ce chapitre présente les résultats des activités de recherche que nous avons menées dans le domaine des systèmes distribués coopératifs.

4.1 Introduction

Dans la principale partie de ce chapitre, nous nous centrons sur la coordination dans les logiciels distribués conçus pour le support des activités de coopération. Nous nous intéressons à l'identification et à la classification des interdépendances concernant : (1) des fonctions de gestion de sessions pour l'utilisateur final des logiciels de support à la coopération, (2) des fonctions de gestion des architectures pour le concepteur de ces logiciels et (3) des fonctions de gestion de l'espace partagé pour l'un ou l'autre.

Nous classons, pour cela, les interdépendances en deux catégories : **interdépendances événementielles** (ou comportementales) et **interdépendances structurelles**.

- les **interdépendances événementielles** peuvent appartenir à l'une des deux catégories suivantes :

  - La première catégorie est celle des interdépendances définies par des relations de "causalité prédéfinie" induites par la distribution des rôles dans le groupe des participants (par exemple l'admission du président de session doit précéder l'admission de tous les participants ordinaires) et par la distribution des instances de composants sur les sites de coopération (par exemple l'ouverture de la session doit être précédée par l'envoi des invitations aux participants.). Il s'agit par exemple de règles pouvant permettre d'accepter ou non les entrées après l'ouverture d'une session (admission des retardataires). Il s'agit aussi de règles pouvant permettre d'accepter ou non les utilisateurs non invités (session ouverte) ou permettant de limiter la portée de la messagerie instantanée (de l'extérieur vers le groupe et réciproquement). Nous parlons dans ce cas de **coordination des événements de coopération**.

  - La deuxième catégorie est celle des interdépendances définies par des relations de "causalité potentielle" induites par les contraintes du niveau communication et permettant d'éviter une éventuelles inconsistance au niveau coopération. Il s'agit par exemple de la réception des réponses avant les questions lors d'interactions asynchrones (coopération basée sur la messagerie électronique), ou de la réception d'un événement pour la modification d'un objet non encore créé ou d'une fenêtre graphique non encore affichée lors des interactions synchrones (coopération basée sur le partage d'applications). Nous parlons dans ces cas de **coordination des événements de communication**.

- les **interdépendances structurelles** peuvent appartenir à l'une des deux catégories suivantes :

  - 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).

Ce chapitre est organisé de la façon suivante :

- Les deux premières sections (§ 4.2 et §4.3) regroupent nos contributions et réalisations spé-

cifiques à la coordination et les présentent selon la classification présentée dans cette introduction.

- La section 4.2 regroupe nos contributions classées sous la catégorie de coordination guidée par les événements. La première partie concerne la coordination des événements de coopération. Les contributions de cette partie concernent la gestion des sessions de coopération impliquant plusieurs utilisateurs et plusieurs outils de coopération. La deuxième partie concerne la coordination des événements de communication. Les contributions de cette partie concernent la gestion de la diffusion causalement ordonnée.
- La section 4.3 regroupe nos contributions classées sous la catégorie de coordination guidée par la structure. La section 4.3.1 présente notre approche de coordination guidée par la structure. La section 4.3.2 présente son application à la gestion des sites de coopération (architectures dynamiques). La section 4.3.3 présente son application à la gestion de l'espace de coopération (partage des informations).

– La troisième section (§ 4.4) présente les autres contributions et réalisations.

4.2 Coordination guidée par les événements

4.2.1 Coordination des événements de coopération pour la gestion des sessions multi-outils multi-utilisateurs

4.2.1.1 La problématique générale de gestion des sessions de coopération

La gestion des sessions de coopération est une fonction de coordination commune à différents types d'activités de coopération. De nombreux problèmes sont posés, et différentes approches peuvent être envisagées pour les résoudre. Différentes équipes de recherche dans le monde se sont intéressées récemment à cette problématique. Différents travaux ont été conduits dans ce domaine. Les problèmes adressés varient de la coordination des interactions [Texier, 2000, Dommel et Aceves, 1997, Dommel et Aceves., 1999] à la gestion des structures de groupe pour configurer et contrôler les communications [Wilde, 1997, Dommel et Garcia-Luna-Aceves, 2000]. On peut classer les approches en deux grandes familles :

- celles qui considèrent que les sessions de coopération sont toujours implicites et regroupent tous les participants qui accèdent au même espace. Ceci est le cas pour les travaux qui traitent la coopération improvisée et non structurée [Beca et al., 1997, Chanbert et al., 1998, Texier et Plouzeau, 1999]

- celles qui considèrent que les sessions sont explicites et doivent être préparées avant le démarrage de l'activité de coopération. Ceci est le cas, par exemple, de [Hall et al., 1996] et [Schukmann et al., 1996] qui s'intéressent respectivement à la gestion des communications et des interactions dans les systèmes de coopération de groupe. C'est le cas aussi de [Costantini et Toinard, 2001] qui s'intéresse à la gestion des autorisations pour l'apprentissage coopératif.

La première famille peut s'adapter aux différentes tailles de groupes et convient particulièrement pour les activités de coopération spontanée. Nous l'avons adoptée pour la gestion des sessions au niveau des différentes versions d'outils d'édition coopérative que nous avons développées. La deuxième famille est nécessaire pour les activités de coopérations planifiées. Ceci est le cas pour les activités de coopération considérées par exemple dans le cadre du projet européen DSE. Pour éviter les inconvénients et les limites de ces deux approches, nous avons développé, en plus, une couche de définition orientée rôle qui peut être considérée comme un compromis entre les approches totalement implicites où tout se décide pendant l'exécution et les approches totalement explicites où tout se décide pendant la définition.

Dans chacune de ces familles on peut distinguer les approches qui traitent un seul outil à la fois et celles qui traitent différents outils simultanément. La plupart des travaux cités ci-dessus se situent dans la première catégorie. Nos travaux se situent dans la 2ème catégorie de façon similaire aux approches de gestion de session des projet Habanero et ISAAC [Chanbert et al., 1998, Isaac,] et Tango [Tango,]. Nous nous distinguons des ces approches ad-hoc par notre méthodologie basée sur des modèles formellement élaborés.

4.2.1.2 Caractérisation des interdépendances événementielles dans les sessions de coopération

Une session de coopération multi-outils multi-utilisateurs [1O] est composée (cf. figure 4.1)

1. d'un **groupe d'utilisateurs** (dit aussi groupe de coopération) qui possèdent chacun, un ou plusieurs rôles (chairman, secretary, participant),

2. d'un ensemble d'**applications distribuées coopératives** (considérées, par les utilisateurs, comme des outils de coopération) distribuées (réellement ou virtuellement par duplication) sur les différents **sites de coopération**.

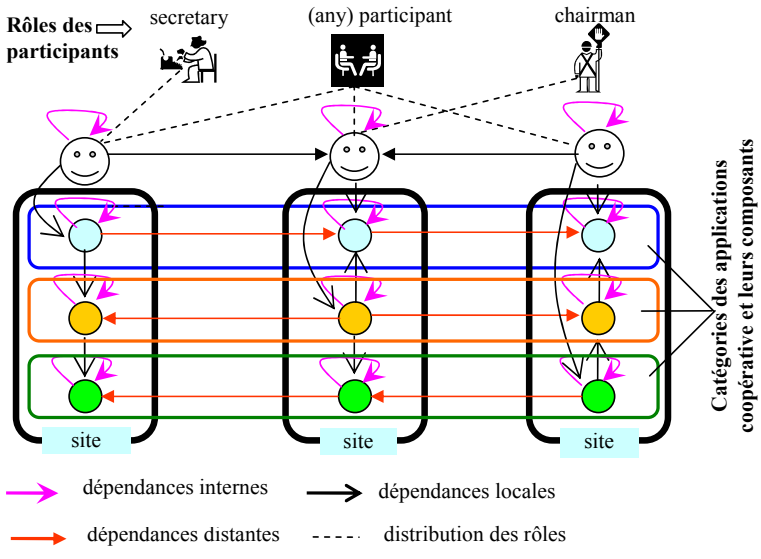


FIG. 4.1 – *Différents scénarios d'interdépendances pour les sessions de coopérations.*

Dans le cadre du projet DSE, nous avons travaillé sur l'identification des interdépendances événementielles pouvant impliquer :

- plusieurs composants entre eux. Les composants peuvent être d'une même ou de différentes applications, sur le même site ou sur différents sites de coopération,

- plusieurs utilisateurs entre eux. Ils peuvent avoir le même rôle ou différents rôles,

- ou plusieurs composants et plusieurs utilisateurs d'une session de coopération multi-utilisateurs et multi-applications.

Une première classification a été développée dans [10Ci]. Trois relations de base ont été définies : "Déclenchement" ("Enabling"), "Inhibition" ("Inhibiting"), et "Précédence" ("Preceding"). Une formalisation par la logique du premier ordre a été développée dans [7Ci] sur la base d'une architecture qui fait abstraction de la distribution et de la localisation des composants et des acteurs. Une seconde formalisation a été développée dans [1O]. Elle tient compte de la distribution des composants sur les sites de coopérations, et de la distribution des rôles et des privilèges d'accès à l'espace de coopération sur les participants (fig 4.1). De nouvelles relations de base ont été définies ensuite par raffinement des trois premières ("LateEnabling", "ImmediatePrecedence",etc...).

Trois classes de dépendances entre événements de coopération sont distinguées :

- Les dépendances **internes**, spécifiant des contraintes sur les actions des participants, ou des contraintes sur les actions d'une catégorie de composant, d'une et une seule application, sur un ou plusieurs sites;

- Les dépendances **locales**, spécifiant des contraintes internes à un site, et impliquant deux composants de deux applications différentes;

- Les dépendances **distantes**, spécifiant des contraintes impliquant, deux participants, deux composants d'une même application sur des sites distants.

Différentes classes d'événements de coopération sont distinguées :

1. les événements concernant l'état de la session (création, démarrage, ouverture, cloture, etc...)

2. les événements concernant les actions des participants (arrivée/départ lors de la formation des groupes)

3. les événements concernant les actions des outils de coopération (démarrage, arrêt, autre action de contrôle) et

4. les événements d'information pour la mémoire de groupe [1] (messages ou avertissements).

1. "Awareness"

4.2.1.3 Description des interdépendances

Les notations utilisées dans cette section ont été définies pour décrire les interdépendances, d'abord de façon abstraite dans [14Ci] puis de façon développée dans [1O]. Nous avons amélioré la description et la classification des interdépendances. L'ensemble des relations de base (précédence, déclenchement, inhibition) initialement définies dans [10Ci] a été étendu, dans [10Ci], par l'introduction des notions de "déclenchement tardif", de "précédence immédiate" de "déclenchement immédiat" et de "non concurrence".

Une session de coopération est définie comme une activité qui implique des acteurs distribués (éléments d'un ensemble fini P). Durant la session, les acteurs exécutent des actions de **coordination** (éléments d'un ensemble fini A) et génèrent des événements. Chaque événement est l'occurrence d'une action donnée exécutée par un acteur donné. Le couple (acteur, action) désigne le nom d'un événement, et l'ensemble des noms d'événement est $\Sigma = P \times A$. Plusieurs événements peuvent être des occurrences du même nom d'événement pendant une session.

Définition 4.1 (session de coopération) *Une* **session de coopération** p *sur l'ensemble des noms d'événements* Σ *est un triplet* $p = (E, \leq, l)$ *où :*

E *est un ensemble fini dont les éléments sont appelés événements ,*

$\leq$ *est un relation binaire réflexive, anti-symétrique et transitive sur* E*, et*

$l : E \to \Sigma$ *une fonction d'étiquetage des événements. Elle définit un événement* $e \in E$ *comme une occurrence du nom d'événement* $l(e) \in \Sigma$*.*

Conventions des notations logiques

Nous notons $FO(\leq, \Sigma)$ l'ensemble des formules de la logique du premier ordre construites sur la relation $\leq$ et l'alphabet Σ. Ces formules sont définies par la grammaire suivante :

$$\varphi ::= P_\alpha(x) \mid x \leq y \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\varphi$$

Nous écrivons $\varphi(x_1, \ldots, x_n)$ quand $(x_1, \ldots, x_n)$ sont des variables libres qui peuvent apparaître dans une formule $\varphi \in FO(\leq, \Sigma)$. Ces variables libres doivent être instantiatiées par un événement de session quand la formule est appliquée à une session. Soit $p = (E, \leq, l)$ une session de coopération et $e_1, \ldots, e_n \in E$, nous notons $(p, e_1, \ldots, e_n) \models \varphi(x_1, \ldots, x_n)$, pour dire que φ est satisfait par p quand x_i a la valeur e_i pour $i = 1, \ldots, n$. La relation de satisfaction est définie inductivement comme suit :

$(p, e) \models P_\alpha(x)$ ssi $l(e) = \alpha$

$(p, e_1, e_2) \models x \leq y$ ssi $e_1 \leq e_2$

$(p, e_1, ..., e_n, e_{n+1}, ..., e_{n+p})) \models \varphi(x_1, ..., x_n) \bigwedge \psi(y_1, ..., y_p)$ ssi
$(p, e_1, ..., e_n) \models \varphi(x_1, ..., x_n)$ et $(p, e_{n+1}, ..., e_{n+p}) \models \psi(y_1, ..., y_n)$

$(p, e_1, ..., e_n) \models \neg\varphi(x_1, ..., x_n)$ ssi $\neg((p, e_1, ..., e_n) \models \varphi(x_1, ..., x_n))$

$(p, e_1, ..., e_n) \models \exists x.\varphi(x, x_1, ..., x_n)$ ssi pour certains $e \in E$, $(p, e, e_1, ..., e_n) \models \varphi(x, x_1, ..., x_n)$

Les connecteurs ($\vee$ et $\forall x$)) et les différentes implications ($\Longrightarrow$, etc...) sont dérivés d'une manière standard.

Le prédicat de précédence immédiate ("$\rightarrow$") est un opérateur dérivé défini de la manière suivante :

$$x \rightarrow y \quad \stackrel{def}{\equiv} \quad (x < y \ \wedge \ \forall z.(x < z \leq y \implies z = y))$$

En particulier, si ψ est une phrase (c.-à-d.. ψ ne contient pas des variables libres) alors elle décrit une propriété de p, ce que nous notons $p \models \psi$.

Caractérisation des trois classes de dépendance : interne, locale et distante
Les trois classes des dépendances sont associées aux trois classes de prédicats définies ci-après.

– **les prédicats de dépendance interne** sont des prédicats définissant des "dépendances intra-acteur", également appelées les propriétés de **internes**. Ce sont des prédicats ayant toujours le même acteur dans leur ensemble de noms d'événement, et elles décrivent la cohérence interne du comportement d'un acteur. La plus part du temps ils sont définis pour tous les acteurs de la même catégorie. Formellement, une formule φ est relative aux dépendances internes ssi :

$$\forall (n, a), (m, b) \in Alph(\varphi) : n = m$$

Où $Alph(\varphi)$, l'alphabet dénotant l'ensemble de noms d'événement associé à la formule φ, est défini inductivement par :

$$\begin{aligned} Alph(P_\alpha(x)) &= \{\alpha\} \\ Alph(x \leq y) &= \emptyset \\ Alph(\varphi \wedge \varphi) &= Alph(\varphi) \cup Alph(\varphi) \\ Alph(\neg\varphi) = Alph(\exists x.\varphi) &= Alph(\varphi) \end{aligned}$$

Ces formules permettent de décrire des propriétés de type : les acteurs de la catégorie "Session Manager" ne peuvent pas exécuter l'action "détruire un session" avant d'avoir exécuté l'action "ouvrir une session". Il s'agit de règle de cohérence dans le protocole interne à un acteur.

– **les prédicats de dépendance locale** sont les prédicats définissant des "dépendances intra-site", également appelées les prédicats **locaux**. Ce sont des prédicats dont les acteurs associés sont localisés sur le même site. Ils décrivent la cohérence mutuelle des comportements des acteurs localisés sur ce site. Elles sont généralement communes à tous les sites d'une session de coopération. Formellement, une formule φ est relative aux dépendances locales ssi :

$$\forall (n, a), (m, b) \in Alph(\varphi) : Site(n) = Site(m)$$

Ces classes de prédicats permettent de décrire des propriétés de type : les acteurs de la catégorie "Floor Control" ne peuvent pas exécuter l'action "start FC" avant que l'acteur de la catégorie "AudioVideo Conferencing" n'ait exécuté l'action "start AV".

– **les prédicats de dépendance distante** définissent des dépendances intra-catégorie, également appelées les prédicats **distants**. Ce sont des prédicats dont les acteurs appartiennent à la même catégorie et sont localisés sur différents sites. Ils décrivent la cohérence mutuelle des comportements des acteurs distribués de cette catégorie. Formellement, une formule est relative aux dépendances distantes ssi :

$$\forall (n,a),(m,b) \in Alph(\varphi) : Cat(n) = Cat(m)$$

Cette classe de prédicats permet de décrire des propriétés de type : un acteur de la catégorie "Users" ne peut pas exécuter l'action "accept" avant qu'un acteur distant et de la même catégorie n'ait exécuté l'action "grant".

Spécification des dépendances de base

Les relations qui expriment les dépendances impliquent les relations internes, locales, ou distantes reflétant des contraintes intra-acteur, inta-site ou intra-catégorie.

Définition 4.2 (Précédence) *Pour tout couple d'actions $\alpha, \beta \in \Sigma$, la formule de précédence, notée $Pred(\alpha,\beta)$, est définie par :*

$$Pred(\alpha,\beta) \stackrel{def}{\equiv} \forall x\,.\,P_\beta(x) \implies (\exists y.\, y < x \wedge P_\alpha(y))$$

Cette formule signifie que chaque fois qu'une action β est exécutée, l'action α doit avoir eu lieu avant, au moins une fois.

Définition 4.3 (Déclenchement tardif) *Pour tout couple d'actions $\alpha, \beta \in \Sigma$, la formule de "déclenchement tardif", notée $LEnable(\alpha,\beta)$, est définie par :*

$$LEnable(\alpha,\beta) \stackrel{def}{\equiv} \forall x\,.\,P_\alpha(x) \implies (\exists y.\, x < y \wedge P_\beta(y))$$

Cette formule signifie que chaque occurrence de l'action α doit être suivie par une occurrence de l'action β.

Définition 4.4 (Inhibition) *Pour tout couple d'actions $\alpha, \beta \in \Sigma$, la formule d' "inhibition", notée $Inhib(\alpha,\beta)$, est définie par :*

$$Inhib(\alpha,\beta) \stackrel{def}{\equiv} \forall x\,.\,P_\alpha(x) \implies (\forall y.\, x < y \Rightarrow \neg P_\beta(y))$$

Cette formule signifie qu'une fois l'action α est exécutée, il n'est plus autorisé d'exécuter l'action β.

Définition 4.5 (Précédence Immédiate) *Pour tout couple d'actions $\alpha, \beta \in \Sigma$, la formule de "précédence immédiate", notée $ImPred(\alpha,\beta)$, est définie par :*

$$ImPred(\alpha,\beta) \stackrel{def}{\equiv} \forall x\,.\,P_\beta(x) \implies (\exists y.\, y \rightarrow x \wedge P_\alpha(y))$$

Cette formule signifie que toute occurrence de l'action β est immédiatement précédée par une occurrence de l'action α.

Définition 4.6 (Déclenchement Immédiat) *Pour tout couple d'actions $\alpha, \beta \in \Sigma$, la formule de "déclenchement immédiat", notée $ImEnable(\alpha, \beta)$, est définie par :*

$$ImEnable(\alpha, \beta) \stackrel{def}{\equiv} \forall x.\, P_\alpha(x) \implies (\exists y.\, x \rightarrow y \ \wedge \ P_\beta(y))$$

Cette formule signifie que toute occurrence de l'action α est immédiatement suivie par une occurrence de l'action β.

Définition 4.7 (Non concurrence) *Pour tout couple d'actions $\alpha, \beta \in \Sigma$, la formule de "non concurrence", notée $InSequence(\alpha, \beta)$, est définie par :*

$$InSequence(\alpha, \beta) \stackrel{def}{\equiv} \forall x.\, \forall y.\, P_\alpha(x) \ \wedge \ P_\beta(y) \implies (\, x < y \ \vee \ y < x \,)$$

Cette formule signifie les actions α et β ne peuvent pas être exécutées en parallèle.

L'absence de concurrence interne à un acteur est vraie pour les sessions coopératives. Cette propriété s'exprime par la formule suivante :

$$\bigwedge_{n \in P,\, a,b \in A} InSequence((n,a),(n,b))$$

Les règles de coordination des sessions de coopération

En utilisant les différentes formules de base présentées ci-dessus, nous avons modélisé les règles de coordination pour la gestion de session dans le contexte du projet DSE. L'ensemble des applications coopératives généralement utilisées se compose de 8 applications représentées par les 8 catégories : *SM : Session Manager, PM : presence Manager, FC : Floor Controller, IM : Instant messaging, LF : Life cycle Manager, AV : Audio Video conferencing, AS : Application Sharing, SE : Shared editing*. $C = \{Part, SM, PM, FC, IM, LF, AV, AS, SE\}$. Le chairman est un participant spécial qui est autorisé à exécuter plus d'actions de gestion que les participants ordinaires. Le rôle chairman peut être joué par différents participants possédant l'attribut "chair". Mais à tout moment, un seul participant est chairman.

La figure 4.2 illustre un exemple de configuration et de distribution de rôles pour une session de coopération.

Règles de gestion des dépendances internes

Ces règles expriment le contrôle correct des événements exécutés par un acteur quelconque, et ce pour différentes catégories y compris la catégorie "participant". Elles permettent d'assurer la cohérence d'état interne.

Propriétés du "Session Manager" La règle de la coordination représentée dans l'équation 4.1 montre les lois de transitions d'état pour une session de coopération. Une session commence son cycle de vie après l'exécution de l'événement "initialize" qui signifie session initialisée. Après sa création, une session peut passer à l'un des états : "announce"

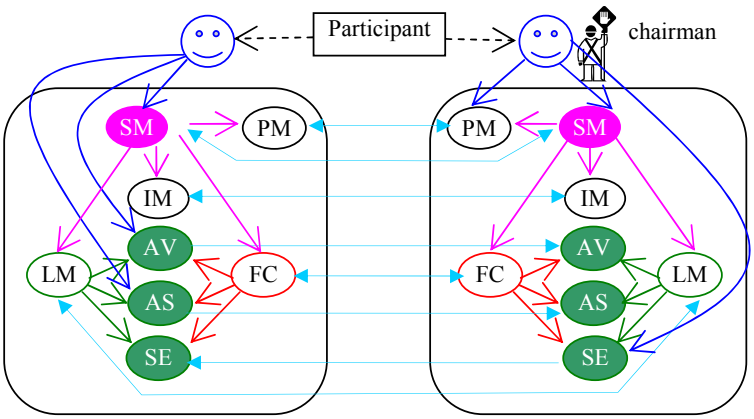


FIG. 4.2 – *Description d'une configuration donnée et d'une distribution de rôles pour une session de coopération.*

ou "delete", selon l'événement ayant eu lieu. L'état "announce" signifie que les participants ont été invités à la session. Tandis que l'état "delete" signifie l'arrêt de la session. En outre, une session annoncée est soit ouverte, auquel cas elle passe à l'état "open", soit annulée, auquel cas elle passe à l'état "delete".

L'état "open" marque le début de l'activité de coopération qui finit en atteignant l'état "close". L'activité de coopération peut être annulée depuis l'état "open" auquel cas elle passe directement à l'état "delete" sans passer par "close". La session fermée est terminée correctement par le passage de l'état "close" à l'état "delete".

$$\delta_1 = \bigwedge_{Cat(t)=SM} \begin{pmatrix} Pred((t,create),(t,announce)) & \wedge \\ Pred((t,announce),(t,open)) & \wedge \\ Pred((t,open),(t,close)) & \wedge \\ Pred((t,close),(t,delete)) & \end{pmatrix} \quad (4.1)$$

Propriétés d'un "Participant" L'équation4.2 définit la règle de coordination qui indique qu'un participant peut se déconnecter si et seulement s'il était déjà connecté.

$$\delta_2 = \bigwedge_{Cat(p)=Part} Pred((p,join),(p,leave)) \quad (4.2)$$

Propriétés d'un composant de coopération : L'équation 4.3 définit une propriété relative à la gestion des outils. Elle consiste à dire qu'un composant de coopération (c-à-d un acteur appartenant à une des trois catégories : AV, AS ou SE) ne peut être arrêté que s'il a été déjà démarré. En conséquence, le "session manager" doit gérer la liste des composants démarrés.

$$\delta_3 = \bigwedge_{Cat(t) \in \{AV,AS,SE\}} Pred((t,start),(t,stop)) \quad (4.3)$$

Règles de gestion des dépendances locales

Ces règles expriment le contrôle correcte des événements exécutés par les acteurs d'un site donné. Elles permettent d'assurer la cohérence de la configuration d'un site. La règle de l'équation 4.4 indique que les composants doivent être arrêtés automatiquement après l'arrêt de la session.

$$\delta_4 = \bigwedge_{Cat(t_1)=SM, Cat(t_2)=LF} ImEnable((t_1, delete), (t_2, stop)) \quad (4.4)$$

La règle décrite par l'équation 4.5 indique la portée du transfert du rôle "chairman". Elle indique que le participant qui peut accepter ce rôle doit appartenir au groupe et doit être connecté à la session.

$$\delta_5 = \bigwedge_{Cat(t)=SM, Cat(p)=Part} Pred((t, join), (p, accept)) \quad (4.5)$$

Règle d'appartenance au groupe (Group membership)
Cet ensemble de règles définit, en fonction de l'état de la session, les actions correctes pour aboutir à la formation, l'évolution, et la dissolution du groupe de coopération.

Règles de formation du groupe

Admission sur invitation: Nous avons considéré que la connexion d'un participant à une session est possible si et seulement s'il était invité à cette session. Cette règle est illustrée par l'équation 4.6.

$$\delta_6 = \bigwedge_{Cat(t_1)=SM, Cat(t_2)=PM} Pred((t_1, invite), (t_2, join)) \quad (4.6)$$

Admission seulement avant terminaison: La règle illustrée par l'équation 4.7 indique que les participants ne peuvent pas se connecter à une session qui a été déclarée terminée. Cette règle signifie autrement, qu'on ne se connecte pas à une session qui n'existe plus.

$$\delta_7 = \bigwedge_{Cat(t_1)=SM, Cat(t_2)=PM} Inhib((t_1, delete), (t_2, join)) \quad (4.7)$$

Règles de dissolution du groupe

Gestion du départ des participants: La règle de l'équation 4.8 indique que les participants sont automatiquement déconnectés après l'arrêt de la session.

$$\delta_8 = \bigwedge_{Cat(t_1)=SM, Cat(t_2)=PM} ImEnable((t_1, delete), (t_2, leave)) \quad (4.8)$$

Portée des Communications à l'intérieur et à l'extérieur d'un groupe de coopération

Seuls les membres du groupe sont autorisés à communiquer La règle représentée par l'équation 4.9 indique que les participants doivent se connecter avant d'envoyer et de recevoir des informations. C'est une règle d'interdépendance locale.

$$\delta_9 = \bigwedge_{Cat(t_1)=PM, Cat(t_2)=IM} \begin{pmatrix} Pred((t_1, join), (t_2, send)) & \wedge \\ Pred((t_1, join), (t_2, receive)) & \end{pmatrix} \tag{4.9}$$

Seuls les membres connectés peuvent communiquer La règle de l'équation 4.10 indique que les participants ne peuvent ni envoyer ni recevoir des informations après leur déconnexion.

$$\delta_{10} = \bigwedge_{Cat(t_1)=PM, Cat(t_2)=IM} \begin{pmatrix} Inhib((t_1, leave), (t_2, send)) & \wedge \\ Inhib((t_1, leave), (t_2, receive)) & \end{pmatrix} \tag{4.10}$$

Règles de gestion des dépendances distantes

Un seul chairman à tout moment: L'équation 4.11 et l'équation 4.12 représentent les règles qui définissent le transfert du rôle chairman entre les participants. L'équation 4.11 utilise le prédicat précédemment défini InSequence, et s'assure que toutes les actions "concessions"(grant) et "acceptation" (accept) sont séquentiellement ordonnées (c.-à-d. elles ne peuvent pas être concurrentes).

$$\delta_{11} = \bigwedge_{Cat(p)=Cat(q)=Part} \left(\bigwedge_{a,b \in \{grant, accept\}} InSequence((p, a), (q, b)) \right) \tag{4.11}$$

L'équation 4.12 indique qu'un participant peut accepter le rôle chairman si, et seulement si, le chairman actuel lui accordait ce rôle. Cette propriété est assurée par les deux prédicats, "Accept" et "Grant", qui garantissent que deux actions grant consécutives de concession (resp. accept) sont nécessairement séparées par une action accept (resp. grant).

$$\delta_{12} = \bigwedge_{Cat(p)=Cat(q)=Part} (Accept(p, q) \wedge Grant(p, q)) \tag{4.12}$$

Avec :

$$Grant(p, q) = \forall x, y.(P_{(p,grant)}(x) \wedge P_{(q,grant)}(x) \wedge x \leq y)$$

$$\Longrightarrow (\exists z.(P_{(q,accept)}(z) \wedge x \leq z \leq y)$$

$$Accept(p, q) = \forall x, y.(P_{(p,accept)}(x) \wedge P_{(q,accept)}(x) \wedge x \leq y)$$

$$\Longrightarrow (\exists z.(P_{(p,grant)}(z) \wedge x \leq z \leq y)$$

4.2.1.4 Réalisations associées

Nos études sur la coordination des événements de coopération se sont concrétisées par la conception et la réalisation des services RMS et SMS pour la configuration et la gestion des sessions coopératives multi-applications. La conception et la mise en œuvre de ces services ont été effectuées dans le cadre des études menées durant le projet DSE. La figure 4.3 illustre l'intégration de l'outil SMS dans l'environnement de coopération DSE. Nous avons répondu aux exigences exprimées par les utilisateurs de l'environnement DSE et nous avons étendu nos travaux afin de proposer des modèles génériques et des architectures ouvertes qui seraient utilisables dans d'autres travaux de développement des systèmes distribués coopératifs. Le modèle de structuration des groupes de coopération ouverts est décrit dans [14Ci]. Le modèle UML qui représente la conception du service de gestion de sessions SMS est décrit dans [19Ci]. L'architecture orientée événements du service SMS qui contient les définitions des événements relatifs à la session, aux participants, à l'information et aux applications, est décrite dans [15Ci].

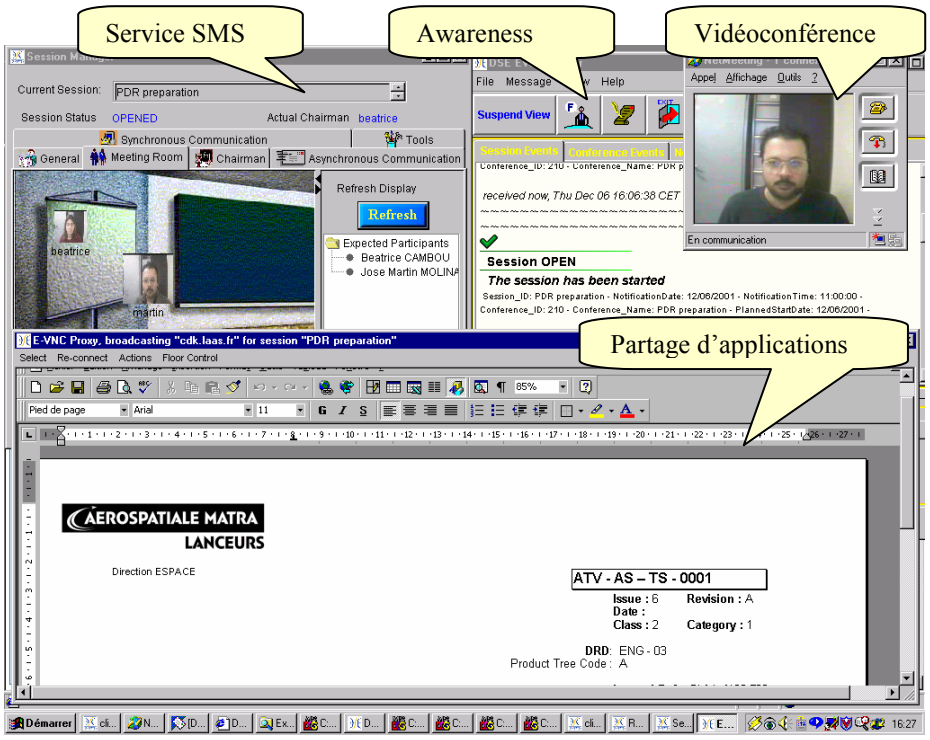


FIG. 4.3 – *Interface de l'environnement DSE intégrant l'outil SMS.*

Service de Préparation et de configuration des sessions multi-outils (RMS) La structuration du groupe coopératif permet aux acteurs de la session de posséder des connaissances sur eux-mêmes et sur les autres participants avec lesquels ils coopèrent. Elle introduit des règles d'action pour chaque participant à l'intérieur des groupes coopératifs. La structuration du groupe, en mettant en place les liaisons entre les divers membres du groupe, permet aussi de choisir les types de coordination entre participants et d'y adapter le fonctionnement des composants dans la structure logicielle. Suivant leur position dans le groupe, certains membres peuvent avoir un rôle priori-

taire : des responsabilités peuvent leur être affectées afin de favoriser l'avancée du travail, d'éviter ou d'arrêter les déviations, de supprimer d'éventuels blocages ou ralentissements inacceptables. Dans le cadre de ces travaux, nos résultats ont été :

– la définition d'un modèle, orienté rôle, de structuration des sessions de coopération. L'attribution des rôles permet de planifier les sessions de travail, d'éviter ou de résoudre d'éventuels conflits. Cette réflexion a été d'abord développée dans [7O] et ensuite exploitée dans le cadre du projet DSE.

– L'expérimentation de ce modèle et sa validation dans le cadre du scénario de coopération PDR (Preliminary Design Review) [19Ci] du projet DSE.

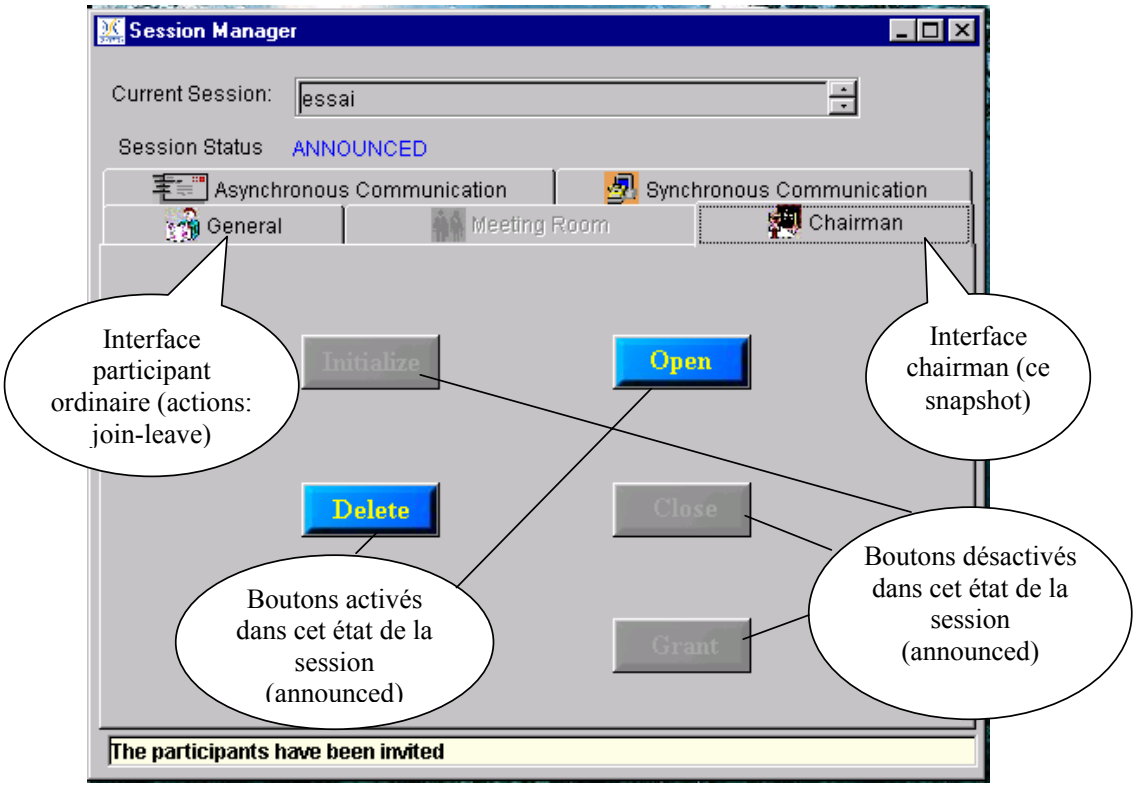


FIG. 4.4 – *Interface de l'environnement SMS.*

Service de Coordination des sessions de coopération (SMS) Ce travail a été conduit dans le cadre du projet DSE [19Ci], [4Rc] et s'inscrit dans la ligne de réflexion des groupes de travail sur les applications Internet (Application Area) : PRIM (Presence and Instant Messaging Protocol) [Presence et (prim),] et IMPP (Instant Messaging and Presence Protocol) de l'IETF [IETF,]. Le but de ce travail est de fournir un service pour la coordination des phases synchrones de coopération. Il s'agit en particulier de gérer la constitution du groupe de participants, de gérer la présence des membres de ce groupe et de soutenir l'animateur de session dans sa gestion de l'évolution de l'état de la session. Les événements échangés permettent de garantir la validité du contexte de la session lors de son ouverture et de sa fermeture ainsi que lors des changements relatifs à la

structure de groupe ou aux rôles des participants lors de son déroulement. Il s'agit en pratique de définir un scénario d'actions locales à exécuter par les participants pour permettre à l'animateur et au reste du groupe d'être informés sur l'état du site de chaque participant après réception des événements relatifs à la présence ou l'absence d'un participant. En fonction du rôle affecté au participant et en fonction de l'événement ayant eu lieu, on peut décider, par exemple, de démarrer certains outils, ou d'accepter telle requête d'admission sans risque de dysfonctionnement.

Dans le cadre de ces travaux, nos résultats ont été :

– la définition d'une architecture d'exécution orientée événements pour la coordination des sessions [15Ci]. Cette architecture possède l'avantage de permettre la gestion des sessions structurées et ouvertes au cours desquelles on n'exige pas toujours la présence de tous les participants pour le démarrage; de plus la connexion des participants est acceptée pendant les différentes étapes du déroulement de la session, le contexte de coopération étant maintenu à jour et communiqué aux nouveaux arrivants. Les informations représentées dans le contexte incluent : la présence des participants, l'état des sous-sessions applicatives, et l'état des espaces de coopération.

– La mise en œuvre de cette architecture par un environnement de gestion de présence et de messagerie instantanée et sa validation dans le cadre de deux scénarios de coopération qui nous ont été proposés dans le cadre du projet DSE [4Rc], [2Rc].

4.2.2 Coordination des événements de communication

Afin de garantir la cohérence des interactions entre les différents utilisateurs d'un outil de coopération, il est nécessaire de maintenir la causalité entre les différents événements qui sont échangés sur le canal de communication utilisé par cet outil. Si nous considérons l'exemple de la messagerie de groupe. La prise en compte de la causalité permet d'éviter le dépassement des questions par les réponses chez l'un des membres du groupe. Pour une cohérence globale à l'échelle de la session, il est aussi parfois nécessaire d'étendre cette cohérence à l'ensemble des canaux créés dans la session. En effet les utilisateurs peuvent interagir via plusieurs outils de coopération simultanément. Si nous considérons l'exemple d'une activité de rédaction coopérative, un commentaire échangé par la messagerie et concernant un document diffusé par un outil d'édition partagée ne doit pas dépasser l'arrivée et l'affichage du document en question chez les partenaires. La figure 4.5 décrit un scénario de coopération qui regroupe l'ensemble de ces situations. Le scénario considéré implique quatre utilisateurs : U1, ..., U4, et trois outils de coopération : un outil partage d'applications, un outil de messagerie instantanée et un outil d'édition et d'annotation coopératives. Chaque utilisateur peut être producteur ou consommateur pour un ou plusieurs outils de coopération. Chaque outil utilise son propre canal de diffusion. Les conflits liés à la causalité peuvent concerner les événements intra-outil diffusés sur le même canal de communication. C'est le cas pour l'outil de partage d'applications entre U1,U2 et U3. Les conflits peuvent également concerner la causalité inter-outils et sont diffusés sur des canaux différents. C'est le cas pour l'outil de partage d'applications et la messagerie instantanée impliquant U1,U2 et U3.

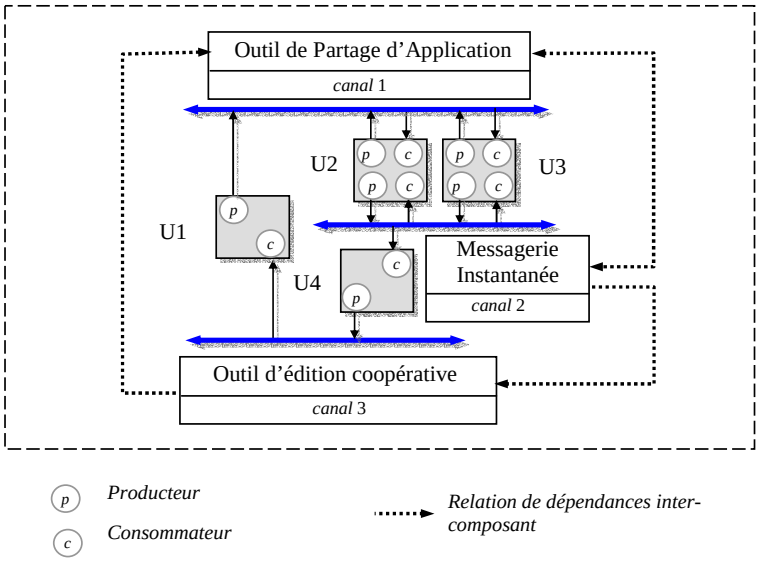


FIG. 4.5 – *Exemple de scénarios de coopération avec conflits liés à la causalité des communications multi-canaux multi-utilisateurs*

4.2.2.1 La livraison causale

La livraison causale est basée sur la relation de précédence causale de Lamport [Lamport, 1978]. Cette relation relie deux messages de la façon suivante :

Définition 4.8 (La relation de causalité) *La relation de causalité, notée* $\rightarrow$*, est définie par les 3 règles suivantes :*

1. $\langle i, a\rangle \rightarrow \langle j, b\rangle$ **si** $i = j \wedge a < b$
2. $\langle i, a\rangle \rightarrow \langle j, b\rangle$ **si** *$\langle i, a\rangle$ représente l'émission d'un message et si $\langle j, b\rangle$ représente la livraison de ce message.*
3. $\langle i, a\rangle \rightarrow \langle j, b\rangle$ **si** $\exists \langle k, c\rangle | (\langle i, a\rangle \rightarrow \langle k, c\rangle \wedge \langle k, c\rangle \rightarrow \langle j, b\rangle)$ *où i, j, et k sont des identificateurs de processus*[2]*, et a, b, et c sont les valeurs d'horloges locales de i, j, et k, respectivement.*

Dans le cas de la communication de groupe, la livraison causale peut-être étudiée dans deux contextes. Le premier est celui de la diffusion (braodcast) dans un groupe. Le second est celui des multi-groupes qui peuvent éventuellement se chevaucher. La livraison causale pour le premier cas est définie comme suit [Birman, 1993] :

Définition 4.9 (Livraison causale mono-groupe) *Si* $send(m) \rightarrow send(m')$*, alors* $\forall k \in c : delivery_k(m) \rightarrow delivery_k(m')$

2. Dans le cadre de nos applications, un processus est soit un outil de coopération soit un composant à l'intérieur d'un outil de coopération.

La livraison causale mono-groupe garantit que si la diffusion d'un message m précède causalement la diffusion d'un message m', dans un groupe c, alors la livraison de m précède causalement la livraison de m' pour tout participant p_k appartenant au groupe c.

Le cas de la livraison causale dans un contexte de multi-groupes est plus courant dans la communication de groupe. Elle se définit de la façon suivante :

Définition 4.10 (Livraison causale multi-groupes) *Si* $send_i(m, c) \rightarrow send_j(m', c')$, *alors* $\forall k \in c \cap c'$: $delivery_k(m) \rightarrow delivery_k(m')$

De façon générale, nous pouvons dire que la livraison causale multi-groupes garantit que : si la diffusion d'un message (m, c) précède causalement la diffusion d'une message (m', c') (où c et c' sont les groupes de diffusion des messages m et m', respectivement), alors la livraison de m précède causalement la livraison de m' pour tous les participants p_k qui appartiennent à l'intersection des groupes c et c' [Mostefaoui et Raynal., 1993].

Le sur-coup lié à l'information de contrôle estampillée par message et celle stockée localement pour le cas de la diffusion mono-groupe est $\theta(n)$ [Schwarz et Mattern., 1994], où n est le nombre de participants au groupe. De même, le sur-coup dans le cas de la diffusion multi-groupe est $\theta(g \times n)$ [Birman et al., 1991], où g est le nombre de groupes. En pratique, il n'est pas toujours nécessaire de transmettre toutes les informations de contrôle avec chaque message.

En maintenant la relation de dépendance immédiate , il est possible de réduire la quantité d'information de contrôle. Ceci s'adapte à des communications fréquentes comme c'est le cas dans les activités coopératives.

La taille de l'information de contrôle attachée à un message m dépend, donc, du nombre de messages concurrents liés à m par la RDI. Dans le meilleur cas, quand nous traitons un message avec causalité totale, la taille de l'information de contrôle est $|IC| = 1$. Puisque les messages diffusés par un participant ne peuvent pas être mutuellement concurrents [3], pour le cas le plus défavorable l'information de contrôle est $|IC| = n$. Dans [3O], il est montré que lorsque le nombre de participants croit, la probabilité du pire cas décroît. Cette probabilité vaut 0.0497 pour un groupe de 3 participants.

4.2.2.2 La relation de dépendance immédiate

La relation de dépendance immédiate (RDI) [Prakash et al., 1997] est le seuil de propagation de l'information de contrôle, CI, concernant les messages introduits dans le passé causal et qui doivent être transmis pour assurer une livraison causale. Nous la dénotons par le symbole $\downarrow$, et sa définition formelle est la suivante :

Définition 4.11 (relation de dépendance immédiate $\downarrow$ (RDI)) $m \downarrow m' \Leftrightarrow [(m \rightarrow m') \wedge \forall m'' \in M, \neg(m \rightarrow m'' \rightarrow m')]$

Ainsi, un message m précède directement un message m', ssi aucun autre message m'' appartenant à M n'existe (M est l'ensemble de messages du système), tels que m'' appartient en même temps au futur causal de m, et au passé causal de m'.

3. Ceci est une hypothèse de l'algorithme développé pour améliorer l'efficacité du traitement de la causalité. Cette hypothèse est réaliste pour les application coopératives distribuées.

Cette relation est importante puisque si la livraison des messages respecte l'ordre de leur diffusion pour tous les couples de messages dans la RDI, alors la livraison respectera la livraison causale pour tous les messages. Cette propriété est formalisée par la proposition suivante. Elle montre que l'information causale limitée aux messages précédant immédiatement un message donné est suffisante pour assurer une livraison causale d'un tel message.

Proposition 4.1 (Relation entre RDI et la livraison causle)
Si $\forall m, m' \in M, m \downarrow m' \Rightarrow \forall k \in c : delivery_k(m) \rightarrow delivery_k(m')$
alors $m \rightarrow m' \Rightarrow \forall k \in c \ : delivery_k(m) \rightarrow delivery_k(m')$ ■

4.2.2.3 Réalisations

Pour résoudre le problème de la causalité des événements de communication, un service de diffusion causalement cohérent a été conçu et implanté selon une architecture totalement distribuée. Pour cela des protocoles causals efficaces mono-canal et multi-canaux ont été définis. Cette efficacité s'exprime en termes de quantité d'information de contrôle estampillée par message. A la différence des protocoles standards proposés par [Baldoni et al., 1997, Mostefaoui et Raynal, 1995, Rodrigues et Verissimo, 1995], les protocoles dévelopés dans le cadre de la thèse de Pomares ciblent les applications distribuées coopératives et ne font aucune hypothèse sur la topologie du réseau ni sur la structure logique du groupe de coopération. La première version du protocole mono-canal causal [1J] est basée sur la relation de dépendance immédiate (RDI), c'est à dire, que la seule information de contrôle estampillée par message pour des messages émis par un même participant, appartient aux messages qui sont arrivés entre l'événement "$envoi(m_t)$" et l'événement "$envoi(m_{t+1})$". A partir de la relation RDI, un nouvel algorithme a été construit. Il borne la taille de l'information de contrôle sans perdre la cohérence causale. Les résultats ont été ensuite étendus pour résoudre la causalité dans un environnement multi-canaux [3O]. Cet algorithme a été intégré à l'outil d'édition coopérative. Pour cela, une interface compatible avec l'API JSDT a été développée par S. Pomares [13Ci]. Ceci a facilité le développement de la version causale de l'éditeur Sedit dont la version initiale était basée sur la bibliothèque JSDT. Cette étude a été conduite, aussi, grâce à la participation de Jean Fanchon, pour la partie protocoles, et Mehdi Eljed, pour la partie application à l'édition coopérative.

4.3 Coordination guidée par la structure

De nombreux travaux se sont intéressés aux problèmes de coordination liés à l'évolution des structures de coopération (groupes ou sites) et au partage de l'espace de coopération. Parmi les plus récents, nous citons en particulier ceux qui se basent sur les modèles formels et ciblent la spécification [Métayer, 1998] et ceux qui ciblent la conception et la programmation [Omicini et al., 2001, Pellegrini et Riveill, 1999]. Notre approche intègre les avantages de ces deux directions. De façon similaire à l'approche de [Métayer, 1998], nous utilisons la transformation des graphes pour décrire l'évolution de l'architecture dynamique et nous exploitons cette description pour générer les règles de coordination et leur implantation dans une technologie de middleware.

4.3.1 La description des interdépendances structurelles par les graphes de coordination

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 objets distribués du standard CORBA. L'architecture distingue trois niveaux fonctionnels : (1) La communication, (2) la coordination, et (3) la coopération. L'environnement logiciel CDK qui implante cette architecture (sous Orbix, Orbacus et CorbaScript [Philippe Merle et al., 1998]) fournit un service de communication à granularité objets pour la communication de groupe. 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. Des expérimentations ont été conduites dans le domaine de l'édition coopérative pour le niveau coopération. Ces différents points seront détaillés dans la suite.

Notre approche se base sur les graphes de coordination [30Ci] que nous avons 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 symétrique 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'implantation de ce modèle est décrite dans [31Ci] pour la version qui implante le comportement par les objets Java et dans [30Ci] pour la version qui implante le comportement par les objets corba.

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'unification 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 deux 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 Figure 4.7, et Figure 4.8).

Des formats visuels ou textuels peuvent être utilisés pour décrire une règle. Nous présentons, dans la figure 4.7, un exemple de représentation de chaque catégorie. La règle décrite est extraite du modèle de coordination linéaire utilisé pour la gestion de l'accès par zone dans les outils d'édition coopérative [30Ci].

Nous présentons, dans la figure 4.6, 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. Cette même règle est décrite de façon complète dans les listings Java décrits par les figures 8.12, 8.13 et 8.14 et fournis en annexe.

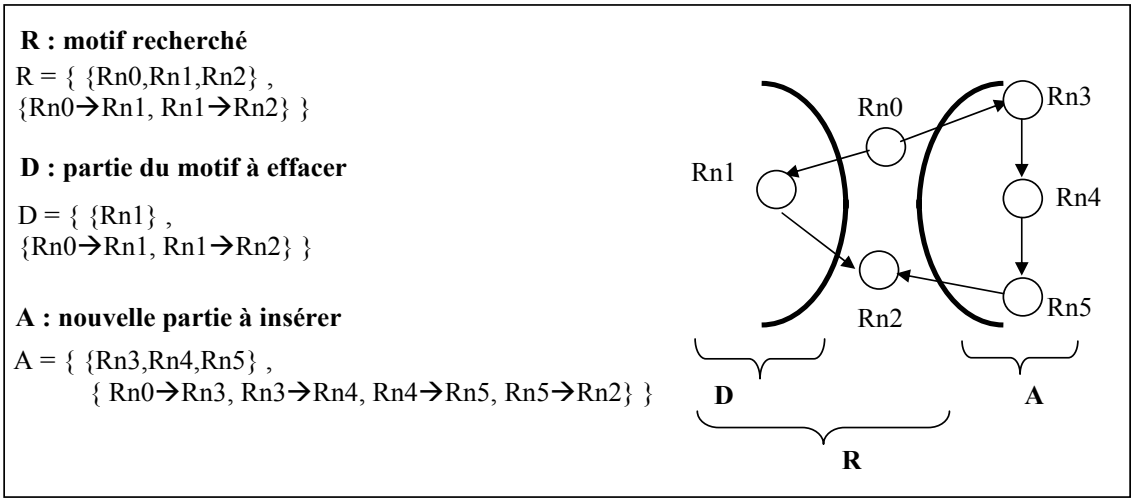


FIG. 4.6 – *Notation visuelle pour décrire une règle simplifiée de transformation de graphe*

La notation visuelle (figure 4.6) 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”, “**(**”.

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.

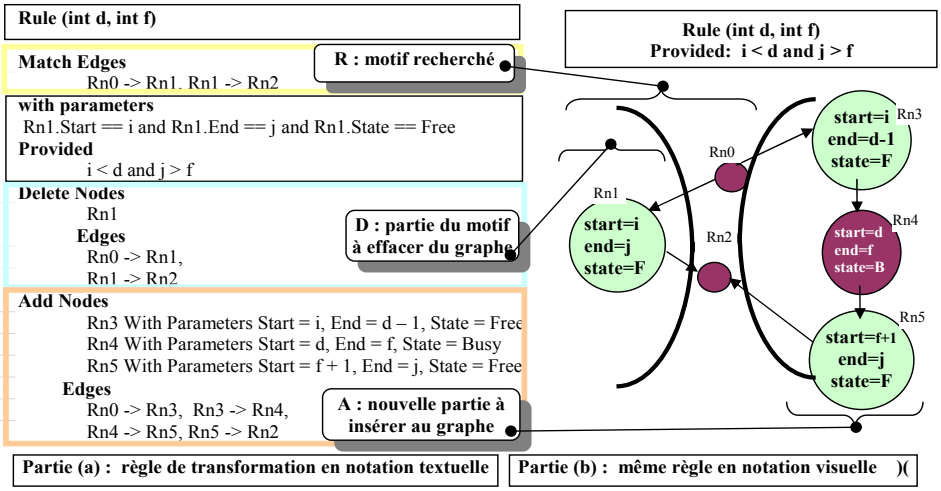


FIG. 4.7 – *Exemple de règle simplifiée au format textuel [3J] et visuel*

4.3.2 Application à la gestion de l’architecture dynamique des sites de coopération

Le système coopératif à réaliser est conçu comme un ensemble coordonné d’entités spécialisées dans l’échange et le traitement d’information (pouvant être des objets multimédias) dont le comportement est décrit par le programmeur qui utilise cet environnement. Les règles de fonctionnement du système à un instant donné sont alors définies par les capacités de coopération entre les entités qui constituent sa configuration à cet instant.

Une configuration est définie comme une “composition dynamique d’entités coopérantes”. La prise en compte de changements du “contexte coopératif” (relatifs par exemple au changement du type d’information traitée) est traitée par un changement de configuration dont la réalisation est à la charge de l’environnement selon un protocole de reconfiguration orienté règles de coordination qui sont décrites par le programmeur et qui peuvent être modifiées ainsi que le protocole de coordination qui les utilise.

Ceci permet la dé-corrélation entre les fonctions d’échange d’informations de coordination entre les entités qui offrent le service au niveau coopératif et l’échange des informations de co-

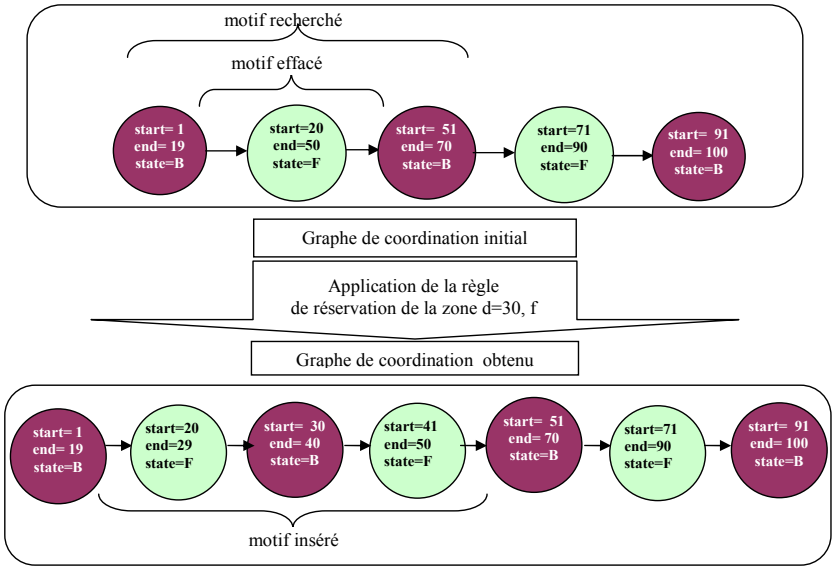


FIG. 4.8 – *Exemple de graphe et de transformation par la règle de* FIG. *4.7*

opération propres à ce niveau. De façon générale, le système est décrit au niveau coordination, comme “un graphe d’entités coopératives coordonnées” qui est transformé de façon adéquate avec l’évolution de l’activité de coopération et ses besoins de coopération. Les règles de transformation constituent l’outil de programmation des applications coopératives permettant, par exemple, de changer de règles de partage d’un espace de travail sans remettre en cause le comportement des entités coopérantes spécialisées dans le traitement de l’information partagée.

Dans le reste de 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.

4.3.2.1 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] :

– “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 d’organiser ses éléments et leurs interactions.

– “La description de type style” (”architecture as style”) 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 [6J],

[30Ci], [31Ci]. Des travaux récents sur la “description de type style” ont été commencés dans le cadre de la thèse de Karim Guennoun [5Ci], [3Cn].

4.3.2.2 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 ’m’ *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 défini dans [1Ci] et 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.

4.3.2.3 Formalisation

Dans cette section, nous utilisons les notations récentes définies dans [3Cn] qui se basent sur la structure abstraite de graphe appelée ACG (Graphe Abstrait de Composants) définie initialement dans [5Ci] . 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 [4], 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

4. 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.

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[5] *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.

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

– **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.

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

5. 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.

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.

***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.

***Trans**:* $\mathcal{P}$*(*α *(:*$\mathcal{P}$*(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 4.12 (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 4.13 (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 4.14 (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 4.15 (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,*

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

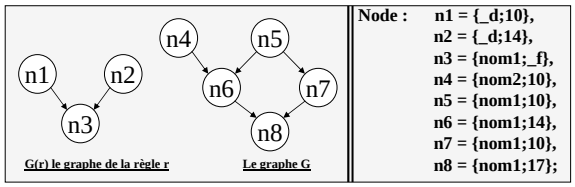


FIG. 4.9 – *G(r) le graphe de la règle r, et G le graphe de l'architecture courante*

Exemple

Soit le graphe de règle G(r) et le graphe G décrits dans FIG.4.9.
On a alors,

1. - **S_Unifiable**([n1,n2,n3],[n4,n5,n6])?
   $\Longleftrightarrow$
   Unifiable(n1,n4) $\wedge$ **Unifiable**(n1.Successeurs,n4.Susccesseurs) $\wedge$
   Unifiable(n2,n5) $\wedge$**Unifiable**(n2.Successeurs,n5.Susccesseurs) $\wedge$
   Unifiable(n3,n6) $\wedge$**Unifiable**(n3.Successeurs,n6.Susccesseurs)
   $\Longrightarrow$
   $(\_d = nom2) \wedge (10 = 10) \wedge (\_d = nom1) \wedge \ldots$
   $\Longrightarrow$Inconsistance: _d unifié avec deux valeurs différentes .
   $\Longrightarrow$ **Faux**.
2. - **S_Unifiable**([n1,n2,n3],[n7,n6,n8])?
   $\Longleftrightarrow$
   $(\_d = nom1) \wedge (10 = 10) \wedge (\_d = nom1) \wedge (14 = 14) \wedge (nom1 = nom1) \wedge (\_f = 17) \wedge (\_f > 15) \wedge$
   Unifiable({n3},{n8}) $\wedge$ (**Unifiable**({n3},{n8}).
   $\Longleftrightarrow$
   $(\_d = nom1) \wedge (10 = 10) \wedge (\_d = nom1) \wedge (14 = 14) \wedge (nom1 = nom1) \wedge (\_f = 17) \wedge (\_f > 15) \wedge (nom1 = nom1) \wedge (\_f = 17) \wedge (\_f > 15) \wedge$ **Unifiable**({ },{ }) $\wedge(\_d = nom1)$
   $\Longleftrightarrow$ **Vrai**.

Définition 4.16 (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,*

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

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)*.

4.3.2.4 Environnement de conception associé

L'approche de gestion des architectures dynamiques des sites de coopération a été implantée dans un environnement CORBA décrit par la figure 4.10. Une application distribuée coopérative conçue sur cet environnement utilise :

– le service de communication de groupe (au dessus de la communication Corba point à point) pour diffuser les informations de coopération et les informations de coordination (requêtes, réponses pour le floor-control et instructions de modification d'architecture vers les co-ordinateurs locaux)

– le service de coordination basé sur la transformation du graphe de coordination

– Les librairies de programmation des objets coopératifs et des fabriques d'objets (programming facilities).

Une première version de la réalisation de cet environnement [6] a considéré que les objets coopératifs sont programmés en Java [30Ci]. Cette version permet aussi la programmation des applications comme des applets pour un déploiement dynamique et exécution via les navigateurs. La 2ème version [7] a permis d'élargir le support de programmation au langage C++ [31Ci].

Cette approche a été aussi expérimentée avec succès pour l'implantation de 2 versions d'un outil d'édition partagée pour des documents au contenu formatté en "Texte" [30Ci] puis en "HTML" [3J]. Les règles de coordination ont été décrites par le modèle linéaire présenté dans la section 4.3.4.1.

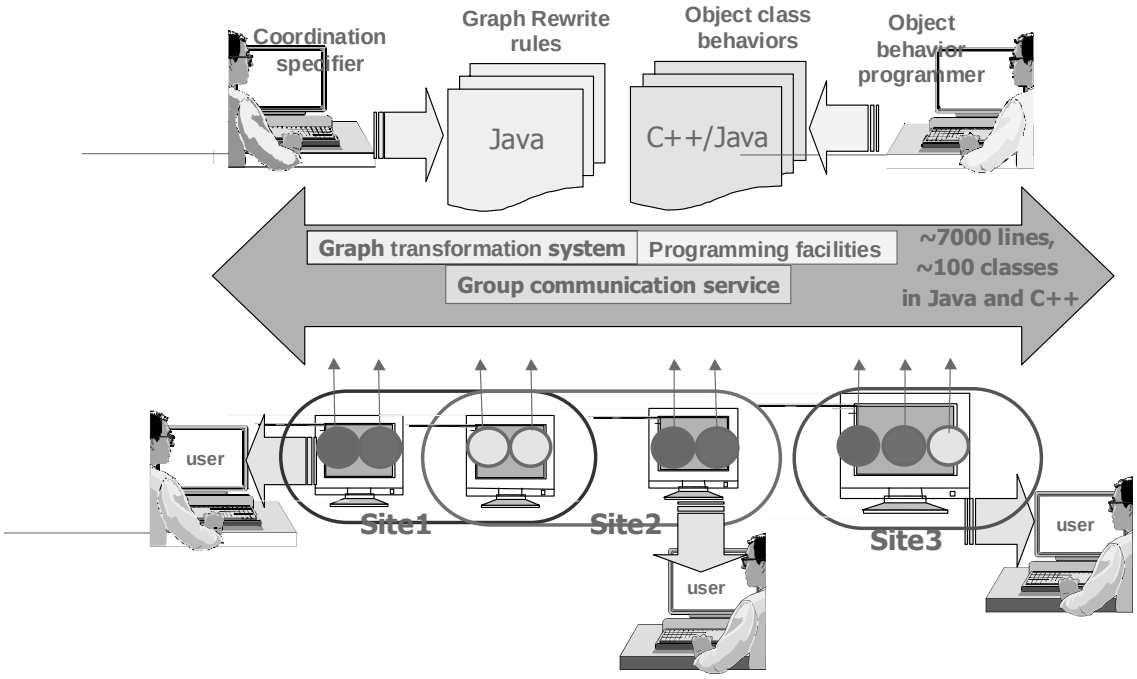


FIG. 4.10 – *Structure de l'environnement de gestion des architectures dynamiques sous Corba.*

6. Stage élève ingénieur de F. Gouëzec.
7. Stages de DEA de F. Gouëzec et A. Hbabi.

4.3.3 Application à la gestion de l'espace de coopération

L'échange et le partage de l'information sont les deux fonctions principales dans la gestion de l'espace de coopération. Nos contributions dans cette partie couvrent ces deux fonctions.

Comme souligné par la synthèse sur le groupware dans [Ellis et al., 1991], pour favoriser et améliorer l'efficacité de la collaboration, la gestion du partage d'information ne doit pas se baser sur des politiques de verrouillage strictes classiques mais doit offrir des mécanismes de gestion du partage à granularité plus fine. Ceci constitue l'objectif de notre approche de contrôle guidée par la structure de l'espace de coopération. Nous avons élaboré des modèles basés sur les graphes de coordination qui permettent une coordination guidée par la structure de l'espace partagé [30Ci], [31Ci], [3J]. L'objectif est d'éviter les conflits lorsque la distribution des rôles ne permet pas de le faire en amont de l'activité coopérative.

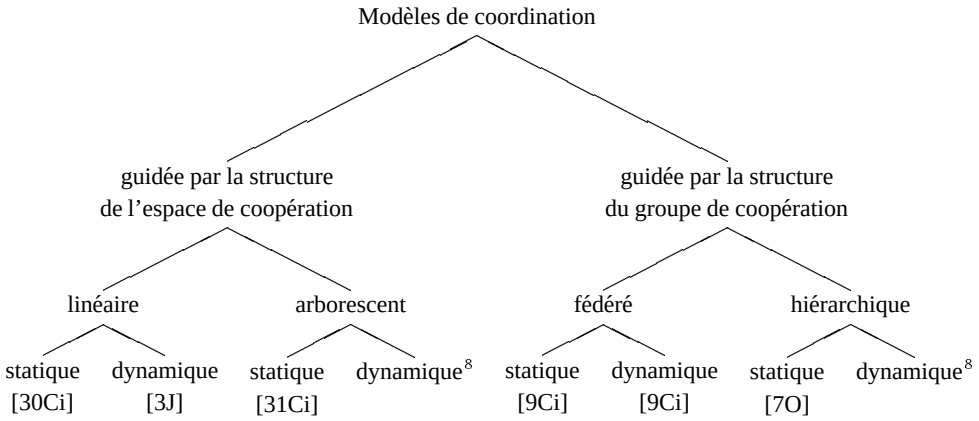
D'un autre côté, et pour coordonner l'échange de l'information dans les sessions de coopération synchrone, nous avons élaboré des modèles basés sur les graphes de coordination qui permettent une coordination guidée par la structure du groupe de coopération. Il s'agit notamment :

– de gérer la concurrence inhérente aux activités des groupes de coopération pour que la distribution des tâches ne conduit pas à des accès simultanés aux mêmes information [7O] par différents participants.

– de gérer la cohérence entre la distribution des rôles et la distribution des outils de coopération en charge de produire ou de consommer les informations échangées [9Ci]. L'objectif est de contrôler et de favoriser l'échange d'information entre les participants.

4.3.4 Classes des modèles développés

L'ensemble des modèles que nous avons développés en utilisant les graphes de coordination permettent ainsi une gestion intégrée du partage (ou l'échange de l'information) et de l'évolution de l'architecture dynamique des sites de coopération.



TAB. 4.1 – *Classes des modèles de coordination développés*

Les différentes classes de modèles basés sur les graphes de coordination sont résumées dans le schéma de la table 4.1 illustre cette classification.

Les modèles se déclinent en deux familles : les modèles orientés structure de l'espace partagé, et les modèles orientés structure du groupe de coopération. Pour la première famille, nous avons développé le modèle linéaire statique [30Ci] et dynamique [3J] et un modèle arborescent statique [31Ci] puis dynamique [8]. Pour la deuxième famille, nous avons développé les modèles fédéré [31Ci] et hiérarchique dynamiques et statiques [9Ci].

4.3.4.1 Exemple : le modèle linéaire dynamique appliqué au partage de documents

Le partage d'un document de type texte peut, selon les besoins de l'activité, aller d'un granularité caractère, vers la granularité document entier en passant par toutes les granularités d'une structure linéaire : mot, ligne, paragraphe, ou séquence de ces derniers, que nous appelons "zone".

Le modèle linéaire représente l'état d'un document par une succession de zones libres ou réservées. L'attribution des zones aux différents rédacteurs se gère de façon dynamique. Aucun rédacteur ne détient a priori le privilège pour une zone déterminée. Comme l'illustre la figure 4.11, la répartition en zones libres ou occupées peut varier dans le temps depuis le début jusqu'à la fin de la procédure de rédaction coopérative : l'état d'une zone peut ainsi passer de l'état libre à l'état réservé. Cette répartition peut varier aussi entre les utilisateurs et à l'intérieur de la structure du document : une zone peut être réservée à différents auteurs durant les différentes phases de rédaction, et les limites d'une zone réservée peuvent varier pour s'élargir ou se rétrécir durant les différentes phases de rédaction.

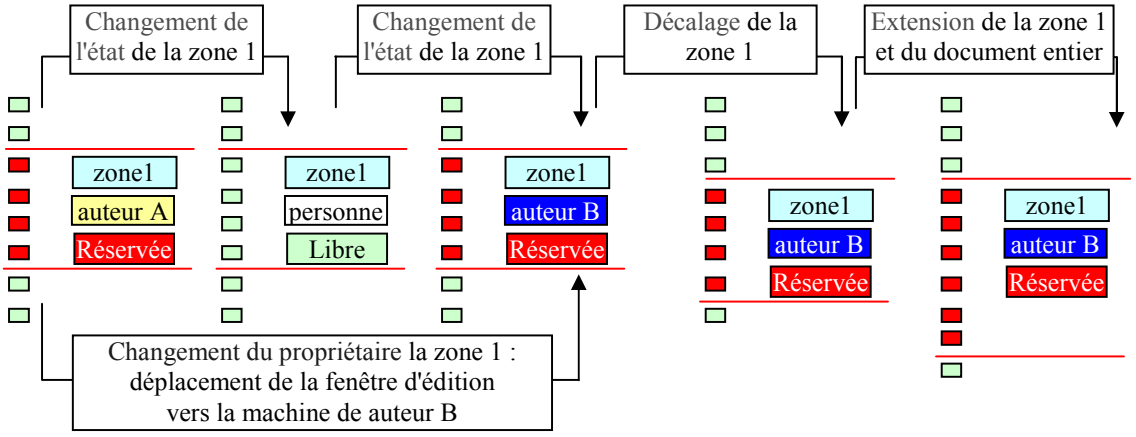


FIG. 4.11 – *Partage de texte dans un document selon une structure linéaire.*

Nous avons modélisé ces différentes variations de répartitions selon un modèle linéaire dynamique. Selon ce modèle, l'état du document est représenté par un graphe dont les nœuds peuvent représenter une zone disponible ou une zone déjà réservée.

A la demande de réservation, le service de coordination, en fonction de l'état du document (codé dans le graphe de coordination), décide de répondre favorablement ou négativement à la demande selon un protocole basé sur l'application de 9 règles de transformation de graphe : 4 pour la réservation, 4 pour la libération et 1 pour les décalages et extensions ou réductions de zones. Par analyse des différentes situations possibles, le protocole a été prouvé complet : aucune requête ne

8. Rapport de DEA E. Roblet

peut être refusée alors que la zone demandée est libre. Le protocole a été prouvé correct : aucune requête ne peut être acceptée si la zone demandée est encore en cours d'utilisation.

Les règles de réservation distinguent 4 situations possibles : (1) réservation d'une zone entière, (2) réservation strictement au milieu d'une zone, (3) réservation strictement au début d'une zone, (4) réservation strictement à la fin d'une zone.

Les règles de libération distinguent 4 situations possibles : (1) libération d'une zone entourée de deux zone libres, (2) libération d'une zone précédée d'une zone réservée et suivie d'un zone libre, (3) libération d'une zone précédée d'une zone libre et suivie d'un zone réservée, (4) libération d'une zone entourée de deux zone réservées.

Formalisation selon [3Cn] Nous supposons qu'un document est organisé en composantes (pouvant représenter des caractères, des lignes, des paragraphes, des pages etc. . .) numérotées de 0 à n. Un utilisateur doit, avant d'écrire dans une zone (une zone est un ensemble de composantes consécutives), la réserver. Une fois qu'un utilisateur a fini sa rédaction, il peut libérer la zone qu'il a auparavant réservée. Chaque composante du document ne peut se trouver que dans l'un des deux cas de figures suivants : ou bien elle est libre, ou bien elle est reservée par un utilisateur unique. Pour éviter une surcharge inutile dans la description de l'édition partagé, nous nous limitons à décrire les attributs des nœuds qui sont significatifs pour la coordination de l'application d'édition partagée.

Modélisation de l'architecture

L'architecture de l'application d'édition partagée est modélisée par un graphe linéaire représentant les zones consécutives du document (un nœud $n1$ est fils d'un nœud $n2$ `ssi` la zone repésentée par $n1$ est consécutive à celle représentée par $n2$). Les nœuds du graphe modélisant ces zones devront donc décrire les paramètres de début et de fin de zone, l'état de la zone (*F* pour l'état libre et *B* pour l'état réservé), le propriétaire de la zone [9], et la localisation de la zone (par exemple l'adresse IP de la machine ou se trouve le propriétaire [10]). Les nœuds de notre graphe auront donc pour champ un quintuplet contenant ces cinq paramètres (un exemple est donné dans FIG.4.12).

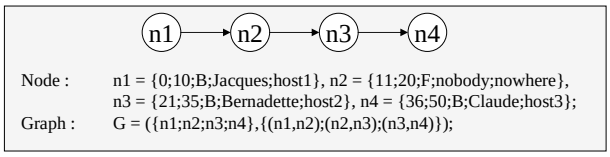


FIG. 4.12 – *Graphe de l'architecture représentant le document où les composantes, de* 0 *à* 10 *sont réservées par Jacques, les de* 11 *à* 20 *sont libres, de* 21 *à* 35 *sont réservées par Bernadette, et de* 36 *à* 50 *sont réservées par Claude.*

Les règles de coordination

9. dans le cas d'une zone libre ce paramètre sera positionné $\grave{a}$ *nobody*
10. ce paramètre sera positionné à *nowhere* dans le cas d'une zone libre

Les règles de réservation et de libération des composantes

Les règles de transformation de l'application d'édition partagée sont classées en trois catégories selon les événements qu'elles traitent. La première catégorie concerne la réservation, la deuxième concerne la libération et la troisième la vérification des deux propriétés de sûreté et de complétude. Par souci de simplification, et pour éviter le cas particulier d'un graphe avec un nœud unique [11], nous rajoutons deux nœuds (auxquels nous affectons des champs garantissant qu'ils ne soient jamais détruits, et qu'ils ne lèveront jamais une violation des règles de sûreté ou de complétude) l'un en tête et l'autre en queue du graphe à structure linéaire.

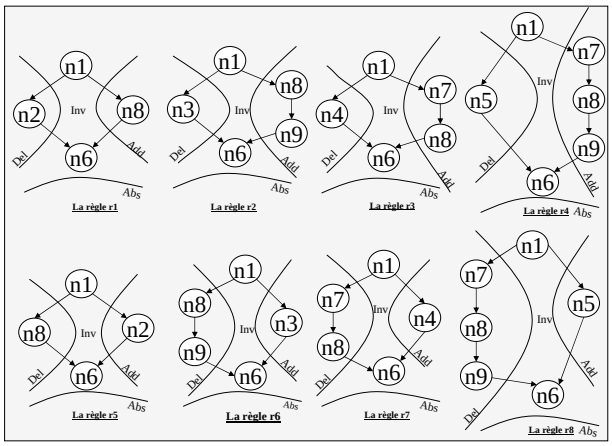


FIG. 4.13 – *Les graphes des règles de réservation et de libération.*

Considérons un utilisateur "owner" sur la machine "host" et voulant réserver la zone $[begin, end]$. Pour que cela puisse être possible, il doit exister dans le document une zone libre $[begin_f, end_f]$ *telle que* $[begin, end] \subseteq [debut_f, end_f]$. Quatre cas de figure sont possibles: 1) Cas 1 : $(begin_f = begin) \wedge (end_f = end)$, 2) Cas 2 : $(begin_f = begin) \wedge (end_f > end)$, 3) Cas 3 : $(begin_f < begin) \wedge (end_f = end)$, 4) Cas 4 : $(begin_f < begin) \wedge (end_f > end)$.

Considérons maintenant un utilisateur "owner" sur la machine "host" et possédant la zone z, lors de la libération de cette zone, quatre cas de figures sont possibles: 1) Cas 5 : La zone précédant z et celle qui la suit sont libres, 2) Cas 6 : La zone précédant z et celle qui la suit sont réservées, 3) Cas 7 : La zone précédant z est libre, et celle qui la suit est réservée, 4) Cas 8 : La zone précédant z est réservée et celle qui la suit est libre.

On obtient donc les quatre règles de réservation ($r1, r2, r3, r4$) et les quatre règles de libération ($r5, r6, r7, r8$) présentées dans FIG.4.13. La réservation (respectivement la libération) d'une zone pour un utilisateur est donc possible *ssi* l'une des quatre règles r1, r2, r3, ou r4 (respectivement r5, r6, r7, ou r8) est applicable au graphe courant. Il est, aussi, à noter que la fonction d'unification telle qu'elle a été définie nous assure qu'un utilisateur ne peut libérer une zone que s'il en est le propriétaire.

Boolean supAtt1 (P(Nodes) {N1;N2}) { *return(N1.champ(1) > N2.champ(1));* }

Boolean supAtt0 (P(Nodes) {N1;N2}) { *return(N1.champ(0) > N2.champ(0));* }

11. le traitement de ce cas particulier doublerait inutilement le nombre de règles de réservation et de libération

Node : *n1 = {_x;_y;B;_owner1;_host1},n2 = {_begin;_end;F;nobody;nowhere},*
n3 = {_begin;_end';F;nobody;nowhere},n4 = {_begin';_end;F;nobody;nowhere},
n5 = {_begin';_end';F;nobody;nowhere},n6 = {_a;_b;B;_owner2;_host2},
n7 = {_begin';_begin-1;F;nobody;nobody},n8 = {_begin;_end;B;_owner;_host};
n9 = {_end+1;end';F;nobody;nowhere};

Graph : *Inv = ({n1;n6},{ }),Del(r1) = ({n2},{(n1,n2);(n2,n6)}),*
Del(r2) = ({n3},{(n1,n3);(n3,n6)}),Del(r3) = ({n4},{(n1,n4);(n4,n6)}),
Del(r4) = ({n5},{(n1,n5);(n5,n6)}),Add(r1) = ({n8},{(n1,n8);(n8,n6)}),
Add(r2) = ({n8;n9},{(n1,n8);(n8,n9);(n9,n6)}),
Add(r3) = ({n7,n8},{(n1,n7);(n7,n8);(n8,n6)}),
Add(r4) = ({n7;n8,n9},{(n1,n7);(n7,n8);(n8,n9);(n9,n6)});

Const : *C(r2)={(supAtt1,{n9;n8})},C(r3)={(supAtt0,{n8;n7})},*
C(r4) ={(supAtt1,{n9;n8});(supAtt0,{n8;n7})};

Rule : *r1 = (Inv,Del(r1),Add(r1),{ },C(r1)),r2 = (Inv,Del(r2),Add(r2),{ },C(r2)),*
r3 = (Inv,Del(r3),Add(r3),{ },C(r3)),r4 = (Inv,Del(r4),Add(r4),{ },C(r4)),
r5 = (Inv,Add(r1),Del(r1),{ }, { }),r6 = (Inv ,Add(r2),Del(r2),{ }, { }),
r7 = (Inv ,Add(r3),Del(r3),{ }, { }),r8 = (Inv ,Add(r4),Del(r4),{ }, { }),

Vérification des propriétés de sûreté et de complétude

Les deux propriétés de sûreté[12] et de complétude[13] seront vérifiées par l'unification des deux

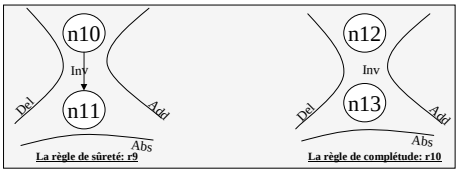


FIG. 4.14 – *Les règles pour la vérification de la sûreté et de la complétude.*

règles les représentant (FIG.4.14) déclarées comme suite :

Boolean Intersect (P(Nodes){n1;n2}) {

return((n1.champ(0) ≤ n2.champ(1))AND(n1.champ(0)≤ n2.champ(1)));}

Node : *n10={_x;_y;F;nobody;nowhere},n11={_a;_b;F;nobody;nowhere},*
n12={_x;_y;_state1;_owner1;_host1},n13={_a;_b;_state2;_owner2;_host2};

Graph : *Inv(r9)=({n10;n11},{(n10,n11)}),Inv(r10)=({n10;n11},{ }),Empty=({ },{ });*

Constr : *C(r10)={(Intersect,{n12;n13})};*

Rule : *r9=(Inv(r9),{ },{ },{ },{ }),*
r10=(Inv(r10),{ },{ },{ },C(r10));

Une unification possible d'une de ces deux règles indiquerait que la propriété correspondante est violée dans l'architecture courante.

Les événements traités par l'application

L'application d'édition Partagée doit traiter quatre types d'événements. Le premier concerne une demande d'un utilisateur de réserver une zone du document. Un événement de ce type doit donc transporter le début et la fin de la zone à réserver, le nom ou l'alias du demandeur, et la machine sur laquelle se il trouve. Le deuxième concerne une demande de libération. Un événement de ce type

12. La sûreté est violée ssi il existe deux zones consécutives libres
13. La complétude est vérifiée ssi pour chaque paire de zones, l'intersection est vide

transportera exactement le même nombre et types de paramètres qu'un événement de réservation. Le troisième concerne la vérification de la propriété de sûreté, ce type d'événement ne transportera aucun paramètre et le quatrième concernera la propriété de complétude et ne transportera également aucun paramètre. On obtient donc la description suivante :

Event : Book = ("book", {begin(:Int);end(:Int);owner(:String);host(:Int)}),
Free = ("free",{ begin(:Int);end(:Int);owner(:String);host(:Int)}),
VerifySafety = ("safety",{ }),
VerifyCompletness = ("completness",{ });

Le protocole de coordination

L'événement *Book* doit répercuter ses paramètres sur les nœuds des règles $r1, \ldots, r4$ et donc les nœuds $n1, \ldots, n9$, l'événement *Free* sur les règles $r5, \ldots, r8$ et donc les nœuds $n1, \ldots, n9$, l'événement *VerifySafety* sur la règle $r9$ et donc les nœuds $n10$ et $n11$, et l'événement *VerifyCompletness* sur la règle $r10$ et donc les nœuds $n12$ et $n13$. Au vu de ce qui a été développé précédemment, le protocole de coordination pour l'architecture de l'application d'édition partagée peut-être décrit sous la forme suivante :

Node : *InitialNode = (⊤, ⊤, B, nobody, nowhere),*
FinalNode = (⊥, ⊥, B, nobody, nowhere),
node1 = (0, Nb_Composantes - 1, F, nobody, nowhere);
Graph : *G = ({InitialNode; node1; FinalNode},*
{(InitialNode, node1);(node1, FinalNode)});
ProtocolCoo : *EPCoo = (G,*
{r1; r2; r3; r4; r5; r6; r7; r8; r9; r10},
{Book; Free; VerifySafety; VerifyCompletness},
{("book", {r1;r2;r3;r4}, {(T, {n1;...;n8})});
("free", {r5;r6;r7;r8}, {(T, {n1;...;n8})});
("safety", {r9}, { });
("completness", {r10}, { })});

Où,

```
Void T ( Event event, P(Nodes) nodes)
        {for ( node in nodes)
        {if (node.champ(0) == _begin)     { node.champ(0) = event.parameters(0);}
        if (node.champ(0) == _end+1)      { node.champ(0) = event.parameters(1) + 1;}
        if (node.champ(1) == _end)        { node.champ(1) = event.parameters(1);}
        if (node.champ(1) == _begin-1)    { node.champ(1) = event.parameters(0) - 1;}
        if (node.champ(3) == _owner)      { node.champ(3) = event.parameters(2);}
        if (node.champ(4) == _host)       { node.champ(4) = event.parameters(3);}}}
```

4.4 Environnements logiciels pour la coopération

4.4.1 Cadre Conceptuel et architecture d'intégration

Après les études présentées, nous avons abordé la définition d'un Cadre Conceptuel et d'une architecture d'intégration opérationnelle des différents sous-systèmes formant l'environnement global de coopération (Figure 4.15). Ceci est particulièrement utile pour l'intégration des applications coopératives que nous n'avons pas développées nous mêmes selon l'approche guidée par le

modèle. Cette étude a été appliquée au domaine de la téléformation [4J].

L'architecture développée tient compte des différents types d'interaction : la communication, la coordination et la coopération. Elle soutient leur séparation en permettant à chaque application coopérative d'utiliser son propre protocole de communication : Multicast fiable pour les applications distribuées coopératives non sensibles à des contraintes temporelles (comme les outils de partage d'applications ou de partage de documents tels que le "white board"), ou Multicast non fiable pour les applications sensibles aux contraintes temporelles (comme les outils de diffusion audio-vidéo telles que l'outil de visioconférence).

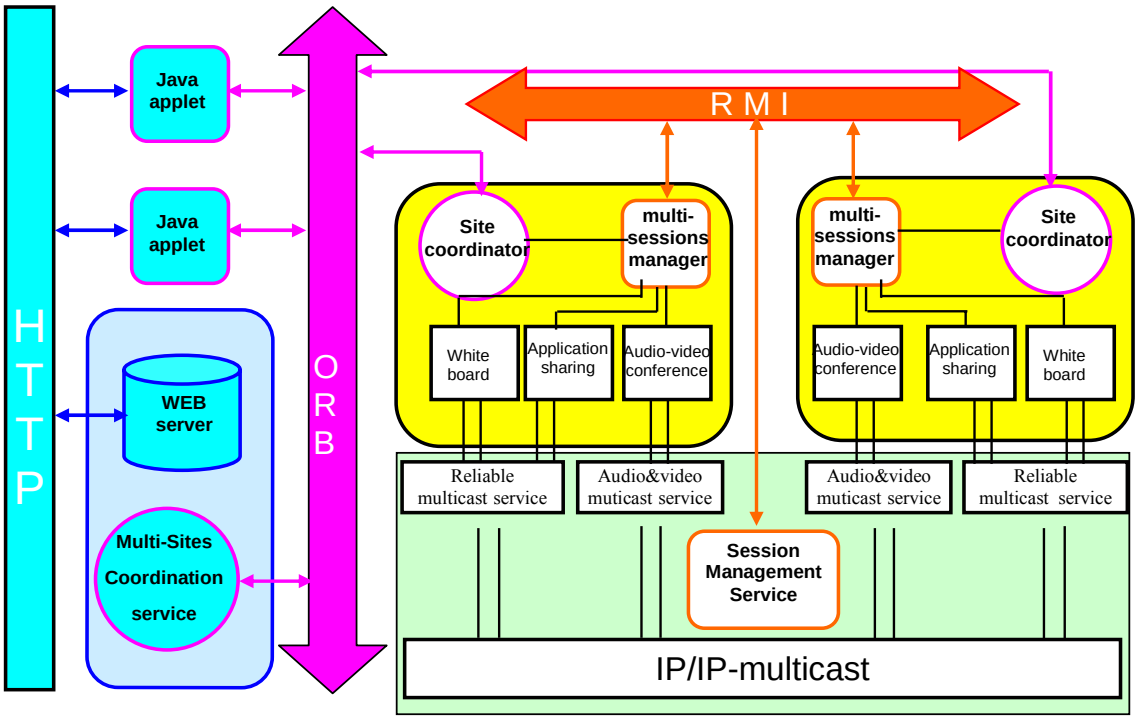


FIG. 4.15 – *Architecture d'intégration des composants et des services de coordination.*

L'architecture supporte l'utilisation d'un service de gestion d'architecture, et de déploiement dynamiques. Cet travaux sont en cours de développements dans le cadre des thèses de K. Guennoun [5Ci].

4.4.2 Réalisations

Dans le cadre des projets de recherche internes et contractuels, nous avons conçu et implanté un ensemble de composants élémentaires et de services que nous avons ensuite intégrés au sein d'outils et d'environnements plus complets pour reproduire, dans un espace de travail virtuel et distribué, les espaces de discussion et les outils de travail des réunions conventionnelles. Ceci inclut notamment les outils de conférences textuelles point-à-point ou multi-points, les outils d'annotation de documents HTML, mais aussi de nouveaux outils de travail permettant d'améliorer les anciennes procédures de travail, comme notamment les outils d'édition et de visualisation coopératives [3J], [11Ci].

Différentes versions d'outils d'édition coopérative ont été réalisées d'abord pour traiter des

documents au format TEXTE [30Ci] et ensuite pour traiter des document au format HTML [3J]. Dans un premier temps, nous avons adopté l'approche de développement orientée modèle utilisant notre environnement de développement sur CORBA et le modèle de coordination linéaire [30Ci], [3J]. Ensuite, nous avons conçu et développé une nouvelle version dans laquelle le document est répliqué sur les sites de chaque participant [11Ci]. Cette version est basée entièrement sur la technologie Java. Elle respecte le modèle de programmation orienté événement supporté par un service à événements dont nous avons implanté une version qui respecte la causalité entre les événements. L'éditeur du document peut jouer le rôle de rédacteur et apporter des modifications au contenu du document ou bien le rôle de lecteur (reviewer) et apporter des propositions en utilisant un système d'annotations textuelles.

La figure 4.16 illustre l'interface et l'architecture de la dernière version de l'outil l'édition partagée [11Ci]. Une publication présentant cet outil sera fournie en annexe de ce mémoire.

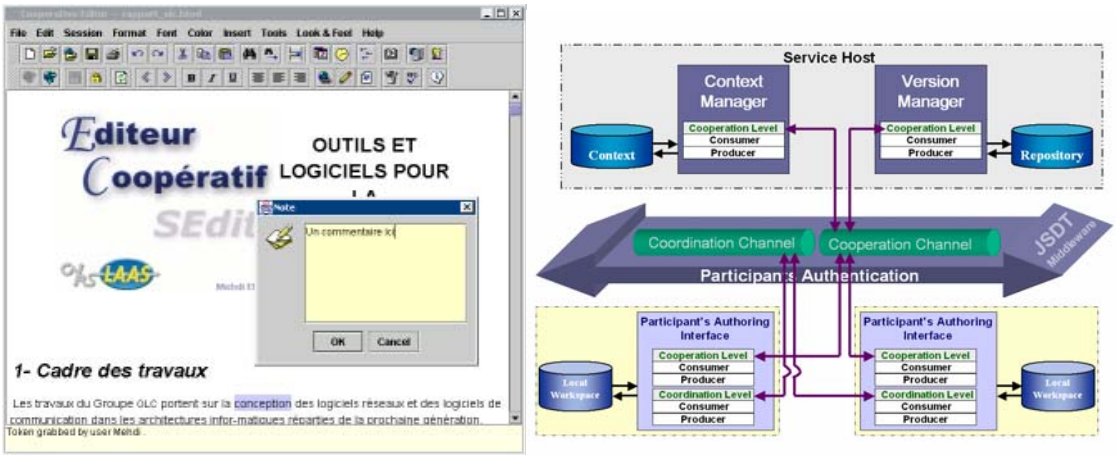


FIG. 4.16 – *Architecture de l'outil d'édition coopérative [11Ci].*

4.4.2.1 Expérimentations

Dans le cadre du projet européen IST DSE, les modules RMS et SMS ont étés validés dans deux scénarios coopératifs. Le premier s'agit d'un scénario coopératif orienté conception (Co-Des). Ce scénario, appelé Co-Des/PDR, est le processus d'étude de conception préliminaire "Preliminary Design Review". Le scénario est basé sur la conception du programme de développement du segment de vol de l'ATV (Automated Transfer Vehicle). L'ATV est une contribution de l'Europe à la station spatiale internationale (ISS). Le PDR est une étape de révision pendant laquelle la conception préliminaire de l'ATV est contrôlée conforme aux plans industriels et aux normes internationales. La fabrication, l'assemblage, l'intégration et les plans d'essai sont vérifiés comme conformes avec les dates de livraison prévues. Pendant le processus PDR, des spécialistes des diverses domaines analysent un ensemble de documents de conception afin de trouver des désaccords vis-à-vis des exigences prédéfinies. Une base cohérente et vérifiée de documents de conception est le résultat de cette activité. Le scénario PDR a été constitué de différents sites pour les serveurs ou les utilisateurs localisés à travers l'Europe : EADS-LV (Les Mureaux, Paris), LIP6 (Paris), LAAS (Toulouse), ALENIA (Turin), D3 Group (Berlin). La figure 4.17 décrit une

configuration de Distribution des rôles et des outils pour ce scénario.

Le deuxième est un scénario coopératif orienté validation (Co-Ver). Il s'agit de la conception des systèmes de contrôle dans le domaine spatial. Ce scénario décrit l'interconnexion de deux simulateurs distribués afin de vérifier le scénario de simulation de l'arrimage entre l'ATV et l'ISS. L'objectif principal est d'aider les ingénieurs dans la préparation et dans l'exécution de la simulation, à travers l'analyse coopérative des résultats de simulation depuis leur site de travail.

Le scénario Co-Ver a été constitué de différents sites pour les serveurs ou les utilisateurs localisés à travers l'Europe : EADS-LV (Les Mureaux, Paris), LIP6 (Paris), LAAS (Toulouse), ALENIA (Turin), D3 Group (Berlin). La figure 4.18 décrit une configuration de Distribution des rôles et des outils pour ce scénario.

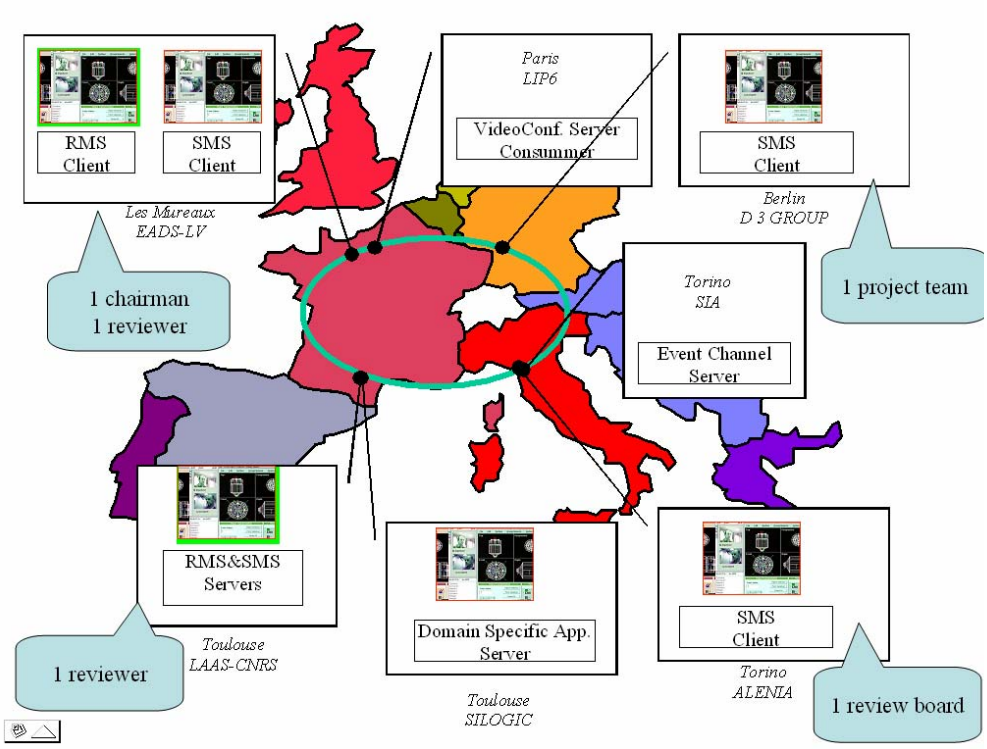


FIG. 4.17 – *Une Distribution des rôles et des outils pour le scénario CoDes.*

4.5 Synthèse classifiée des contributions

4.5.1 Contributions pour la gestion des interdépendances événementielles

Pour la **gestion des interdépendances événementielles**, nous avons conduit une étude pour le niveau communication et une étude pour le niveau coopération.

– La **coordination des événements de coopération** a été développée dans le cadre de la thèse de Martin Molina. Les solutions proposées implantent des services de gestion centralisée basée sur trois classes d'interdépendance que nous appelons **précédence**, **déclenchement**, et

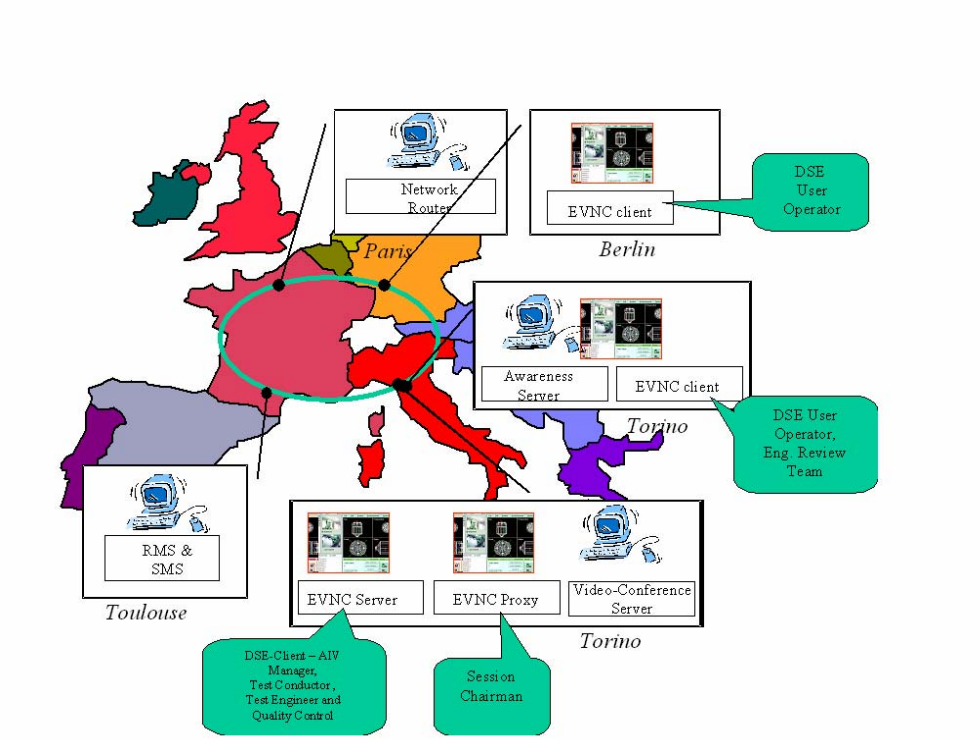


FIG. 4.18 – *Une Distribution des rôles et des outils pour le scénario CoVer.*

inhibition [10Ci], [7Ci]. Ces solutions couvrent les étapes de préparation [14Ci] et d'exécution des sessions par contrôle et déclenchement des événements de coopération [19Ci], [15Ci].

– Pour la **coordination des événements de communication**, des solutions distribuées appropriées ont été définies par extension de la relation de dépendance causale à la diffusion multi-canal dans le cadre de la thèse de Saul Pomarès. Ces solutions permettent de résoudre l'inconsistance de type n-producteurs / m-consommateurs dans les activités coopératives en maintenant l'ordre des interactions qui respecte la dépendance producteur / consommateur pour chaque exécution des actions [13Ci], [1J].

4.5.2 Contributions pour la gestion des interdépendances structurelles

Pour 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 [5Ci], leurs conditions et leurs conséquences, ceci de façon adaptée aux exigences du niveau coopération et aux contraintes et capacités du niveau communication [6J]. 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. Ces travaux ont été développés essentiellement dans le cadre des sujets de DEA de Frédéric Gouëzec [30Ci] et de thèse en cours de Karim Guennoun [5Ci].

  - Nous avons, dans ce contexte, défini des modèles génériques d'interdépendance pour les structures **linéaires** et **arborescentes**. Nous avons instancié et spécialisé ces modèles pour une gestion guidée par les structures de l'espace de coopération partagé servant pour la distribution du droit d'accès dans les activités d'édition coopérative (travaux de DEA de Etienne Roblet [3J] et Mehdi ElJed [11Ci])

- D'autres solutions de "**coordination guidée par la structure**" ont été développées dans le cadre des travaux de thèse de Laura Rodriguez [9Ci] et de DEA de Olfa Jenhani. Ces solutions s'articulent autour de la structuration de groupe de coopération et permettent d'initier et de contrôler les échanges entre les producteurs et les consommateurs d'information lors des sessions de coopération. Ces travaux ont inclus, respectivement : le développement d'un service de gestion de session accessible à l'utilisateur final des environnements de coopération; et un service de diffusion hiérarchique d'évenements, au niveau middleware par extension de JSDT, accessible au développeur d'applications logicielles pour environnements de coopération.

4.6 Conclusion

Nos contributions au domaine des systèmes distribués coopératifs ont couvert les différentes fonctions de gestion de la cohérence pour l'ensemble des sites et des groupes impliqués dans les activités de coopération. Nous avons adopté une démarche orientée modèle formel pour l'ensemble de ces fonctions. Nous avons défini des approches de coordination pour chacune des sémantiques d'interaction : synchrone et asynchrone. Nous avons considéré pour cela différentes classes de coordination guidées par les événements et par la structure. Notre démarche est utilisable pour la conception et le développement d'applications distribuées coopératives orientées composant. L'architecture de conception sous-jacente distingue trois niveaux fonctionnels : coopération, coordination et communication. La gestion et le contrôle de l'évolution de l'architecture d'une application distribuée coopérative concernent le niveau coopération et le niveau communication. Dans le premier cs, il s'agit de gérer le cycle de vie des composants et les relation d'interdépendances qui les relient ou qui relient leurs utilisateurs. Dans le deuxième cas, il s'agit de gérer les liens de communications pour permettre l'échange d'informations entre les composants ou entre leurs utilisateurs.

En complément de ce travail qui s'applique dès la phase de conception aux applications dont nous connaissons et gérons l'architecture interne, nous avons travaillé sur l'élaboration de solutions qui s'appliquent à la coordination dans un environnement de coopération conçu par intégration d'applications considérées comme monolithiques. Nous avons élaboré une architecture d'integration qui traite le niveau "instance d'application par site de coopération". Les réalisations et expérimentations de cette partie se basent sur la transformation ad-hoc des graphes de coordination. L'élaboration des modèles basés sur les règles de réécriture de graphe est une perspective de ce travail. Cette perspective permettrait de facilement généraliser les protocoles de gestion de session, élaborés pour la téléformation coopérative, à d'autres types d'activité coopérative structurée.

L'effort actuel porte essentiellement sur trois points que nous allons présenter dans les trois paragraphes qui suivent.

4.6.1 Description et gestion des architectures dynamiques orientées composants

Dans cette perspective, nous nous intéressons, d'une part, au développement de la notion de conformité d'une architecture à un style donné, la spécification. D'autre part, nous nous intéressons à la définition de règles de transformation qui garantissent la conformité de l'*architecture générée* au *style spécifié*. Cette perspective, en cours de développement, renforce la continuité dans la démarche orientée modèle dans le processus de conception qui distingue les phases de spécification et de développement. Dans ce cas, l'effort porte sur l'élaboration de deux modèles qui décrivent respectivement la spécification d'une architecture dynamique et ses règles de transformation. Des résultats dans cette direction ont été développés pour le modèle linéaire. L'effort actuel porte aussi sur l'élaboration de nouveaux modèles représentant les styles d'architecture de référence pour les systèmes récents, ainsi que la définition des règles de leur transformation avec préservation de la conformité. Nos études couvrent les styles fortement couplés, comme le style à deux niveaux du modèle "Client-Serveur" ou faiblement couplés, comme le style à trois niveaux du modèle "Producteur-Consommateur". La dynamique de l'évolution est traitée au niveau inter

et intra-composants dans chacun des deux cas.

Le cadre scientifique général de cette perspective inclut le développement de grammaires de graphes et de protocoles de coordination.

4.6.2 Description des architectures distribuées coopératives

Dans cette perspective, nous nous intéressons, au développement d'un ensemble de notations spécialisées pour la description des architectures distribuées coopératives. Il s'agit en particulier de définir les règles d'intégration et de distribution pour les applications qui forment un environnement de support pour la coopération.

Les aspects dont nous faisons abstraction pour généraliser, à toutes les architectures dynamiques, la description des styles traitée dans la première partie du travail décrite ci-avant, sont, dans cette deuxième partie, développés et décrits rigoureusement et explicitement de façon adaptée pour les architectures distribuées coopératives. Il s'agit en particulier de décrire la distribution des composants de chacun des trois niveaux fonctionnels sur les sites de coopération, et la description des règles de compatibilité de l'interaction locale ou distante, intra ou inter-niveaux, intra ou inter-applications, intra ou inter-sites entre les différents composants.

Le cadre scientifique général de cette perspective inclut le développement de langage de description d'architecture (ADL) et le support de développement logiciel.

4.6.3 Déploiement dynamique des composants d'une application distribuée coopérative

Cette perspective concerne la prise en compte de la phase de déploiement dans un contexte d'architectures dynamiques P2P appliquées pour l'adaptabilité des environnements de support pour la coopération. Ces travaux constituent le complément des deux parties précédentes. Ils traitent la gestion du déploiement dynamique des composants de façon conforme aux règles d'évolution dynamique de l'architecture et selon les règles de distribution des applications sur les sites de coopérations.

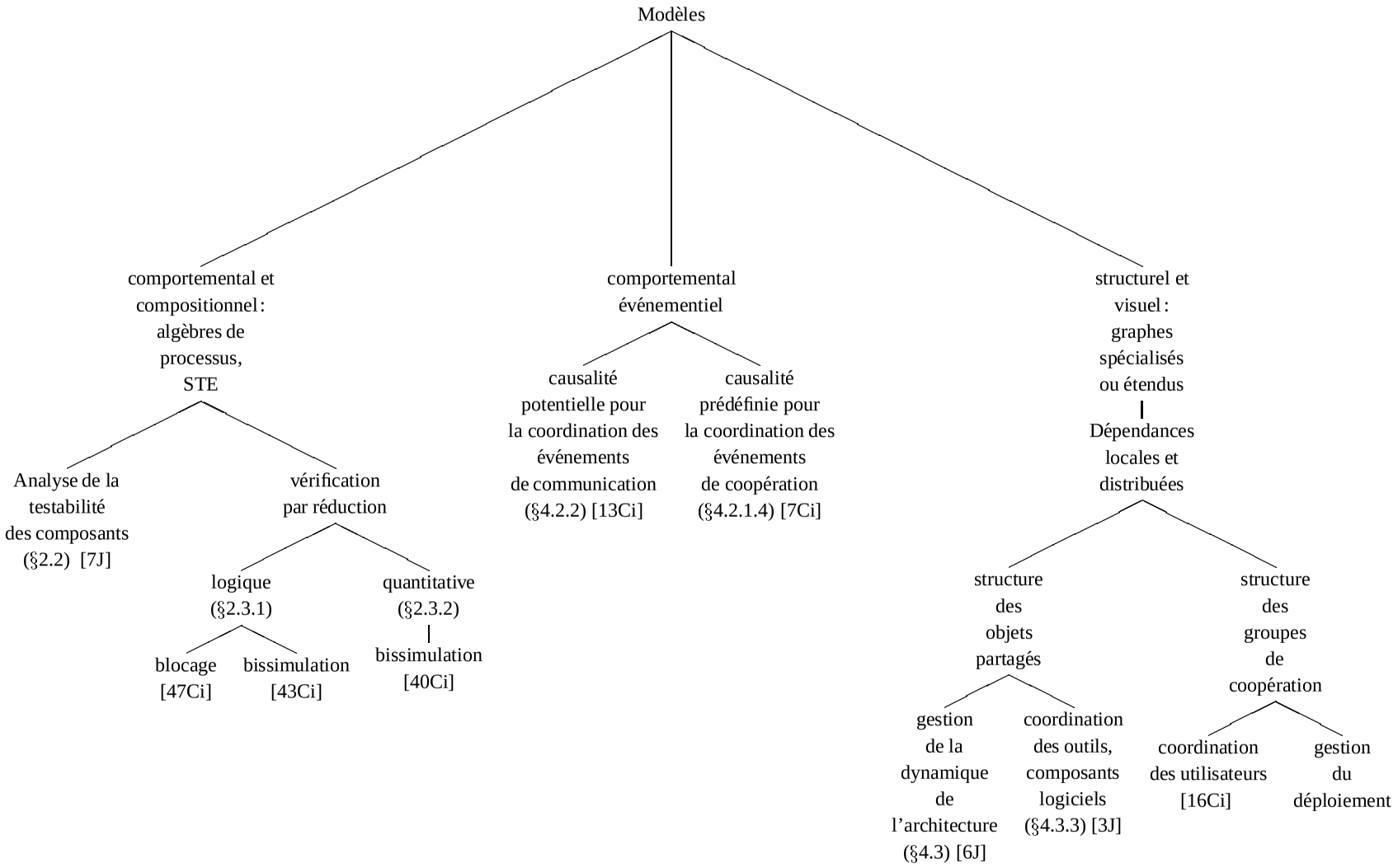
Le cadre scientifique général de cette perspective est formé par les protocoles de publication et découverte des ressources dans un environnement distribué.

5

Conclusion Générale et perspectives

5.1 Récapitulatif structuré des classes de modèles développés

Nous avons abordé les différents problèmes d'analyse et de conception en adoptant des approches orientées modèle, et ce pour les différents niveaux d'interaction étudiés. Cette orientation constitue une caractéristique importante de nos travaux. Le tableau 5.1 classe les différents modèles élaborés (dans le domaine des systèmes communicants et dans le domaine des systèmes distribués coopératifs) et les différents problèmes résolus grace à ces modèles en donnant des références vers une publication pour chaque catégorie.



TAB. 5.1 – *Table des catégories de modèles développés et leurs applications*

5.2 Conclusion et perspectives

Dans ce rapport nous avons présenté nos travaux de recherche depuis 1993 jusqu'à présent. Les thèmes de recherche principaux concernés par les travaux développés s'articulent autour des architectures et des protocoles de communication et de coordination pour les logiciels coopératifs distribués. Les contributions ont concerné la spécification, la modélisation, la conception, la validation, l'implémentation et l'expérimentation. L'adoption d'une approche orientée modèle constitue la caractéristique partagée par les différents résultats de cette recherche.

La gestion de la dynamique de l'architecture et des interactions distribuées et son application pour le support des activités de coopération génériques ont été deux axes majeurs dans le domaine des applications traitées. L'édition coopérative en groupe a constitué une activité générique que nous avons modélisée et soutenue par différentes versions d'outils de support. L'ingénierie distribuée a été le domaine d'expérimentation principal des modèles de coordination où la gestion de session était la principale fonction.

Le comportement des composants d'une architecture, qui peut se définir comme l'ensemble des règles qui décrivent la dynamique des interactions, a été au centre de nos travaux de recherche initiaux. Depuis quelques années, et pour compléter nos travaux initiaux, nous avons orienté nos recherches vers la gestion de la dynamique de la coordination et de la distribution des composants. Cette direction continue à être pour nous une perspective de recherche pour les prochaines années. Nous projetons de la consolider par des modèles plus développés, des outils et des services plus puissants et par des applications plus variées et plus importantes. L'adaptabilité sera pour nous une exigence que nous étudierons par la généralisation et l'extension de nos travaux sur la gestion de la dynamique. Les architectures logicielles dynamiques orientées composants et adaptables constituent le thème que nous comptons développer dans les prochaines années. La conception, la modélisation, l'implantation et la validation continuent à être des objectifs pour cette recherche.

L'axe des systèmes -informatiques en général et logiciels en particulier- de support des activités coopératives continue à être pour nous un domaine d'application important pour plusieurs raisons. D'une part parce qu'il constitue un domaine qui pourrait fortement influencer le déploiement et l'utilisation des réseaux d'ordinateurs et où il y a un effort de recherche international qui essaie d'anticiper et de maîtriser cette orientation. D'autre part, il y a une forte demande des utilisateurs pour des solutions dans des domaines d'activité qui, malgré leur différence, convergent vers la même problématique d'un point de vue recherche. Enfin les systèmes de support des activités coopératives introduisent de nombreux problèmes depuis la conception et jusqu'à l'implémentation, problèmes qui n'étaient pas présents dans les systèmes ciblés par nos travaux initiaux. Il s'agit notamment, de problèmes introduits par la complexité des schémas d'interaction dynamiques N-M tant des points de vue communication, coordination et coopération et des contraintes d'adaptabilité.

Ainsi, en particulier, nous allons développer :

- au niveau modèle, des solutions pour la gestion de l'évolution et le contrôle de la conformité des architectures dynamiques et le déploiement des composants dans un environnement P2P en général et dans un environnement coopératif plus particulièrement.

- au niveau architecture, un cadre générique avec une décomposition fonctionnelle qui permet de définir des solutions générales pour les architectures dynamiques adaptatives au niveau

middleware que l’on spécialisera dans le contexte des architectures distribuées coopératives orientées composant.

6

Bibliographie

[Abramsky, 1987] ABRAMSKY, S. (1987). « Observation equivalence as a testing equivalence ». *Theoretical Computer Science*, 53:225–241.

[Baldoni et al., 1997] BALDONI, R., FRIEDMAN, R., et van RENESSE, R. (1997). « The Hierarchical Daisy Architecture for Causal Delivery ».

[Beca et al., 1997] BECA, L., CHENG, G., FOX, G. C., JURGA, T., OLSZEWSKI, K., PODGORNY, M., SOKOLOWSKI, P., STACHOWIAK, T., et WALCZAK, K. (1997). « Tango : a Collaborative Environment for the World-Wide Web. Northeast Parallel Architectures Center, Syracuse University, New York, http://trurl.npac.syr.edu/tango/papers/tangowp.html ».

[Bird et Kasper, 1995] BIRD, D. et KASPER, M. (1995). « Problem Formalization Techniques for Collaborative Systems ». Dans *IEEE Transactions on Systems Man and Cybernetics, 25(2):231–242 6, February*.

[Birman, 1993] BIRMAN, K. (1993). « The Process Group Approach to Reliable Distributed Computing ». *Communications of the ACM*, 36(12):36–53.

[Birman et al., 1991] BIRMAN, K., SCHIPER, A., et STEPHENSON, P. (1991). « Lightweight Causal and Atomic Group Multicast ». *ACM Trans. Compt. Syst.*, 9(3):272–314.

[Bolognesi et Brinksma, 1988] BOLOGNESI, T. et BRINKSMA, E. (1988). « Introduction to the ISO Specification Langage LOTOS ». *Computer Networks and ISDN Systems*, 14(1):25–29.

[Bourdeau et Wasson, 1997] BOURDEAU, J. et WASSON, B. (1997). « Orchestrating collaboration in collaborative telelearning ». Dans *B. du Boulay and R. Mizoguchi (Eds.) Proceedings of the 8th World Conference on Artificial Intelligence in Education, 565-567. Amsterdam: IOS Press*, pages 565–567.

[Brinksma, 1988] BRINKSMA, E. (1988). A theory for the derivation of tests. Dans AGGRAWAL, S. et K.SABANI, éditeurs, *Protocol Specification Testing and Verification*, volume VIII. Elsevier Science Publishers B.V., North-Holland.

[Brinksma et al., 1987] BRINKSMA, E., SCOLLO, G., et STEENBERGEN, C. (1987). Lotos Specifications, their implementations and their tests. Dans SARIKAYA, B. et BOCHMANN, G., éditeurs, *Protocol Specification Testing and Verification*, volume VI. Elsevier Science Publishers B.V., North-Holland.

[Brna, 1998] BRNA, P. (1998). « Models of Collaboration ». Dans *Proceedings of the Workshop on Informatics in Education, XVIII Congresso Nacional da Sociedade Brasileira de Computaço Rumoa Sociedade do Conhecimento in Belo Horizone, Brazil (1998). http://cbl.leeds.ac.uk/ paul/papers/bcs98paper/bcs98.html#roschellesharedknowledge*.

[Chanbert et al., 1998] CHANBERT, A., GROSSMAN, E., JACKSON, L., et PIETROVICZ, S. (1998). « NCSA Habanero- Synchronous collaborative framework and environment, Software Development Division at the National Center for Supercomputing Applications, white paper. http://habanero.ncsa.uiuc.edu/habanero/ ».

[Coleman, 2002] COLEMAN, D. (2002). « Levels of Collaboration ». Rapport Technique, Collaborate New Letter. http://www.collaborate.com/publication/newsletter/publications_newsletter_march02.html.

[Conen et Neumann, 1998] CONEN, W. et NEUMANN, G. (1998). *Coordination Technology for Collaborative Applications. Organizations, Processes, and Agents. LNCS 1364*. Springer Verlag.

[Costantini et Toinard, 2001] COSTANTINI, F. et TOINARD, C. (2001). « Collaborative Learning with the Distributed Building Site Metaphor ». Dans *IEEE Multimedia juil-Sept 01.pp. 21-29.*

[Dillenbourg et al., 1995] DILLENBOURG, P., BAKER, M., BLAYE, A., et O'MALLEY, C. (1995). « The Evolution of Research on Collaborative Learning ». Dans *P. Reimann and H. Spada (Eds). Learning in humans and machines. Towards an interdisciplinary learning science, 189-211. London: Pergamon. (1995).*

[Dommel et Aceves, 1997] DOMMEL, H. et ACEVES, J. G. L. (1997). « Floor Control for multimedia conferencing and collaboration ». Dans *Multimedia Systems 5:23-38.*

[Dommel et Aceves., 1999] DOMMEL, H. P. et ACEVES., J. J. G. L. (1999). « Group Coordination Support for synchronous Internet collaboration ». Dans *IEEE Internet Computing, pp. 74-80, March-April.*

[Dommel et Garcia-Luna-Aceves, 2000] DOMMEL, H.-P. et GARCIA-LUNA-ACEVES, J. (2000). « Network Support for Group Coordination ». Dans *4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000), Orlando, FL.*

[Dssouli et Fournier, 1990] DSSOULI, R. et FOURNIER, R. (1990). « Communication Software Testability ». Dans *3rd International Workshop on Protocol Test Systems,Washington D.C., USA*. IFIP WG6.

[Durfee et al., 1985a] DURFEE, E., LESSER, V., et CORKILL, D. (1985a). « Coherent Cooperation Among Communicating Problem Solvers ». *IEEE Transactions on Computers*, C-36(11):1275–1291.

[Durfee et al., 1985b] DURFEE, E., LESSER, V., et CORKILL, D. (1985b). « Coherent Cooperation Among Communicating Problem Solvers ». Dans *IEEE Transactions on Computers C-36(11)*.

[Ellis et al., 1991] ELLIS, C., GIBBS, S., et REIN, G. (1991). « *Groupware. Some issues and experiences.* ». *Communications of the ACM*, 34(1):38–58.

[Ellis et al., 1996] ELLIS, W., HILLIARD, R., POON, P., RAYFORD, D., SAUNDERS, T., SHERLUND, B., et WADE, R. (1996). « Toward a recommended practice for architectural description ». Dans *Proceedings of 2nd IEEE International Conference on Engineering of Complex Computer Systems, Montreal, Quebec, Canada, October 21-25, 1996.*

[Finin et al., 1993] FININ, T., WEBER, J., WIEDERHOLD, G., GENESERETH, M., FRITZSON, R., MCGUIRE, J., SHAPIRO, S., et BECK, C. (1993). « Specification of the KQML Agent-Communication Language ». Rapport Technique, The Darpa Knowledge Sahring Initiative. http://www.cs.umbc.edu/kqml/papers/kqmlspec.pdf.

[Garcia et al., 2004] GARCIA, E., GUYENNET, H., LAPAYRE, J., et ZERHOUNI, N. (2004). « A new industrial cooperative tele-maintenance platform ». *Computers and Industrial Engineering*, 46:851864.

[Ghedamsi et al., 1992] GHEDAMSI, A., DSSOULI, R., et BOCHMANN, G. (1992). « Diagnostic Tests for Single Transition Faults in Non-Deterministic Finite State Machines ». Dans *5th International Workshop on Protocol Test Systems, Montreal Canada*. IFIP WG6.

[Hall et al., 1996] HALL, R., MATHUR, A., JAHANIAN, F., PARKASH, A., et RASSMUSSEN, C. (1996). « CORONA: A Communication Service for Scalable, Reliable Group Collaboration Systems ». Dans *International Conference on Computer Supported Cooperative Work, Cambridge , MA, USA, pp.140-149*.

[H.Krasner et al., 1991] H.KRASNER, J.MCINROY, et WALZ, D. (1991). « Groupware Research and Technology Issues with Application to Software Process Management ». Dans *IEEE Transactions on Systems Man and Cybernetics,21(4):704–712, July*.

[Hoare, 1985] HOARE, C. (1985). *Communicating Sequential Processes*. International series in computer science. Prentice-Hall, New York.

[Hopcroft et Ullman, 1989] HOPCROFT, J. et ULLMAN, J. (1989). *Introduction to Automata Theory, Languages and Computation*. Addison-Welsey.

[IETF,] IETF. « IMPP Information Homepage. URL: http://www.imppwg.org ».

[Isaac,] ISAAC. « ISAAC : Framework for Integrated Synchronous And Asynchronous Collaboration http://www.isrl.uiuc.edu/isaac/ ».

[IST-Proteus, 2003] IST-PROTEUS (2003). « A generic platform for e-maintenance ». http://www.proteus-iteaproject.com/.

[Jacinto, 1995] JACINTO, R. (1995). « *Protocoles de validation à deux phases dans les systèmes transactionnels répartis : spécification, modélisation et vérification* ». Doctorat num. 1938, Université Paul Sabatier, Toulouse. Rapport LAAS num. 95005.

[Lamport, 1978] LAMPORT, L. (1978). « Time, Clocks and the Ordering of Events in Distributed Systems ». *Communications of the ACM*, 21(7):558–565.

[Langerak, 1990] LANGERAK, R. (1990). « A testing theory for LOTOS using deadlock detection ». Dans *Protocol Specification, Testing and Verification*, volume IX. international IFIP WG6.1.

[Leduc, 1991] LEDUC, G. (1991). « A Framework based on implementation relations for implementing LOTOS specifications ». *Computer Networks & ISDN Systems*, 25(1):23–41.

[Leduc, 1995] LEDUC, G. (1995). « Failure-based Congruences, Unfair Divergences and New Testing Theory ». Dans VUONG, S. T. et CHANSON, S. T., éditeurs, *Protocol Specification, Testing and Verification XIV, Proceedings of the Fourteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification (PSTV), Vancouver, BC, Canada, 1994*, volume 1 de *IFIP Conference Proceedings*, pages 252–267. Chapman & Hall.

[Lesser et Corkill, 1981] LESSER, V. et CORKILL, D. (1981). « Functionally Accurate, Cooperative Distributed Systems ». Dans *IEEE Transactions on Systems Man and Cybernetics, 11(1):81–96, January*.

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

[Malone et Crwoston, 1994] MALONE, T. et CRWOSTON, K. (1994). « The interdisciplinary Study of Coordination ». *ACM computing Surveys*, 26(1):87–119.

[McCormack, 2001] MCCORMACK, K. (2001). « The degrees of collaboration, the drivers and visualization within the SCOR model ». Rapport Technique, http://www.drkresearch.org/Publications/The_degrees_of_collaboration_and_the_drivers.pdf.

[Michel, 1996] MICHEL, F. (1996). « *Validation de systèmes répartis. Symétries d'architecture et de données* ». Thèse de doctorat num. 1253, Institut National Polytechnique, Toulouse. (Rapport LAAS num. 96483).

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

[Milner, 1980] MILNER, R. (1980). A Calculus of Communicating Systems. Dans *Lecture Notes in Computer Science*, volume 92. Springer-Verlag, Berlin Heidelberg.

[Mol et Koppius, 2002] MOL, M. J. et KOPPIUS, O. R. (2002). « Information Technology and the Internationalization of the Firm ». Dans *Journal of Global Information Management, Vol. 10, No. 4*.

[Mostefaoui et Raynal., 1993] MOSTEFAOUI, A. et RAYNAL., M. (1993). « Causal Multicast in Overlapping Groups: Towards a Low Cost Approach ». Dans *4th IEEE Workshop on Future Trends of Distributed Systems. Lisboa, September*.

[Mostefaoui et Raynal, 1995] MOSTEFAOUI, A. et RAYNAL, M. (1995). « Causal multicast in overlapping groups towards a low cost approach ». Dans *IEEE Workshop on Future Trends of Distributed Computer Systems, pp 136-142, September*.

[Métayer, 1998] MÉTAYER, D. L. (1998). « Describing Software Architecture Styles Using Graph Grammars ». Dans *IEEE Transactions on Software Engineering, 24:7. July*.

[Nicola et Hennessy, 1984] NICOLA, R. D. et HENNESSY, M. (1984). « Testing equivalences for processes ». *Theoretical Computer Science*, 34:83–133.

[Omicini et al., 2001] OMICINI, A., ZAMBONELLI, F., KLUSCH, M., et TOLKSDORF, R., éditeurs (2001). *Coordination of Internet Agents: models, technologies, and applications. Springer Verlag. ISBN 3-540-41613-7*.

[Panitz, 1996] PANITZ, T. (1996). « A Definition of Collaborative vs Cooperative Learning. ». Rapport Technique, Deliberations/Collaborative Learning. 1996. http://www.lgu.ac.uk/deliberations/collab.learning/index.html.

[Pellegrini et Riveill, 1999] PELLEGRINI, M.-C. et RIVEILL, M. (1999). « Dynamic Architecture Management of Component based Applications ». Dans *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas (USA), pp.800-806, 28 June - 1st July*.

[Petrenko et al., 1993] PETRENKO, A., DSSOULI, R., et KOENIG, H. (1993). « On Evaluation of Testability of Protocol Structures ». Dans RAFIQ, O., éditeur, *Proc. 6th International IFIP Workshop on Protocol Test Systems- IWPTS VI*, Pau.

[Philippe Merle et al., 1998] PHILIPPE MERLE, C. G., ROOS, J.-F., et GEIB., J.-M. (1998). « CorbaScript: A Dedicated CORBA Scripting Language ». Dans *CHEP'98 Computing in High Energy Physics*, Chicago, Illinois, USA, August 31 - September 4.

[Phillips, 1987] PHILLIPS, I. (1987). « Refusal testing ». *Theoretical Computer Science*, 50:241–284.

[Poole et DeSanctis, 2003] POOLE, M. S. et DESANCTIS, G. (2003). « Structuration Theory in Information Systems Research: Methods and Controversies ». Dans *M. E. Whitman and A. B. Woszczynsli (Eds.), The handbook for information systems research. Idea Group Publishing*.

[Prakash et al., 1997] PRAKASH, R., RAYNAL, M., et SINGHAL, M. (1997). « An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments ». *Journal of Parallel and Distributed Computing*, pages 190–204.

[Presence et (prim),] PRESENCE, I. et (PRIM), I. M. P. « http://www.ietf.org/html.charters/prim-charter.html ».

[Raposo et al., 2000] RAPOSO, A. B., MAGALHES, L. P., et RICARTE, I. L. M. (2000). « Coordination Mechanisms for Collaborative Virtual Environments ». Dans *WRV'2000 Proceedings - III Workshop on Virtual Reality, Gramado, RS, october* , pages 277–278.

[Roca,] ROCA, V. « The MCL Multicast Library: Concepts, Architecture and Use ».

[Rodrigues et Verissimo, 1995] RODRIGUES, L. et VERISSIMO, P. (1995). « Causal Separators and Topological Timestamping: an Approach to Support Causal Multicast in Large-Scale Systems ». Dans *Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, British Columbia, Canada, May*.

[Roschelle, 1992] ROSCHELLE, J. (1992). « What should collaborative technology be?: A perspective from Dewey and situated learning ». *ACM SIGCUE Outlook (Spring)*, 21(3):39–42.

[Roschelle et Teasley, 1995] ROSCHELLE, J. et TEASLEY, S. (1995). « The construction of shared knowledge in collaborative problem solving ». Dans *O'Malley, C.E., (ed.), Computer Supported Collaborative Learning. pages 69–97. Springer-Verlag, Heidelberg.*

[Salber et al., 1995] SALBER, D., COUTAZ, J., DECOUCHANT, D., et RIVEILL, M. (1995). « De l'observabilité et de l'honnêteté : le cas du contrôle d'accès dans la Communication Homme-Homme Médiatisée ». Dans *Proceedings IHM'95, pp. 27-34, CEPAD*.

[Schmid et Simone, 1996] SCHMID, K. et SIMONE, C. (1996). « Coordination mechanisms:Towards a conceptual foundation of CSCW systems design ». *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 5(2-3):155–200. Special issue: The Design of Cooperative Systems, edited by The COOP Group.

[Schmidt, 2002] SCHMIDT, K. (2002). « Remarks on the complexity of cooperative work ». Dans *Hakim Benchekroun and Pascal Salembier (eds.): Cooperation and Complexity, RSTIA, Hermes, Paris 2002.*

[Schukmann et al., 1996] SCHUKMANN, C., KIRCHNER, L., SCHUMMER, J., et HAAKE, J. (1996). « Designing Object-oriented synchronous groupware with COAST ». Dans *International Conference on Computer Supported Cooperative Work, Cambridge , MA, USA, pp.30-38*.

[Schwarz et Mattern., 1994] SCHWARZ, R. et MATTERN., F. (1994). « Detecting Causal Relationships in Distributed Computations: in Search of the Holy Grail ». *Distributed Computing*, 7:149–174.

[Sinia, 2002] SINIA (2002). « an overview of CSCL (Computer Supported Collaborative Learning), which by many is regarded as an emerging paradigm of educational IT use (computers in education). ». Rapport Technique, http://www.uib.no/People/sinia/CSCL/index.html.

[Spector, 2001] SPECTOR, J. M. (2001). « Tools and principles for the design of collaborative learning environments for complex domains. ». *Journal of Structural learning and Intelligent Systems*, 14(4):483–510.

[Stenning et al., 2003] STENNING, K., GREENO, J. G., HALL, R., SOMMERFELD, M., et WIEBE, M. (2003). « Coordinating Mathematical with Biological Multiplication: Conceptual Learning

as the Development of Heterogeneous Reasoning Systems ». Dans *Brna, P. and Baker, M. and Stenning and K. and Tiberghien, A. (Eds). The Role of Communication in Learning to Model Lawrence Erlbaum Associates*. To appear.

[Tango,] TANGO. « TANGO Interactive (TM) : http://www.webwisdom.com/Technologies/collaboration.html, http://trurl.npac.syr.edu/handout/tango.html ».

[Texier, 2000] TEXIER, G. (2000). « Gestion des interactions implicites dans les environnements de travail coopératifs ». Dans *Thèse de l'Université de Rennes 1. September*.

[Texier et Plouzeau, 1999] TEXIER, G. et PLOUZEAU, N. (1999). « Automatic Management of Sessions in Shared Spaces ». Dans *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas (USA), Vol.IV, pp.2115-2121, 28 June - 1st July*.

[Tonino et al., 2002] TONINO, H., BOS, A., de WEERDT, M., et WITTEVEEN, C. (2002). « Plan coordination by revision in collective agent based systems ». *Artificial Intelligence*, (142):121–145.

[Tretmans, 1992] TRETMANS, J. (1992). « *A Formal Approach to Conformance Testing* ». PhD Thesis, University of Twente, The Netherlands.

[van Glabbeek, 1990] van GLABBEEK, R. (1990). « The linear Time-Branching Time Spectrum ». Dans *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458, Amsterdam, The Netherlands. Lecture Notes in Computer Science, Springer-Verlag.

[Vuong et al., 1993] VUONG, S., LOUREIRO, A., et CHANSON, S. (1993). « A Framework for the Design for Testability of Communication Protocols ». Dans [42], pages 297–312.

[Wasson, 1999] WASSON, B. (1999). « Design and Evaluation of a Collaborative Telelearning Activity aimed at Teacher Training ». Dans *Proceedings of the Computer Support for Collaborative Learning (CSCL) 1999 Conference, C. Hoadley and J. Roschelle (Eds.) Dec. 12-15, Stanford University, Palo Alto, California. Mahwah, NJ: Lawrence Erlbaum Associates.*

[Wegner, 1997] WEGNER, P. (1997). « The Paradigm Shift from Algorithms to Interaction ». *Communications of the ACM*.

[Wilde, 1997] WILDE, E. (1997). « Group and Session Management ». Dans *Thèse de doctorat ETH No. 12075, ETH Zurich. Shaker Verlag. ISBN 3-8265-2411-X*.

7

Liste des publications et travaux

7.1 Publications

Thèse

[1T] K. Drira. *Transformation et composition de graphes de refus : analyse de la testabilité*. Thèse de doctorat, Université Paul Sabatier, Toulouse, Octobre 1992. (Rapport LAAS num. 92435).

Articles et préfaces dans des revues scientifiques avec comité de lecture

[1J] S. Pomares Hernandez, J. Fanchon, K. Drira, et M. Diaz. Causal broadcast protocol for very large group communication systems. *Studia Informatica Universalis*, pages 175–188, 7/5 2003. (Éditions Suger) ISBN 2-912590-14-0 ISSN H.S. 1624-0065.

[2J] T. Villemur, K. Drira, V. Baudin, et M. Diaz. Services, methodologies and platforms for cooperative environments. *Studia Informatica Universalis*, pages 213–234, 7/5 2003. (Éditions Suger) ISBN 2-912590-14-0 ISSN H.S. 1624-0065.

[3J] E. Roblet, K. Drira, et M. Diaz. Formal design and development of a CORBA-based application for cooperative HTML group editing support. *Journal of Systems and Software*, (60):113–127, 2002. (Elsevier journal).

[4J] K. Drira, T. Villemur, V. Baudin, et M. Diaz. A design methodology applied to distance learning support software . *Interactive Learning Environments Journal*, 9(1):51–78, 2001. (Elsevier Science journal).

[5J] P. Gradit, K. Drira, et F. Vernadat. An integrated approach to coordination description in distributed multimedia applications. *Integrated Computer-Aided Engineering*, 8(4):311–324, 2001. (IOSPRESS journal).

[6J] K. Drira. A coordination middleware for collaborative component-oriented distributed applications. *Special Issue on information and communication middleware. NETNOMICS (Eco-*

nomic Research and Electronic Networking), 2(2000):85–99, 2000. (Baltzer Science Publishers journal).

[7J] K. Drira, P. Azéma, et P. De Saqui-Sannes. Testability Analysis in Communicating Systems. *Computer Networks*, 36:671–693, 2001. (Springer Verlag journal).

[8J] K. Drira et P. Azéma. Analyse de la testabilité des systèmes communicants avec traitement de la divergence. *Technique et Science Informatiques*, 15(8):1051–1078, 1996.

[9J] K. Drira et P. Azéma. Les graphes de refus pour la vérification de conformité et l'analyse de testabilité des protocoles de communication. *Réseaux et Informatique Répartie/ Electronic Journal on Networks and Distributed Processing*, pages 27–48, 1994. URL: http://rerir.univ-pau.fr/rerir.html.

[10J] F. Arbab et K. Drira. Introduction to the special Issue. *The Journal of Supercomputing*, 24(2):119–120, 2003. (Kluwer journal).

[11J] K. Drira. Editor's Preface. *The Journal of Information Technology Theory and Application (JITTA)*, 3(2), 2001. Electronic journal. URL: http://www.jitta.org/journal/volume3_2/preface.pdf.

Contributions à ouvrages

[1O] J. M. Molina Espinosa, J. Fanchon, et K. Drira. *The Internal-Local-Remote Dependency Model for Generic Coordination in Distributed Collaboration Sessions.*, volume 3061 de *Lecture Notes in Computer Science*, pages 158–169. Springer, 2004.

[2O] F. Moo-Mena et K. Drira. *A Component-Based Design Approach for Collaborative Distributed Systems.*, volume 3061 de *Lecture Notes in Computer Science*, pages 197–206. Springer, 2004.

[3O] S. Pomares Hernandez, K. Drira, et J. Fanchon. *The immediate dependency relation: an optimal way to ensure causal group communication*, page 16 pages (chapter 4). Annual Review of Scalable Computing (Volume 6). World Scientific, 2004. ISBN 981-238-902-4.

[4O] V. Baudin, K. Drira, T. Villemur, et S. Tazi. *A model-driven approach for synchronous dynamic collaborative e-learning*, Chapitre 3, pages 44–65. Idea Group, 2004. ISBN: 1-931777-92-6.

[5O] K. Drira, A. Martelli, et T. Villemur. *Chapter1: Introduction*, Chapitre 1, pages 3–6. Lecture Notes in Computer Science 2236. Springer, 2001. ISBN 3-540-43083-0.

[6O] K. Drira, J.M. Molina Espinosa, O. Nabuco, L.M. Rodriguez Peralta, et T. Villemur. *Product data and workflow management*, Chapitre 5, pages 107–151. Lecture Notes in Computer Science 2236. Springer, 2001. ISBN 3-540-43083-0.

[7O] K. Drira et M. Diaz. *Graph-grammar based coordination in inter-corporate computer supported collaborative activities*, Chapitre 1, pages 1–27. Annual Review of Scalable Computing (volume2). World Scientific Publishing, 2000. ISBN 981-02-4413-4.

Conférences internationales avec actes et comité de lecture

[1Ci] O. Nabuco, M.F. Koyama, F.E.D. Pereira, J.R. Silva, K. Drira, et J.M. Rosario. Manufacturing automation network's cooperative e-space. Dans *IFIP 18th World Computer Congress. TC5/WG5.5 - 5th Working Conference on Virtual Enterprises (PRO-VE) Virtual Enterprises and Collaborative Networks. pp. 443-450*, Toulouse (France), 22-27 August 2004. Kluwer Academic Publishers. ISBN 1-4020-8138-3.

[2Ci] O. Nabuco, J.R. Silva, J.M. Rosario, et K. Drira. Scientific collaboration and knowledge sharing in the virtual manufacturing network. Dans *IFIP 18th World Computer Congress. 1st IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI'2004), Artificial Intelligence Applications and Innovations. Vol.6, pp.117-128*, Toulouse (France), 22-27 August 2004. Kluwer Academic Publishers. ISBN 1-4020-8150-2.

[3Ci] I.A. Loulou, A.H. Kacem, M. Jmaiel, et K. Drira. Towards a unified graph-based framework for dynamic component-based architectures description in z. Dans *IEEE/ACS International Conference on Pervasive Services (ICPS'2004)*, pages 227–234, Beyrouth (Lebanon), juillet 19–23 2004. IEEE Computer Society.

[4Ci] O. Nabuco, M.F. Koyama, F.E.D. Pereira, et K. Drira. Finding manufacturing expertise using ontologies and cooperative agents. Dans *11th IFAC Symposium on Information Control Problem in Manufacturing (INCOM'2004), 6p.*, Salvador da Bahia (Brésil),, 5-7 April 2004.

[5Ci] K. Guennoun, K. Drira, et M. Diaz. A proved component-oriented approach for managing dynamic software architectures. Dans *7th IASTED International Conference on Software Engineering and Applications (SEA 2003)*, Marina del Rey, CA, USA, novembre 3–5 2003. ACTA PRESS. ISBN 0889863946.

[6Ci] O. Nabuco, M.F. Koyama, et K. Drira. Ontology based sharing of engineering knowledge: sheik tool. Dans *3rd IFIP Conference on e-commerce, e-business, and e-government (I3E'2003)*, Sao-Paulo (Brasil), 21-24 September 2003.

[7Ci] J.M. Molina-Espinosa, J. Fanchon, et K. Drira. A logical model for coordination rule classes in collaborative sessions. Dans *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03) June 9-11, Johannes Kepler University, Linz, Austria*. IEEE Computer Society, 2003. ISBN 0769519636.

[8Ci] S. Tazi, Y. Al Tawki, et K. Drira. Editing pedagogical intentions for document reuse. Dans Amine Benkiran, éditeur, *4th International Conference on Information Technology Based Higher Education and Training ITHET 2003 July 7-9, Marrakech, Morocco*. IEEE Computer Society, 2003. ISBN 9954835202.

[9Ci] L.M. Rodriguez Peralta, T. Villemur, K. Drira, et J.M. Molina-Espinosa. Managing dependencies in dynamic collaborations using coordination diagrams. Dans *6th International Conference on Principles of DIstributed Systems (OPODIS'02)*, pages 29–42, Reims (France), décembre 11–13 2002.

[10Ci] J.M. Molina Espinosa et K. Drira. A multi-modal coordination service model for cooperative distributed systems engineering. Dans *IEEE International Conference on Systems Man and Cybernetics (SMC'02)*, page 6p, Hammamet (Tunisia), 6-9 October 2002. IEEE Computer Society. ISBN 295123094X.

[11Ci] M. El Jed et K. Drira. A design framework for collaborative authoring environments applied to web using sedit. Dans *IEEE International Conference on Systems Man and Cybernetics (SMC'02)*, page 6p, Hammamet (Tunisia), 6-9 October 2002. IEEE Computer Society. ISBN 295123094X.

[12Ci] V. Baudin, K. Drira, T. Villemur, et M. Diaz. Supporting distributed experts in e-meetings for synchronous collaboration. Dans *IEEE International Conference on Systems Man and Cybernetics (SMC'02)*, page 6p, Hammamet (Tunisia), 6-9 October 2002. IEEE Computer Society. ISBN 295123094X.

[13Ci] S. Pomares Hernandez, K. Drira, J. Fanchon, et M. Diaz. An efficient multi-channel distributed coordination protocol for collaborative engineering activities. Dans *IEEE International Conference on Systems Man and Cybernetics (SMC'02)*, page 6p, Hammamet (Tunisia), 6-9 October 2002. IEEE Computer Society. ISBN 295123094X.

[14Ci] J.M. Molina Espinosa, K. Drira, et T. Villemur. The responsibility management system for collaborative meetings scheduling in the distributed system engineering project. Dans *IEEE International Workshop on Knowledge Media Networking (KMN'2002)*, pages 114–119, Kyoto (Japan), 10-12 July 2002. IEEE Computer Society. ISBN 0769517781.

[15Ci] J.M. Molina Espinosa, K. Drira, et L.M. Rodriguez Peralta. An event driven session management system for collaborative distributed systems engineering. Dans *9th European Concurrent Engineering Conference (ECEC'2002)*, pages 148–152, Modena (Italy), 15-17 Avril 2002. SCS Publications. ISBN 9077039066.

[16Ci] L.M. Rodriguez Peralta, T. Villemur, et K. Drira. An xml on-line session model based on graphs for synchronous cooperative groups. Dans *2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Las Vegas (USA), 25-28 Juin 2001, pp.1257-1263*. CSREA PRESS, 2001. Rapport LAAS 01101.

[17Ci] K. Drira, M. Diaz, T. Villemur, M. Jmaiel, A.E.M. Ben Hamadou, et A. Hadj Kacem. Cooperative systems for information sharing and exchange. Dans *IEEE 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises(WETICE'2001), Cambridge (USA), 20-22 Juin*. IEEE Computer Society, 2001. ISBN 0769517781.

[18Ci] O. Nabuco, K. Drira, et E. Dantas. A layered design model for knowledge and information sharing cooperative systems. Dans *IEEE 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises(WETICE'2001), Cambridge (USA), 20-22 Juin*. IEEE Computer Society, 2001. ISBN 0769517781.

[19Ci] J.M. Molina-Espinosa, O. Nabuco, et K. Drira. A uml model for session management in collaborative design for space activities. Dans *8th European Concurrent Engineering Conference (ECEC'2001), Valence (Espagne), 18-20 Avril 2001, pp.170-174*. SCS PUBLICATIONS, 2001. ISBN 9077039066.

[20Ci] O. Nabuco, K. Drira, et J.M. Rosario. Sharing engineering information and knowledge. contributions to pipefa's platform. Dans *International NAISO Congress on Information Science Innovations (ISI'2001), Dubai (EAU), 17-21 Mars*, pages 431–437. ICSC ACADEMIC PRESS, 2001. ISBN 3906454258.

[21Ci] V. Baudin, K. Drira, T. Villemur, et S.Tazi. Une approche synchrone pour une téléexpertise distribuée. Dans *1eres Journées Francophones des Modèles Formels de l'Interaction (MFI'01), Toulouse (France), 21-23 Mai*, pages 363–377, Vol.III, 2001.

[22Ci] S. Pomares Hernandez, K. Drira, et J. Fanchon. A reference design model for group communication systems. Dans *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, pages 1119–1125, Las Vegas (USA), 26-29 June 2000. CSREA PRESS. ISBN 1892512238.

[23Ci] L.M. Rodriguez-Peralta, T. Villemur, et K. Drira. An xml-based session model for synchronous cooperative groups. Dans *Workshop on Software Architectures for Cooperative Systems (CSCW'2000), Philadelphie (USA), 1st December*, page 4, 2000. http://www.cs.queensu.ca/ cscw2000/positionPapers.html.

[24Ci] K. Drira, T. Villemur, V. Baudin, et M. Diaz. A multi-paradigm layered architecture for synchronous distance learning. Dans *26th EUROMICRO Conference. Workshop on Multimedia and Telecommunications, Maastricht (Netherlands/Pays-Bas), 5-7 September*, pages 158–165, Vol.2. IEEE COMPUTER SOCIETY, 2000. ISBN : 0769507808.

[25Ci] K. Drira, S. Pomares-Hernandez, et M. Diaz. Design and assessment of a corba-based multi-tier architecture for groupware and networked multimedia applications control. Dans *International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000), Monastir (Tunisie), 22-24 Mars*, pages 25–30, 2000.

[26Ci] J.M. Molina-Espinosa, K. Drira, et M. Diaz. A graph-grammar model for coordination definition in workflow process activities. Dans *International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000), Monastir (Tunisie), 22-24 Mars*, pages 168–173, 2000.

[27Ci] T. Villemur et K. Drira. Proposition d'une methodologie pour la conception de services collaboratifs. Dans *8eme Colloque Francophone sur l'Ingénierie des Protocoles*

(CFIP'2000), Toulouse (France), 17-20 Octobre 2000 Ingenierie des protocoles. Qualite de service, multimedia et mobilite, pages 429–444. Hermes, 2000. ISBN 2-7462-0177-1.

[28Ci] T. Villemur et K. Drira. A methodology for the design of collaborative services. Dans *Conference on Software: Theory and Practice*, pages 592–599, Beijing (China), 21-25 August 2000.

[29Ci] T. Villemur, K. Drira, et M. Diaz. Design of a group membership service on top of a distributed java actor platform. Dans *7th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'99), 20-22 December 1999*, pages 119–124, Cape Town (South Africa/Afrique du Sud), 1999. IEEE Computer Society. ISBN 076950468X.

[30Ci] K. Drira, F. Gouezec, et M. Diaz. Design and implementation of coordination protocols for distributed cooperating objects. A general graph-based technique applied to CORBA. Dans P. Ciancarini, A. Fantechi, et R. Gorrieri, éditeurs, *Third IFIP International Conference on Formal Methods for Open Object-based Distributed Systems*, pages 89–104, Florence, Italy, February 15-18, 1999. Kluwer Academic Publishers. ISBN 0-7923-8429-6.

[31Ci] K. Drira, F. Gouezec, et M. Diaz. A Cooperation service for CORBA objects. From the model to the applications. Dans P Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, et D. Ruiz, éditeurs, *5th European conference on parallel computing (EUROPAR'99)*, volume 1685 de *Lecture Notes in Computer Science*, pages 769–776, Toulouse France, 31 August-3 September, 1999. Springer. ISBN N3-540-66443-2.

[32Ci] K. Drira, B. Berthomieu, et M. Diaz. An Approach to the Synthesis of Coordination in Distributed Cooperative Applications. Dans *CSCWD'99 - Computer Supported Cooperative Work in Design*, September 29-October 1, Compiègne, France, 1999.

[33Ci] P. Owezarski, D. Wartelle, P. Perrot, G. Ségarra, S. Guillouet, K. Drira, A. Meftah, et M. Diaz. Assessment methodology of new technologies for collaborative automotive design. Dans *2d Int. Distributed Conference on Network Interoperability*, pages 1–16, Funchal 16-18 june, 1997.

[34Ci] P. Gradit, F. Vernadat, et K. Drira. La réécriture de graphes pour la vérification d'algorithmes distribués. Dans *1er Colloque International sur les NOuvelles TEchnologies de la REpartition (NOTERE'97)*, pages 35–49, Université de Pau 4-6 Novembre 1997, 1997.

[35Ci] R. Jacinto-Montes, G. Juanole, et K. Drira. A two-phase commit protocol for distributed transactions. Dans *4th Int. Symposium on Applied Corporate Computing*, pages 61–70, Mexico October 30-November 1, 1996.

[36Ci] K. Drira, P. Gradit, et F. Vernadat. Graph-based coordination of cooperative agents. Dans O. Spaniol, C. Linnhoff-Popien, et B. Meyer, éditeurs, *Industrial and Short Paper- International workshop on Trends in Distributed Systems'96*, pages 133–146, Aachen, Germany, octobre 1996. Verlag der Augustinus Buchhandlung. (ISBN 3-86073-473-3).

[37Ci] F. Vernadat, K. Drira, et P. Azéma. An integrated description technique for distributed cooperative applications. Dans *IMACS Multiconference on Computational Engineering in*

Systems Applications (CESA'96), Workshop on Petri Nets For Multi-Agent Systems and Groupware, pages 608–613, Lille-France July 9-12, 1996.

[38Ci] F. Michel, P. Azéma, et K. Drira. Selective generation of symmetrical test cases. Dans B. Baumgarten, HJ. Burkhardt, et A. Giessler, éditeurs, *Testing of Communicating Systems*, IFIP, pages 191–206, Darmstadt, septembre 1996. Chapman & Hall.

[39Ci] F. Vernadat, P. Azéma, et K. Drira. Distributed Coin Tossing. Dans *14th International Conference on Distributed Computing Systems*, pages 244–249, Poznan, Poland, juin 21–24 1994. IEEE Computer Society Press.

[40Ci] K. Drira, Y. Atamna, et G. Juanole. Quantified reduced views of state graphs using Markovian and timed observational equivalence. Dans P. Dembinski et M. Sredniawa, éditeurs, *Protocol Specification, Testing and Verification XV*, IFIP, pages 253–268, Warsaw, Poland, 1995. Chapman & Hall.

[41Ci] M. Diaz, K. Drira, A. Lozes, et Ch. Chassot. On the definition and representation of the quality of service for multimedia systems. Dans R. Puigjaner, éditeur, *High Performance Networking VI*, IFIP, pages 116–128, Palma, 1995. Chapman & Hall.

[42Ci] K. Drira. The refusal graph a Tradeoff between Verification and Test (INVITED PAPER). Dans O. Rafiq, éditeur, *Protocol Test Systems VI*, volume C-19 de *IFIP Transactions*, pages 297–312, Pau, France, septembre 1993. North Holland.

[43Ci] K. Drira et P. Azéma. Verifying communication protocols via testing-projection. Dans T. Rus M. Nivat, C. Rattray et G. Scollo, éditeurs, *Proceedings of the Third International Conference on Algebraic Methodology and Software Technology*, Workshops in Computing, pages 255–264, Twente, The Netherlands, juin 1993. Springer-Verlag.

[44Ci] R. Jacinto, G. Juanole, et K. Drira. On the application to OSI-TP of a Structured Analysis and Modelling Methodology Based on Petri Nets Models. Dans *Proceedings of the 4th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, Lisboa, Portugal, September 22-24 1993. IEEE Computer Society Press.

[45Ci] K. Drira, P. Azéma, B. Soulas, et A.M. Chemali. Testability of a communicating system through an environment. Dans M.C. Gaudel et J.P. Jouannaud, éditeurs, *TAPSOFT'93: Theory and Practice of Software Development.*, volume 668 de *Lecture Notes in Computer Science*, pages 529–543, Orsay, France, 1993. Springer-Verlag.

[46Ci] K. Drira, P. Azéma, B. Soulas, et A-M. Chemali. A Formal Assessment of Synchronous Testability for Communicating Systems. Dans *13th International Conference on Distributed Computing Systems*, pages 149–156, Pittsburgh, Pennsylvania, May 25-28 1993. IEEE Computer Society Press.

[47Ci] K. Drira, P. Azéma, et F. Vernadat. Refusal graphs for conformance tester generation and simplification: a computational framework. Dans A. Danthine, G. Leduc, et P. Wolper, éditeurs, *Protocol Specification Testing and Verification XIII*, volume C-16 de *IFIP Transactions*, pages 257–272, Liège, Belgium, 1993. North-Holland.

[48Ci] P. de Saqui Sannes, K. Drira, J.P. Courtiat, et P. Azéma. Séquences de test et testeurs canoniques dérivables de spécifications Estelle* : une expérience avec le protocole MMS. Dans R. Dssouli et G.V. Bochmann, éditeurs, *CFIP93 Ingénierie des Protocoles*, Montreal, 1993. Hermès.

[49Ci] K. Drira, P. Azéma, B. Soulas, et A.M. Chemali. Characterizing and ordering errors detected by conformance testing. Dans *Proc. 5th International Workshop on Protocol Test Systems. IWPTS'92*, Montreal, 28-30 September 1992. IFIP WG6, North Holland.

[50Ci] P. Azéma, K. Drira, et F. Vernadat. A bus instrumentation protocol specified in lotos. Dans Juan Quemada, José A. Mañas, et Enrique Vázquez, éditeurs, *Formal Description Techniques, III, Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, FORTE '90, Madrid, Spain, 5-8 November 1990*. North-Holland, 1991.

[51Ci] P. Azéma et K. Drira. Bus Communication Protocol for Remote Reading and Writing. Formal Specification and Verification. Dans *Proc. 16th Annual Conference of IEEE Industrial Electronics Society. IECON '90*, Asilomar, California, November 27-30 1990. IEEE Computer Society.

Conférences nationales avec actes et comités de lecture

[1Cn] J. Fanchon, K. Drira, et S. Pomares Hernandez. Abstract channels as connectors for software components in group communication services. Dans *Mexican International Conference on Computer Science (ENC 2004)*, page 12, Colima,, Mexico, septembre 20–24 2004.

[2Cn] I.A. Loulou, A.H. Kacem, M. Jmaiel, et K. Drira. Approche intégrée pour la spécification des architectures dynamiques orientées composants. Dans *Maghrebian Conference on Software Engineering and Artificial Intelligence (MCSEAI'2004)*, pages 125–136, Sousse (Tunisia), 9-12 May 2004. Centre de Publications Universitaire. ISBN 9973371933.

[3Cn] K. Guennoun, K. Drira, et M. Diaz. Une approche orientée modèle pour la gestion des architectures logicielles distribuées dynamiques. Dans *Maghrebian Conference on Software Engineering and Artificial Intelligence (MCSEAI'2004)*, pages 149–160, Sousse (Tunisia), 9-12 May 2004. Centre de Publications Universitaire. ISBN 9973371933.

[4Cn] I. Loulou, A. Hadj-Kacem, M. Jmaiel, et K. Drira. Spécification et vérification formelles des architectures dynamiques des systèmes orientés composants. Dans *Journées Scientifiques Francophones (JSF'03)*, pages 237–244, Tozeur (Tunisie), décembre 20–22 2003.

[5Cn] M. Hadj-Kacem, M. Jmaiel, A. Hadj-Kacem, et K. Drira. Un environnement pour l'administration des applications distribuées à base de composants. Dans *Journées Scientifiques Francophones (JSF'03)*, pages 245–251, Tozeur (Tunisie), décembre 20–22 2003.

[6Cn] M. Hadj Kacem, M. Jmaiel, A. Hadj Kacem, et K. Drira. Vers un environnement générique pour l'administration des applications coopératives distribuées orientées composants. Dans

3emes Journées Scientifiques des Jeunes Chercheurs en Génie Electrique et Informatique (GEI'2003), page 12p, Mahdia (Tunisie), 18-20 Mars 2003.

[7Cn] S. Khemakhem, M. Jmaiel, A. Benhamadou, et K. Drira. Un environnement de recherche et d'intégration de composant logiciel. Dans *Maghrebian Conference on Software Engineering and Artificial Intelligence (MCSEAI'2002)*, Annaba (Algeria), 4-6 May 2002.

[8Cn] L. M. Rodriguez Peralta, T. Villemur, et K. Drira. Un modèle de Session XML basé sur des graphes pour des groupes coopératifs synchronous. Dans *JDIR 2002, Toulouse France,Mars*, 2002.

[9Cn] J.M. Molina Espinosa, K. Drira, et M. Diaz. Modele de description de procédures workflow basé sur la réécriture de graphes. Dans *Journées Formalisation des Activités Concurrentes (FAC'2000)*, pages 105–115, Toulouse (France), 18-19 Mai 2000.

[10Cn] K. Drira, F. Vernadat, et P. Gradit. Formalisation des services coopératifs. Dans *Journées de Recherche sur le Contrôle Réparti dans les Applications Coopératives*, pages 11–16, Université Pierre et Marie Curie - Paris, 30 et 31 mai, 1996.

[11Cn] F. Michel, P. Azéma, et K. Drira. Optimisation du test de la conformité par symétrie. Dans *Actes de RENPAR'8*, pages 141–144, Bordeaux, France, mai 1996.

[12Cn] J.M. Molina Espinosa, K. Drira, et M. Diaz. Modelo de descripcion de coordinacion de actividades en procesos workflow. Dans *Taller de Sistemas Distribuidos y Paralelos del Segundo Encuentro Nacional de Computacion*, page 6, Pachuca (Mexique), 12-15 Septembre 1999.

Manifestations internationales sans actes et avec comités de lecture

[1M] K. Drira. The diamond-approach for agent graph transformation. Dans *Planning for Open Networked Multimedia Systems (AAAI PONMS'99)*, 1999.

[2M] K. Drira. Concurrent Communicating Agents for communication protocols modelling. Dans *EISD'98*, Guadalajara/Mexico, 24-28 Dec, 1998. Invited tutorial.

[3M] K. Drira. Coordinated Collaborative Agents for distributed collaborative applications design and implementation. Dans *EISD'98*, Guadalajara/Mexico, 24-28 Dec, 1998. Invited tutorial.

[4M] K. Drira, F. Gouezec, et M. Diaz. Towards a coordination service for distributed cooperative objects. Dans *DISC'98*, Andros/Greece, October, 1998. Poster Session.

Rapports de contrat

[1Rc] T. Villemur, V. Baudin, K. Drira, et S. Tazi. Tele-expertise distribuee. LAAS REPORT 02040, LAAS, 2002. Projet Region TEDI N. 99008957.

[2Rc] B. Baurens, B. Cambou, K. Drira, J.M. Molina Espinosa, et O. Nabuco. Dse v1 integrated implementation report. LAAS REPORT 01280, LAAS, 2001. Delivrable Project IST-1999-10302.

[3Rc] T. Villemur et K. Drira. Nouveaux services méthodologies et environnements pour le travail coopératif distribué. LAAS REPORT 99152, LAAS, 1999. Programme Telecom du CNRS, Projet TL 97028,.

[4Rc] K. Drira, L.M. Rodriguez Peralta, O. Nabuco, J.M. Molina Espinosa, et S. Pomares Hernandez. Dse v1 architecture and technical specifications. LAAS REPORT 00612, LAAS, 2000. Delivrable Project IST-1999-10302.

[5Rc] S. Letailleur, D. Wartelle, P. Perrot, P. Owezarski, et K. Drira. Deliverable 4: trials synthesi. LAAS REPORT 98191, LAAS-CNRS, Mars 1998. Projet Esprit CANET N EP22528.

[6Rc] D. Wartelle, G. Segarra, S. Guillouet, K. Drira, P. Owezarski, et P. Perrot. Deliverable 1: Definition of business scenarios and assessment methodology. LAAS REPORT 97231, LAAS-CNRS, Juin 1997. Projet Esprit CANET N EP22528.

[7Rc] S. Lejay, K. Drira, et P. Azéma. Logiciels Pour l'Analyse de la Testabilité des Systèmes Communicants. LAAS REPORT 93349, LAAS-CNRS, octobre 1993. Rapport du contrat LAAS-EDF (65 pages).

[8Rc] K. Drira et P. Azéma. Graphes de refus pour l'analyse de la testabilité d'un système à travers un environnement. LAAS REPORT 91401, LAAS-CNRS, Décembre 1991. Rapport du contrat EDF-LAAS (34 pages).

[9Rc] K. Drira, F. Michel, et P. Azéma. Analyse de la testabilité, extensions aux cas de divergence et à l'équivalence observationnelle. LAAS REPORT 94029, LAAS-CNRS, Décembre 1994. Rapport du contrat EDF-LAAS (18 pages).

[10Rc] K. Drira et P. Azéma. Le test de conformité à travers un environnement. LAAS REPORT 90290, LAAS-CNRS, Septembre 1990. Rapport du contrat EDF-LAAS.

Rapports de recherche

[1Rr] J.M. Molina-Espinosa et K. Drira. A formal representation of the coordination requirements for the collaborative verification activity within the dse project. LAAS REPORT 03028, LAAS, 2003.

[2Rr] J.M. Molina-Espinosa, K. Drira, et T. Villemur. The responsibility management system for collaborative meetings scheduling in the distributed system engineering project. LAAS REPORT 02356, LAAS, 2002.

[3Rr] T. Villemur, L.M. Rodriguez Peralta, et K. Drira. Conception d'un service de gestion de session coordonnee. LAAS REPORT 02042, LAAS, 2002.

[4Rr] L.M. Rodriguez Peralta, T. Villemur, et K. Drira. Design of a coordinated session management service. LAAS REPORT 01501, LAAS, 2001.

[5Rr] S. Pomares Hernandez, J. Fanchon, K. Drira, et M. Diaz. An adaptable causal message ordering protocol for very large group communication systems. LAAS REPORT 01280, LAAS, 2001.

[6Rr] S. Pomares Hernandez, K. Drira, et M. Diaz. A framework for dynamic distributed group coordination protocols. LAAS REPORT 01011, LAAS, 2001.

[7Rr] T. Villemur, V. Baudin, K. Drira, M. Diaz, X. Guangyou, et S. Yuanchun. A comparative framework for synchronous distance learning environments. LAAS REPORT N01096, LAAS-CNRS, juillet 2001.

[8Rr] L.M. Rodriguez Peralta, K. Drira, et T. Villemur. An xml architecture for a synchronous session model. LAAS REPORT 00506, LAAS, 2000.

[9Rr] K. Drira, T. Villemur, V. Baudin, et M. Diaz. A multi-paradigm layered architecture for distance learning using the Web. LAAS REPORT N99496, LAAS-CNRS, Novembre 1999.

[10Rr] F. Michel, K. Drira, et M. Diaz. Sequential-parallel representation, extension and coverage for partial orders. LAAS REPORT 95004, LAAS-CNRS, janvier 1995. (29 pages).

[11Rr] K. Drira et Pierre Azéma. Decision procedures for failure-based implementation relations. LAAS REPORT 93011, LAAS-CNRS, Mars 1993. 12 pages.

[12Rr] K. Drira. Les graphes de refus : une structure adéquate pour la vérification et l'analyse des protocoles de communication. LAAS REPORT 93106, LAAS-CNRS, Mars 1993. 11 pages.

[13Rr] K. Drira, F. Vernadat, et P. Azéma. Lotos design and implementation of a distributed agreement algorithm. LAAS REPORT 91091, LAAS-CNRS, Mars 1991.

Revues de vulgarisation

[1V] K. Drira. Internet pour les entreprises : enjeux et technologies. *La Revue de l'Entreprise*, 6(33):70–72, Janvier-Fevrier 1998.

[2V] K. Drira. Le réseau Internet et les infrastructures informatiques. Architecture, besoins et orientations. *La Revue de l'Entreprise*, 7(34):64, Mars-Avril 1998.

[3V] K. Drira et Pierre Azéma. Une aide à la validation de systèmes de communication. *La Lettre du LAAS*, Janvier 1995.

Edition d'Ouvrage: Livres, Revues, Actes

[1E] F. Arbab et K. Drira. Special Issue on Coordination in Component-Oriented Systems. Architecture, Models, Languages, and Applications. *The Journal of Supercomputing*, 24(2), 2003. (Kluwer journal).

[2E] K. Drira, A. Martelli, et T. Villemur, éditeurs. Lecture Notes in Computer Science 2236. Springer, 2001. ISBN 3-540-43083-0.

[3E] K. Drira. (eds). *The Journal of Information Technology Theory and Application (JITTA)*, 3(2), 2001. Electronic journal. URL: http://www.jitta.org.

[4E] H. Arabnia et al. With F. Arbab, R. Menezes, éditeur. *Special session on Coordination in Parallel and Distributed Applications and Activities*. Las Vegas (USA), 25-28 Juin 2001.

[5E] H. Arabnia et al. With F. Arbab, R. Menezes, éditeur. *Special session on Coordination in Parallel and Distributed Applications and Activities*. Las Vegas (USA), 26-29 June 2000.

[6E] H. Arabnia et al. With R. Jacinto Montes, éditeur. *Special session on Coordination in Parallel and Distributed Applications and Activities*. Las Vegas (USA), 28 june/01 july 2000.

7.2 Exposés

Exposés sur invitation

[1Xi] Les problèmes de coordination dans les logiciels de support pour les activités distribuées coopératives. Tozeur (Tunisie), décembre 20–22 2003. Journées Scientifiques Francophones (JSF'03).

[2Xi] Software support for distributed system engineering. Bauro (Brazil), septembre 14–17 2003. 6th Brazilian Symposium on Intelligent Automation.

[3Xi] Coordination et environnement de télé-ingénierie. LAAS-CNRS, 29 avril 2003. Visite ST MICROELECTRONICS.

[4Xi] Middleware et coordination. Hôtel Palladia/Toulouse, 13-14 mars 2003. Journées du Pôle Systèmes Informatiques Critiques.

[5Xi] Composants et services de coopération. LAAS-CNRS, 20 juin 2002. Journée Bilan Prospective à 3 ans.

[6Xi] Architectures des logiciels coopératifs distribués. La Marsa/Tunisie, avril 2001. Ecole Polytechnique de Tunis.

[7Xi] Présentation de corba/architecture, programmation et services. LAAS-CNRS, 02 février 2000. Demi journée CORBA/Club des Affiliés du LAAS.

[8Xi] Modèles de coordination pour les logiciels coopératifs distribués. Sfax/Tunisie, avril 2000. Faculté des Sciences Economiques et de Gestion de Sfax (FSEGS).

[9Xi] Les techonologies composants pour les applications distribuées sur internet. Sfax/Tunisie, avril 1999. Ecole Nationale d'Ingénieurs de Sfax (ENIS).

[10Xi] Graph-based coordination models for collaborative activities. Guadalajara (Mexico), novembre 1998. Ecole d'Hivers Systèmes Distribués (EISD)- ITESO/CINVESTAV.

[11Xi] Formal description of communicating systems. Guadalajara (Mexico), novembre 1998. Ecole d'Hivers Systèmes Distribués (EISD)- ITESO/CINVESTAV.

[12Xi] The refusal graph: a tradeoff between verification and test. invited presentation. Pau/France, septembre 1993. Dans : 6th International IFIP Workshop on Protocol Test Systems (IWPTS VI).

[13Xi] Les graphes de refus pour l'analyse de la testabilité à travers un environnement. Les Renardières, décembre 1991. EDF/DER.

[14Xi] Le test de conformité à travers un environnement. Les Renardières, septembre 1990. EDF/DER.

Exposés sur propositions acceptées (présentations des papiers aux conférences listées ci-avant)

[1Xp] A multi-modal coordination service model for cooperative distributed systems engineering. Hammamet/Tunisie, octobre 2002. Dans : 2002 IEEE International Conference on Systems Man and Cybernetics (SMC'02).

[2Xp] Cooperative systems for information sharing and exchange. Cambridge/Etats Unis, juin 2001. Dans : IEEE 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2001).

[3Xp] Sharing engineering information and knowledge. contributions to pipefa's platform. Dubai/UAE, mars 2001. Dans : International NAISO Congress on Information Science Innovations (ISI'2001).

[4Xp] Formal design and development of a corba-based application for cooperative html group editing support. Monastir/Tunisie, mars 2000. Dans : International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000).

[5Xp] A cooperation service for corba objects. from the model to the applications. Toulouse/France, septembre 1999. Dans : 5th International Euro-Par Conference (Euro-Par'99).

[6Xp] Design and implementation of coordination protocols for distributed cooperating objects. a general graph-based technique applied to corba. Florence/Italie, février 1999. Dans : IFIP Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS).

[7Xp] The agent graph coordination approach for the design of distributed cooperative systems. Compiègne/France, septembre–octobre 1999. Dans : international conference of Computer Supported Cooperative Work in Design 99 (CSCWD'99).

[8Xp] The diamond-approach for agent graph transformation. North Falmouth, Massachusetts /Etats Unis, novembre 1999. Dans : the AAAI 1999 Fall Symposium Series/ Modal and Temporal Logics Based Planning for Open Networked Multimedia Systems symp.

[9Xp] A cooperation service for distributed cooperative applications. Andros/Greece, septembre24-26 1998. 12th International Symposium on DIStributed Computing, DISC'98.

[10Xp] Graph-based coordination of cooperative agents. Aachen/Germany, octobre 1996. Dans : Workshop on Trends in Distributed Systems (TreDS'96).

[11Xp] Quantified reduced views of state graphs using markovian and timed observational equivalence. Warsaw/Poland, juin 1995. Dans : 15th International Symposium on Protocol Specification, Testing and Verification (PSTV'95).

[12Xp] Verifying communication protocols via testing-projection. Enschede/Netherlands, juin 1993. Dans : 3rd International Conference on Algebraic Methodology and Software Technology.

[13Xp] A formal assessment of synchronous testability for communicating systems. Pittsburgh/Etats Unis, mai 1993. Dans : 13th International Conference on Distributed Computing Systems (ICDCS'93).

[14Xp] Testability of a system through an environment. Orsay/France, avril 1993. Dans : 4th International Joint Conference CAAP/FASE TAPSOFT'93.

[15Xp] Characterizing and ordering errors detected by conformance testing. Montreal/Canada, septembre 1992. Dans : 5th International Workshop on Protocol Test Systems (IWPTS'92).

7.3 Liste des co-auteurs

Y. Al Tawki [1], F. Arbab [2], Y. Atamna [3], P. Azéma, V. Baudin, B. Baurens[4], A.E.M. Ben Hamadou [5], B. Berthomieu, B. Cambou , A.M. Chemali[6], J.P. Courtiat, E. Dantas [7], P. de Saqui-Sannes, M. Diaz, M. El Jed, J. Fanchon, F. Gouezec, P. Gradit, K. Guennoun, S. Guillouet[8], M. Hadj

1. Université de Thamar, Sanaa, Yémen
2. CWI Amsterdam Pays Bas
3. SRI International, Etats Unis (ex-LAAS)
4. Silogic, Toulouse
5. ISIMSF Sfax Tunisie
6. EDF-DER Les Renardières
7. ITI Campinas, Brésil
8. RENAULT SA, Boulogne Billancourt

Kacem[9], A. Hadj Kacem[10], R. Jacinto-Montes[11], M. Jmaiel[12], G. Juanole, S. Khemakhem[13] M. Koyama[14], S. Lejay, S. Letailleur[15], I. Loulou[16] A. Lozes , A. Martelli[17], A. Meftah[18], F. Michel, J.M. Molina-Espinosa[19], F.J. Moo-Mena, O. Nabuco[20], P. Owezarski, F.E.D. Pereira[21] P. Perrot[22], S. Pomares Hernandez[23], E. Roblet, L.M. Rodriguez Peralta, J.M. Rosario[24], G. Ségarra[25], J.R. Silva[26], B. Soulas[27], S. Tazi, F. Vernadat, T. Villemur, D. Wartelle[28]

7.4 Liste des Thèses, stages de DEA et PFE encadrés

7.4.1 Encadrement de Thèses de Doctorat d'Université

- F. Moo Mena (depuis sept 2003) Conception darchitectures orientées composants pour les logiciels coopératifs distribués. Thèse de Doctorat INP Toulouse.
  ((80%) En co-encadrement avec Michel Diaz)

- Karim Guennoun (depuis sept 2002)
  Gestion des architectures Dynamiques des logiciels distribués Thèse de Doctorat UPS Toulouse
  ((80%)En co-encadrement avec Michel Diaz)

- Saul Pomares (1998-2002)
  Services de coordination et protocoles de diffusion causale pour les applications coopératives distribuées. Thèse de Doctorat INP Toulouse
  ((80%) En co-encadrement avec Michel Diaz)

- Martin Molina (1999-2003)
  Modèle et services pour la coordination des sessions coopératives multi-applications: application à l'ingénierie système distribuée. Thèse de Doctorat INP Toulouse
  ((80%) En co-encadrement avec Michel Diaz)

9. FSEGS, Sfax Tunisie
10. FSEGS, Sfax Tunisie
11. Cinvestave, Guadalajara, Mexique (ex-LAAS)
12. ENIS, Sfax Tunisie
13. ISET, Sfax Tunisie
14. CENPRA Campinas, Brésil
15. RENAULT SA, Boulogne Billancourt
16. LARIS,FSEGS, Sfax Tunisie
17. Alenia Spazio, Turin, Italie
18. FRANCE TELECOM EXPERTEL
19. ITESM-CCM, México, Mexique (ex-LAAS)
20. ITI/CENPRA Campinas, Brésil (ex-LAAS)
21. CENPRA Campinas, Brésil
22. FRANCE TELECOM EXPERTEL
23. INAOE, Puebla, Mexique (ex-LAAS)
24. Université de Campinas, Brésil
25. RENAULT SA, Boulogne Billancourt
26. Université Saõ Paulo (Brésil)
27. EDF-DER Les Renardières
28. SIEMENS Automative, Toulouse

– Olga Nabuco (jan 2000 à juin 2001).
Applications de partage de connaissances dans un environnement de production. PhD Univ Campinas Brésil.
((50%) En co-encadrement avec Joao Mauricio Rosario)

7.4.2 Encadrement de DEA

– Rony Ghostine DEA Systèmes Informatiques UPS (Juin 2004).
Conception et implantation d'un algorithme de transformation de graphes.

– Francisco Moo Mena DEA Programmation Systèmes INPT (Juin 2003).
Conception des Architectures orientées composants pour les logiciels coopératifs distribués.
Publications co-signées : [2O]

– Mehdi Eljed DEA Systèmes Informatiques UPS (Juin 2002)
Environnements de coordination pour l'édition coopérative distribuée.
Publications co-signées : [11Ci]

– Olfa Jenhani DEA Systèmes Informatiques UPS (Juin 2002)
Gestion des sessions pour les activités coopératives distribuées.

– Adil Hbabi DEA Systèmes Informatiques UPS (Juin 2001).
Protocoles de coordination pour les objest coopératifs distribués.
((50%) En co-encadrement avec Thierry Villemur)

– Etienne Roblet DEA Programmation Systèmes UPS (Juin 2001)
Publications co-signées : [3J]
Environnements de coordination pour les espaces de travail partagés.

– JM Molina Espinosa DEA Programmation Systèmes INPT (Septembre 1999):
Modèles Workflow pour la description des procédures de coordination
Publications co-signées : [26Ci]

– Frédéric Gouëzec DEA Informatique Fondamentale et Parallélisme INPT (septembre 1998)
Conception et implantation de protocoles de coordination pour les objets coopératifs distribués
Publications co-signées : [31Ci]

– Laurent Ostiz DEA Informatique Fondamentale et Parallélisme INPT (septembre 1997)
Modélisation de l'activité de communication pour le travail de groupe hiérarchisé

7.4.3 Encadrement de Thèses CNAM

– Béatrice Bastier Cambou Thèse CNAM (2000-2001).
Préparation des sessions pour les activités coopérativess distribuées.
Publications co-signées : [2Rc]

7.4.4 Encadrement de PFE

– Kamal Essajidi PFE lENSA-Agadir (Juin 2004) :
Conception et mise en oeuvre d'un système d'édition et d'annotation coopératives distribuées.
((50%) En co-encadrement avec Said Tazi)

– Mehdi Eljed PFE INSA-Tunis (Janvier 2002) :
Environnements de coordination pour l'édition coopérative distribuée.

– Olfa Jenhani PFE INSA-Tunis (Janvier 2002)
Gestion des sessions pour les activités coopératives distribuées.

– Etienne Roblet PFE INT (Juin 2000) :
Environnements de coordination pour les espaces de travail partagés.

– Frédéric Gouëzec PFE ENSEEIHT (juin 1998)
Conception et implantation de protocoles de coordination pour les objets coopératifs distribués.

– Noureddine ERRAFAY PFE ENSEEIHT (Juin 1997)
Modélisation et simulation de la production et l'échange de documents structurés.

– Mounir Sidhom PFE section spéciale ENSICA (Juin 1996),
Conception et réalisation d'une interface utilisateur pour un éditeur de texte coopératif

– Stéphane Lejay, PFE ENSEEIHT: (Juin 1995),
Analyse de la testabilité des systèmes
((80%) En co-encadrement avec Pierre Azéma)

– Slimane Essaoudi PFE INSA (Septembre 1993) :
Vérification et test de systèmes communicants dans le cadre d'une conception d'un distributeur automatique par les techniques de description formelle
((60%) En co-encadrement avec Pierre Azéma)

– Nizar Saadi PFE ENSEEIHT (Juin 1992) :
Conception assistée par ordinateur des tests de conformité du protocole du bus de terrain industriel FIP.
((80%) En co-encadrement avec Pierre Azéma)

– Marc Buttigieg stage CNAM (Juin 1991) :
Le réseau Flux-Information-Process.

– Corine Hublet PFE Université N.D. de la Paix de Namur, Belgique (Juin 1990):
Sur la base d'une double spécification -Réseaux de Petri/Lotos- de FIP.
((20%) En co-encadrement avec Pierre Azéma)

7.5 Projets de Coopération

CMCU 2003(en cours d'évaluation) GIIP (Gestion Interopérable d'Informations Partagées).
Appel d'offre 2003 du Programme de recherches CMCU franco-tunisien.
Partenaires : LAAS-CNRS, LABRI, Unversité de Sfax.
Contribution particulière : co-responsable scientifique LAAS (avec Michel Diaz), élaboration du projet.

CAPES/COFECUB Oct 2003- Modèles de Coopération, Coordination et Communication appliqués au cas d'une Chaîne de Fournisseurs.
Appel d'offre 2003 du Programme de recherches CAPES/COFECUB.
Partenaires : LAAS-CNRS, Université de Campinas (Brésil), Unversité de So Paulo (Brésil).
Contribution particulière : responsable scientifique LAAS (avec Robert Valette), élaboration du projet.

ECOS-Nord Sept 1999- Sept 2001 COAACT : COordination des Applications et Activités Coopératives distTribuées
Appel d'offre 1999 du Programme de recherches ECOS-Nord.
Partenaires : LAAS-CNRS et CINVESTAV Guadalajara (Mexique).
Contribution particulière : co-responsable scientifique LAAS (avec Michel Diaz), élaboration du projet.

CNRS/DGRST 2001 Services de partage d'information.
Appel d'offre 2001 du Programme franco-tunisien CNRS/DGRST
Partenaires : LAAS-CNRS et Université de Sfax
Contribution particulière : responsable scientifique LAAS (avec Michel Diaz), élaboration du projet.

PRO Sept 2000- Dec 2001 Services CORBA pour la Coordination des Activités Distribuées.
Appel d'offre 1999 du Programme franco-chinois de recherches avancées (PRA)
Partenaires : LAAS-CNRS et Tsinghua University de Pékin.
Contribution particulière : co-responsable scientifique LAAS (avec Michel Diaz), élaboration du projet. [7Rr]

DSE Jan 1999- Dec 2001 Projet européen IST.
Partenaires: EADS-LV (F), Alenia Spazio (I), SIA (I), D3Group (G), IABG (G), LIP6 (F),LAAS (F), Silogic (F).
Objectifs: Concevoir un environnement de support pour l'ingénierie système distribuée coopérative dans le domaine de l'industrie spatiale.
Contribution particulière : responsable scientifique et administratif CNRS, participation à l' élaboration du projet. [5O], [2Rc] [4Rc]

CNRS-Telecoms 1997- 1999 Action CNRS COOACT TL 97028 (Appel d'offre CNRS-Télécoms)
Objectifs: Définir un cadre de conception formelle basé sur deux modèles comportementaux et structurels pour la conception de services de coopération génériques.

Contribution particulière : co-responsable scientifique LAAS (avec Thierry Villemur), participation à l'élaboration du projet [3Rc].

CANET Sept 1996- Jan 1998 Projet européen HPCN (High Performance Computing Networks).
Objectifs : coopération par des outils CSCW impliquant des constructeurs automobiles et des fournisseurs de composants sur réseau haut-débit ATM.
Partenaires : SIEMENS-FRANCE/ SIEMENS-GERMANY/ RENAULT/ RENAULT-SPAIN/ FRANCE TELECOM-EXPERTEL/ LAAS-CNRS.
Contribution particulière : Le coordinateur de la tâche de formalisation du scénario de coopération [33Ci],[6Rc],[5Rc].

CESAME 1992-1995 Projet de coopération CNET-CNRS. Conception formelle des systèmes hauts débits multimédias coopératifs.
Contributions particulières : Participation à l'activité de formalisation de la qualité de service. [41Ci].

DER-EDF 1990-1994 Etude de techniques de validation de systèmes de communication et d'analyse de la testabilité des systèmes partiellement accessibles
Contributions particulières :
- Elaboration d'une technique d'analyse de la testabilité à partir de spécification formelle
- Génération automatique de séquences de test. [48Ci].

LOTOSPHERE 1990-1992 European Project (Esprit) . This project concerned techniques and tools for the specification language LOTOS of ISO.
Contributions particulières :
- specification en LOTOS FieldBus protocols (FIP) [50Ci].
- représentant du LAAS à la réunion plénière de Vilamoura, Portugal October 1990.
- participation à l'organisation du workshop de Toulouse en Mars 1992.

7.6 Activités d'enseignement

7.6.1 Liste des enseignements montés (M) ou dispensés (D)

– INSA-DGEI (DUT+3) SFO (Cours/TD) [M] : Structure et Fonctionnement des Ordinateurs. 1991,1992

– IUT-DGE AP: Algorithmique et Programmation. 1994, 1995, 1996

– INSA-DGEI (4ème Année AEI) RdP (TP) [D] : Réseaux de Petri (Petri Nets). 1994, 1995

– INSA-DGEI (4ème Année OGI) ROPC (TP) [D] : Réseaux d'Ordinateurs et Protocoles de Communication. 1994, 1995, 1996, 1997

– INSA-DGEI&DGMM (4ème Année OGI, 5ème Année) CPO (Cours/TP) [M] : Conception et Programmation Objets 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004

- INSA-DGEI (et LAAS-CNRS&UPS&ENSCIA) (DEA Réseaux Télécoms et DEA et Systèmes Informatiques et 5ème Année R&T) TDF (Cours/TP) [M] : Techniques de Description Formelle/Algèbre de Processus. 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004.
- INSA-DGEI (et ENSEEIHT&UPS&ENSCIA) (DEA Réseaux Télécoms et 5ème Année OGI puis R&T) : (Cours/TP/Projets) [M] : Architecture des Systèmes Coopératifs Distribués/ Programmation Distribuée sur Internet (INSA, ENSHEEIT and ENSICA). 1999,2000,2001, 2002, 2003, 2004.
- ENSICA (DEA Réseaux Télécoms et 3ème Année) CSD (Cours) [M] : Coordination des Systèmes Distribués. 1999, 2001, 2002, 2003, 2004.
- FSEGS (Tunisie) (DEA SINT) FTC (Cours) [M] : Fondements de la Technologie des Composants 1999, 2001, 2002, 2003, 2004.
- ENIS (Tunisie) (DEA NTSID) (Cours) [M] : Modèles de Coordination des Systèmes Distribués 2001, 2002, 2003, 2004.

7.6.2 Liste des polycopiés produits

- Techniques de Description Formelle/Algèbres de Processus (20 pages).
- Programmation orientée Objets en C++ (60 pages) (Utilisé, ensuite avec mon accord, pour l'enseignement de cette matière à l'INSA en DGEI et DGMM).
- Architecture des Systèmes Coopératifs Distribués (Concepts généraux, CORBA, RMI, Services de localisation des objets) (80pages)
- Fondements de la technologie des composants (60 pages)
- Modèles de Coordination des Systèmes Distribués (20pages)

8

Annexe1: Description de l'environnement de gestion des architectures coopératives en CORBA

Ce chapitre annexe décrit le travail de l'implantation faite dans le cadre des projets de DEA de Frédéric Gouëzec, Etienne Roblet et Adel Hbabi.

8.1 Description des Éléments de l'architecture

8.1.1 Architecture générale d'une application

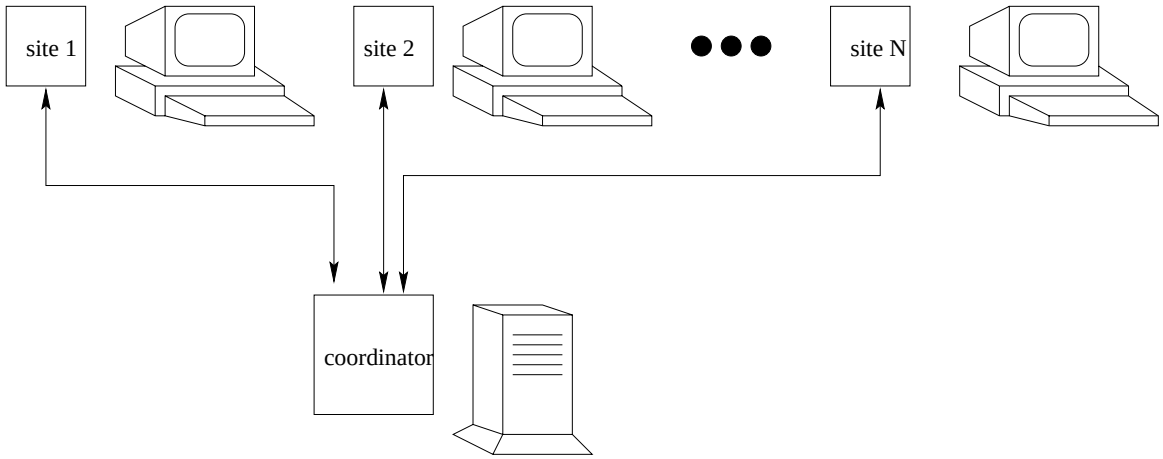


FIG. 8.1 – *Architecture générale d'une application*

L'architecture du service CORBA proposé permet au programmeur de développer, selon la méthodologie présentée au chapitre 4, des applications *standalone* ou sous forme d'*applets* exécutés au sein de *browsers* WWW . La communication étant relayée par le coordinateur qui peut être placé sur le serveur WWW, nous sommes en conformité avec les exigences de sécurité imposées aux *applets* par les *browsers* : un *applet* ne peut établir de connexion qu'avec le serveur WWW d'où il a été téléchargé.

Comme le montre la figure 8.1, un coordinateur qui s'exécute sur la même machine que le serveur Web, autorise la communication entre les objets coopératifs en utilisant des connexions indirectes.

8.1.2 Architecture d'un site

8.1.2.1 Les composants de l'application : Les objets coopératifs

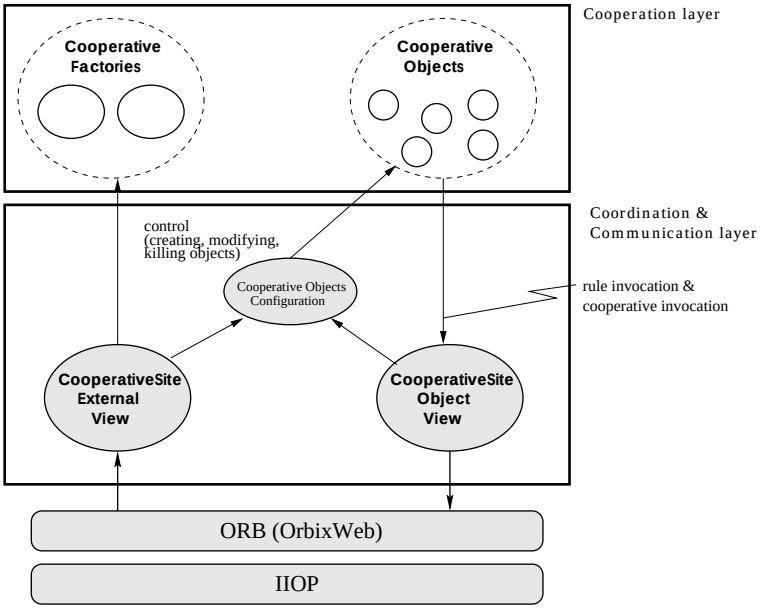


FIG. 8.2 – *Architecture d'un site*

Un objet Un objet coopératif est un objet CORBA. Il implante une interface IDL *CooperativeObject*. Un objet coopératif est désigné par un numéro de groupe materialisé par l'attribut *groupId*. On trouve sur chaque site autant de groupes que d'objets, un objet représentant un groupe. Le comportement d'un objet est déterminé par la valeur de l'attribut *typeId*. Les valeurs de ces deux attributs sont fixées à la création des objets et ne changent pas jusqu'à leur destruction.

La méthode *initialize()* est appelée une fois l'objet créé et ses attributs initialisés. Quant à la méthode *kill()*, elle est invoquée juste avant que le site ne supprime l'objet du bus logiciel.

L'échange d'information coopérative passe par l'emploi de la méthode *method*. Un objet peut demander l'invocation de cette méthode sur n'importe quel autre objet. Le paramètre *methodIndex* permet de réaliser un premier démultiplexage simulant l'existence de plusieurs méthodes coopératives et non d'une seule. Le second paramètre de type *Any* laisse à l'utilisateur le choix du type des données que les objets s'échangent.

Un objet possède deux paramètres. Le premier est le *groupParameter*. Sa valeur est partagée par l'ensemble des objets appartenant à un même groupe. Cette valeur est déterminée lors d'une modification du graphe de coordination. Le second, *ownParameter*, est particulier à chaque objet. Sa valeur est fixée par l'objet lui-même. La fonction de ce dernier paramètre est de permettre aux objets de constituer d'autres groupes différents du groupe définis à la création des objets.

Un autre attribut initialisé à la création de l'objet, *siteRef*, lui donne l'accès à deux services nécessaires à son activité : l'invocation de méthodes coopératives et l'invocation de règles de coordination.

```
interface CooperativeObject
  {
    void initialize();

    void kill();

    oneway void method(in MethodIdType methodIndex,
                       in any methodParameter);

    void setGroupParameter(in any p);
    // paramètre commun à tous les objets coopératifs d'un
    // même groupe

    readonly attribute SimpleParameter ownParameter;
    // paramètre spécifique à chaque objet

    attribute unsigned long groupId;
    // numéro de groupe auquel appartient l'objet

    attribute unsigned long typeId;
    // type de l'objet

    attribute CooperativeSiteObjectView siteRef;
    // référence sur le site où l'objet a été crée
  };
```

Cooperative Factory Pour créer des objets coopératifs, chaque site fait appel à des *factories*. Une *factory* n'a à sa charge la création d'objets de même type. Les objets coopératifs qui ont le même type ont un comportement identique, c'est à dire qu'ils sont implantés par le même code. Une *factory* d'objets coopératifs est un objet CORBA qui implante l'interface IDL *CooperativeFactory*. L'attribut *typeId* a pour valeur le type des objets coopératifs générés.

```
interface CooperativeFactory
  {
    readonly attribute unsigned long typeId;

    CooperativeObject createNewInstance();

    void deleteInstance(in CooperativeObject obj);
  };
```

8.1.2.2 Site coopératif

Un site est constitué d'un ensemble d'objets CORBA. Chaque objet a pour rôle d'offrir des services à des clients bien particuliers.

Service aux objets coopératifs L'objet CSOV d'interface IDL *CooperativeSiteObjectView* offre des services à destination des objets coopératifs uniquement. Ces services sont l'invocation de méthodes coopératives sur des objets coopératifs ou l'invocation de règles de coordination.

```
interface CooperativeSiteObjectView
  {
    attribute long siteId;

    // cooperative level
    void invoke(in CooperativeAddress addr,
                in MethodIdType methodName,
                in any methodParameter);

    void locallyInvoke(in long groupId,
                       in MethodIdType methodName,
                       in any methodParemeter);

    // coordination level
    boolean apply(in RuleIdType ruleIndex, in any ruleParameter);

    CooperativeObjectConfiguration getConfiguration();
  };
```

Pour provoquer l'invocation de la méthode *method* sur un ensemble d'objets coopératifs, un objet fait appel au CSOV par l'intermédiaire de la méthode *invoke*. L'ensemble des objets concernés par cette invocation est défini par le premier paramètre de la méthode *addr* de type *CooperativeAddress*.

```
struct CooperativeAddress
{
  communication::Router::SiteAddress siteAddr;
  message::ObjectLocalAddress objectAddr;
  ConditionIdType conditionId;
  any conditionArgument;
};
```

L'adresse est constituée d'un ensemble d'attributs qui forme un adressage à trois niveaux. L'adressage des sites est le premier niveau. La sous-adresse *siteAddr* désigne les sites dont les objets coopératifs sont susceptibles d'être concernés par l'information coopérative. Le second niveau concerne les objets présents sur un site. La sous-adresse *objectAddr* détermine les objets visés.

On trouve au troisième niveau deux attributs relatifs à une condition de filtrage des objets. Une condition est un objet dont l'interface IDL est *ConditionFunction* et qui comporte une seule méthode qui prend un paramètre du même type que l'attribut *ownParameter* d'un objet coopératif et retourne une valeur de type booléen.

Lorsqu'un site reçoit une demande d'invocation de nature coopérative, il filtre tout d'abord les objets présents sur son site avec l'adresse *objectAddr*, puis applique sur l'attribut *ownParameter* des objets retenus la condition. Si le résultat est positif pour un objet alors sa méthode *method* est exécutée.

```
interface ConditionFunction
  {
    void setConditionArgument(in any conditionParameter);

    boolean conditionIsVerified(in SimpleParameter ownParameter);
  };
```

L'ensemble des fonctions conditions est géré sur chaque site par l'objet d'interface IDL *ConditionManager*. C'est à cet objet que le site s'adresse pour obtenir une référence sur une fonction condition.

```
interface ConditionManager
  {
    ConditionFunction getConditionFunction(in ConditionIdType cid);
  };
```

Services externes Ce service se charge de traiter les messages arrivant sur le site en provenance du coordinateur. Ces messages sont aussi bien de nature coopérative que coordinatrice. Un message de coopération est traduit en invocations sur les méthodes *method* des objets coopératifs. Dans le cas d'un message de coordination, le service exécute les commandes qui composent le message afin de modifier la configuration d'objets coopératifs. Sur chaque site, un objet implantant l'interface IDL *CooperativeSiteExternalView* assure ce service.

```
interface CooperativeSiteExternalView : communication::Site
  {
    attribute long siteId;

    void go(in message::CMMMessage mess);
  };
```

Services à l'application Sur chaque site se trouve un objet CORBA implantant l'interface IDL *CooperativeSiteApplicationView* ; cet objet est appelé CSAV. Pour
construire un site, le programme principal doit créer un tel objet. L'exécutable poursuit en déclarant au CSAV les *factories* d'objets coopératifs qu'il sera susceptible d'utiliser. Cette déclaration est faite par l'appel de la méthode *addFactory*. Le programme doit aussi lui donner la référence

sur un objet CORBA implantant l'interface IDL *ConditionManager* en invoquant la méthode *setConditionManager*. Une fois ces initialisations effectuées, il reste à lancer l'activité coopérative du site en faisant appel à la méthode *connect* du CSAV. Lors de l'appel, on précise la session utilisée.

L'objet *CooperativeSiteApplicationView* se charge ensuite de créer les deux objets implantant les interfaces *CooperativeSiteObjectView* et *CooperativeSiteExternalView*.

```
interface CooperativeSiteApplicationView
  {
    readonly attribute long siteId;

    void addFactory(in CooperativeFactory factory);

    void setConditionManager(in ConditionManager condManager);

    boolean connect(in ccc::Session s, in string siteName);

    void disconnect();
  };
```

8.1.3 Architecture du coordinateur

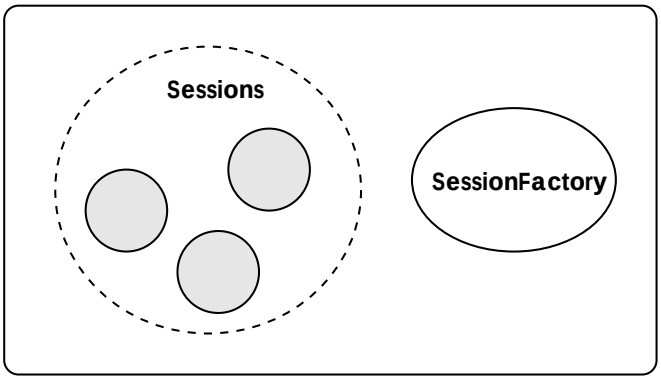


FIG. 8.3 – *Architecture du coordinateur*

8.1.3.1 Session Factory

Le *Session Factory* permet de créer différentes sessions ; une session correspondant à un service de coopération. Pour que des sites puissent s'échanger des informations coopératives et partager la même coordination, ils doivent utiliser la même session.

```
interface SessionFactory
  {
    Session createNewSession(in string sessionName);

    void deleteSession(in Session s);
```

```
};
```

8.1.3.2 Session

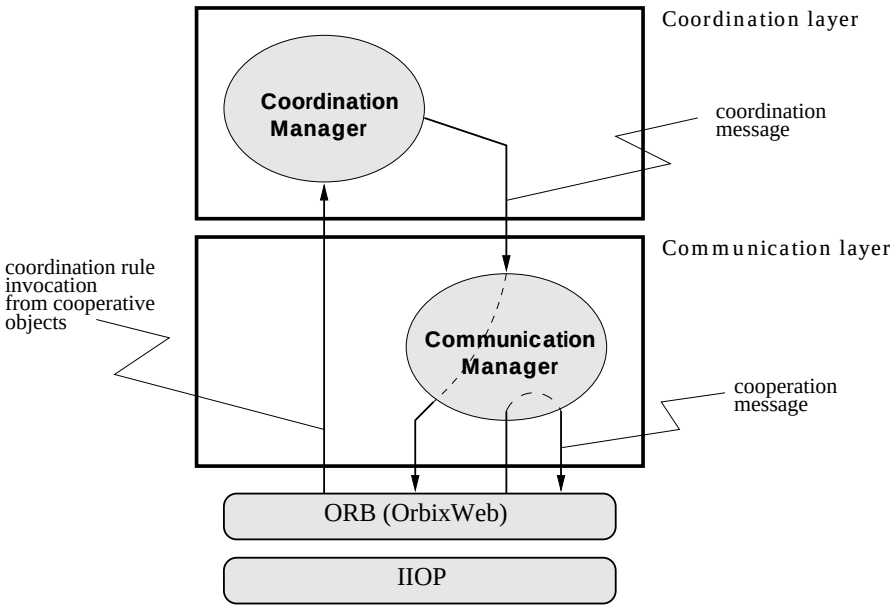


FIG. 8.4 – *Architecture d'une session*

Dans la définition IDL d'une session, l'attribut *router* est le *Communication Manager* de la session et l'attribut *grm* le *Coordination Manager*.

```
interface Session
  {
    readonly attribute communication::Router router;

    readonly attribute coordination::GraphRewritingModule grm;
  };
```

Communication Manager Le *Coordination Manager* implante l'interface IDL *Router*. Il assure un service de communication entre les différents sites connectés. Pour cela il maintient dans une liste pour chaque site un identificateur et une référence sur l'objet CSEV (CooperativeSiteExternalView, cf 8.1.2.2) du site.

Coordination Manager Le *Coordination Manager* est un objet CORBA implantant l'interface IDL *GraphRewritingModule*. Il est en charge de la gestion du médium de coordination implanté par le graphe de coordination accessible par les règles de coordination.

Le déclenchement d'une règle est provoqué par l'invocation de la méthode *applyRule* ou *applyRuleWithAck* de cet objet en lui donnant pour arguments l'identificateur de la règle et son paramètre. Lorsque la règle est applicable, son application conduit à une réécriture du graphe de

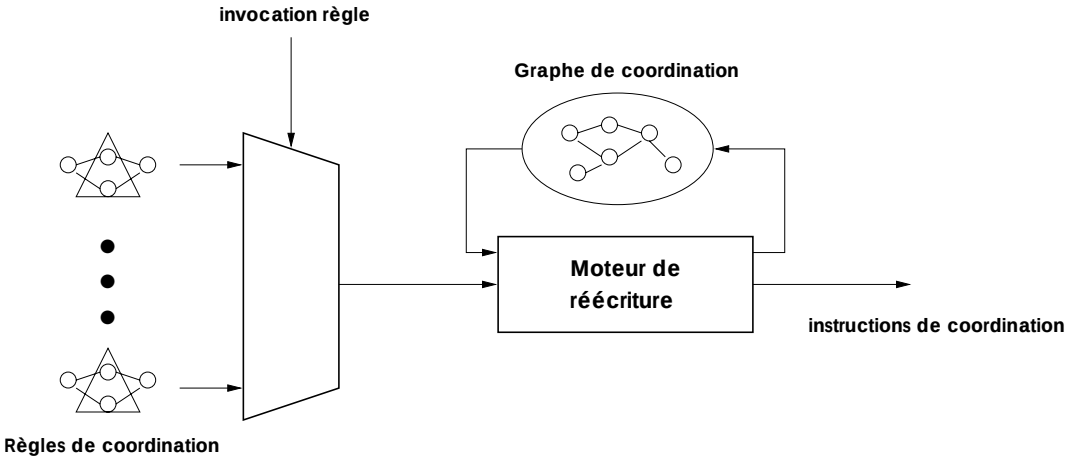


FIG. 8.5 – *Comportement du Coordination Manager*

coordination et à la génération d'actions coordinatrices qui sont passées au *Communication Manager* pour les envoyer à tous les sites connectés.

```
interface GraphRewritingModule
{
  message::CMMMessage getObjects();

  oneway void applyRule(in short ruleNumber, in any ruleParameter);

  boolean applyRuleWithAck(in short ruleNumber, in any ruleParameter);
};
```

8.1.4 Interoperabilité

Le principal intérêt d'avoir défini des interfaces IDL pour les objets coopératifs réside dans l'interopérabilité que procure CORBA. Ainsi la partie coopérative de l'application (les objets coopératifs) peut être programmée dans un langage différent de celui avec lequel le service de coordination a été réalisé. La figure 8.6 illustre cette possibilité. Le service de coordination est réalisé en Java et les objets coopératifs en C++. Deux *brokers* CORBA y sont utilisés : ORBacus pour C++ et OrbixWeb pour Java.

8.2 Description de l'implantation

8.2.1 Introduction

L'architecture CORBA que nous proposons pour les applications coopératives distribuées est constituée de deux types d'objets : des objets dont le comportement est spécifique à une application particulière et des objets dont le comportement est commun à toutes les applications.

Cette distinction nous permet de proposer une implantation des objets communs, ne laissant ainsi à la tache du programmeur que la réalisation des objets spécifiques à l'application qu'il dé-

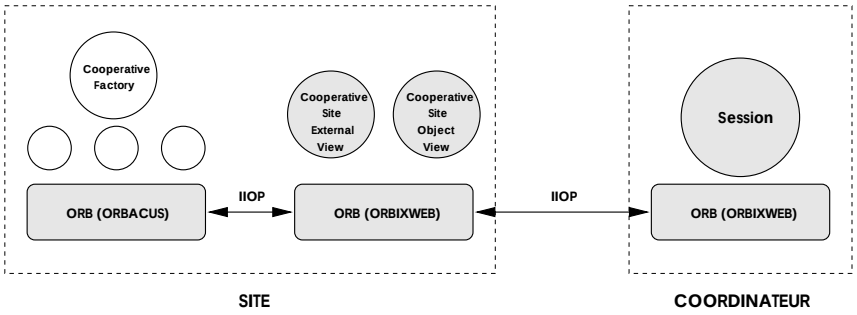


FIG. 8.6 – *Interoperabilité de l'architecture*

veloppe. Nous avons donc implanté dans le langage Java en utilisant le *broker* CORBA OrbixWeb ces parties communes de l'architecture. L'ensemble de classes Java ainsi conçues forme l'environnement logiciel dont nous accompagnons notre technique de coordination. Il incarne un support de réalisation exploitable par les programmeurs utilisant notre architecture. Le chapitre suivant présente un exemple de sa mise en œuvre pour la réalisation d'une application.

Dans la suite, nous traitons d'abord de la réalisation de la partie site puis celle de la partie coordinateur.

8.2.2 Site

La figure 8.7 fait apparaître tous les objets constituant un site, aussi bien les objets spécifiques à une application (*CooperativeFactory*, *CooperativeObject*, *ConditionFunction*, *ConditionManager*) que les objets génériques. Nous ne nous intéressons dans ce chapitre qu'aux objets génériques auxquels nous proposons une implantation.

CooperativeSiteExternalView traite les *callbacks* en provenance du coordinateur. Si la requête est de nature coopérative, elle est traitée par le *CooperationDataBroker* et si elle est de nature coordinatrice, l'objet *ConfigurationManagerModule* la prend en charge.

CooperationDataBroker filtre parmi les objets coopératifs répertoriés par l'objet *CooperativeObjectConfiguration* ceux qui sont concernés par l'information et effectue les invocations de *method* sur les objets filtrés. Pour réaliser le filtrage il fait appel au *ConditionManager*.

ConfigurationManagerModule exécute les commandes de coordination contenues dans le message. Ces commandes ont pour effet de modifier de l'état de *CooperativeObjectConfiguration*. La création et la destruction d'objets coopératifs passent par les *CooperativeFactory* qui sont gérées par l'objet *CooperativeFactories.*

CooperativeFactories est une liste de toutes les *CooperativeFactory* connectées au site.

CooperativeObjectConfiguration est une liste de tous les objets créés sur le site.

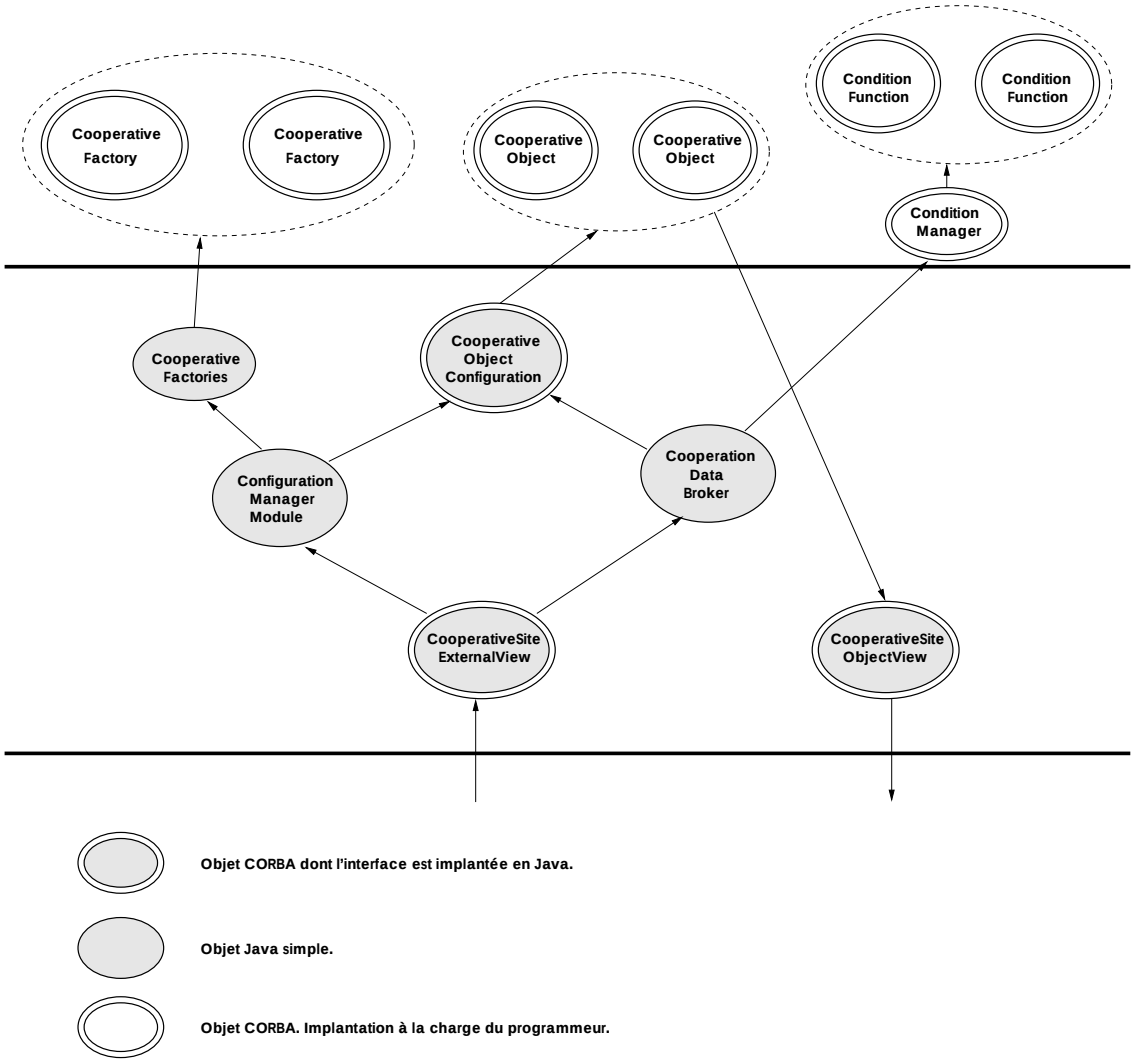


FIG. 8.7 – *Implantation d'un site*

CooperativeSiteObjectView implante les services nécessaires à l'activité des objets coopératifs.

Bien que les interfaces *CooperativeFactory* et *CooperativeObject* soient présentées sur la figure 8.7 sans implantation dans notre environnement, nous offrons des classes les implantant sommairement en Java. Bien évidemment l'implantation pour *CooperativeObject* est partielle car le comportement d'un objet coopératif est spécifique à l'application développée et sa définition reste du ressort du programmeur. Comme leur nom le laisse deviner *BasicCooperativeFactoryBoaImpl* implante l'interface *CooperativeFactory* et *BasicCooperativeObjectBoaImpl* l'interface *CooperativeObject*.

L'objet implantant l'interface IDL *CooperativeSiteApplicationView* n'est pas représenté sur la figure 8.7 mais il n'en est pas pour autant moins important que les autres objets. En effet c'est en créant un objet de classe *CooperativeSiteApplicationViewBoaImpl* qu'un programme provoque l'initialisation des différents composants constituant l'architecture du site.

8.2.3 Coordinateur

8.2.3.1 Les sessions

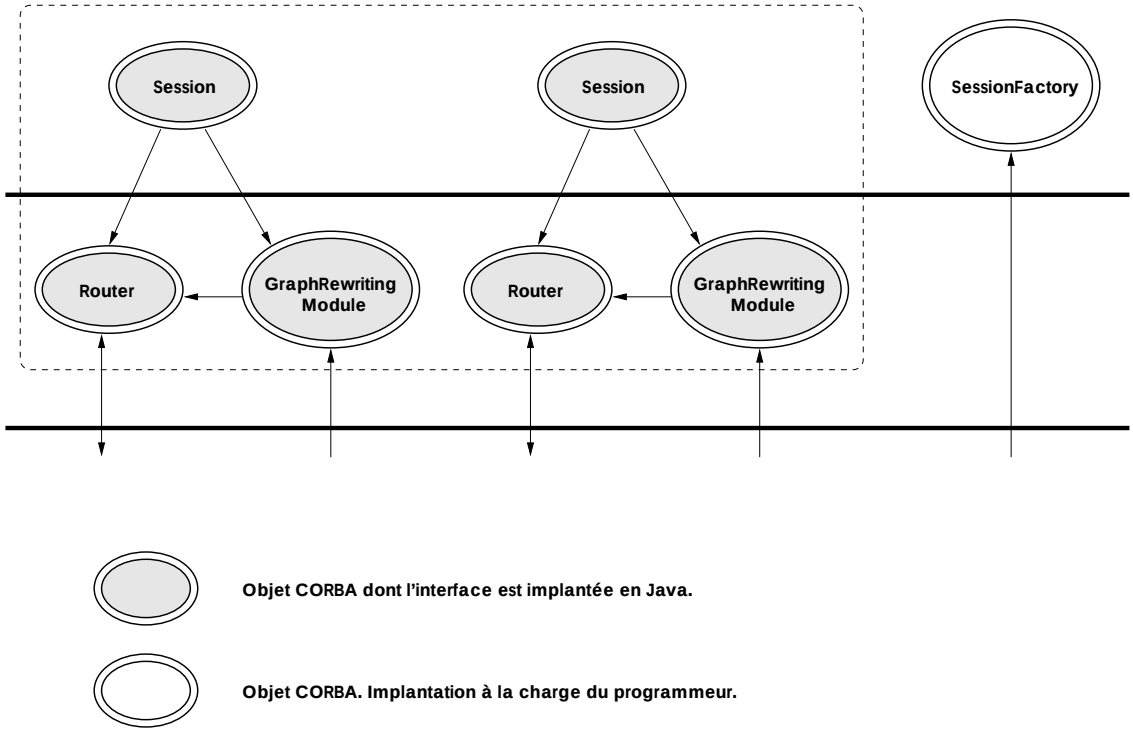


FIG. 8.8 – *Implantation du coordinateur*

La figure 8.8 montre que l'implantation de *SessionFactory* est en général à la charge du programmeur. Elle doit être réalisée dans le langage Java. Le support propose cependant une implantation basique, *BasicSessionFactoryBoaImpl*. En ce qui concerne l'interface *Session*, son implan-

tation *SessionImpl* est fournie par l'environnement. Pour cela nous avons bien évidemment réalisé une implantation du *Communication Manager*, *Router*, et une implantation de *Coordination Manger*, *GraphRewritingModule*. Ce dernier met en œuvre tout le mécanisme de transformation de graphe, mécanisme basé sur un algorithme complexe d'unification par *backtracking*. Le constructeur du *GraphRewritingModule* nécessite la donnée du graphe de coordination initial et des règles de coordination.

8.2.3.2 Médium de coordination

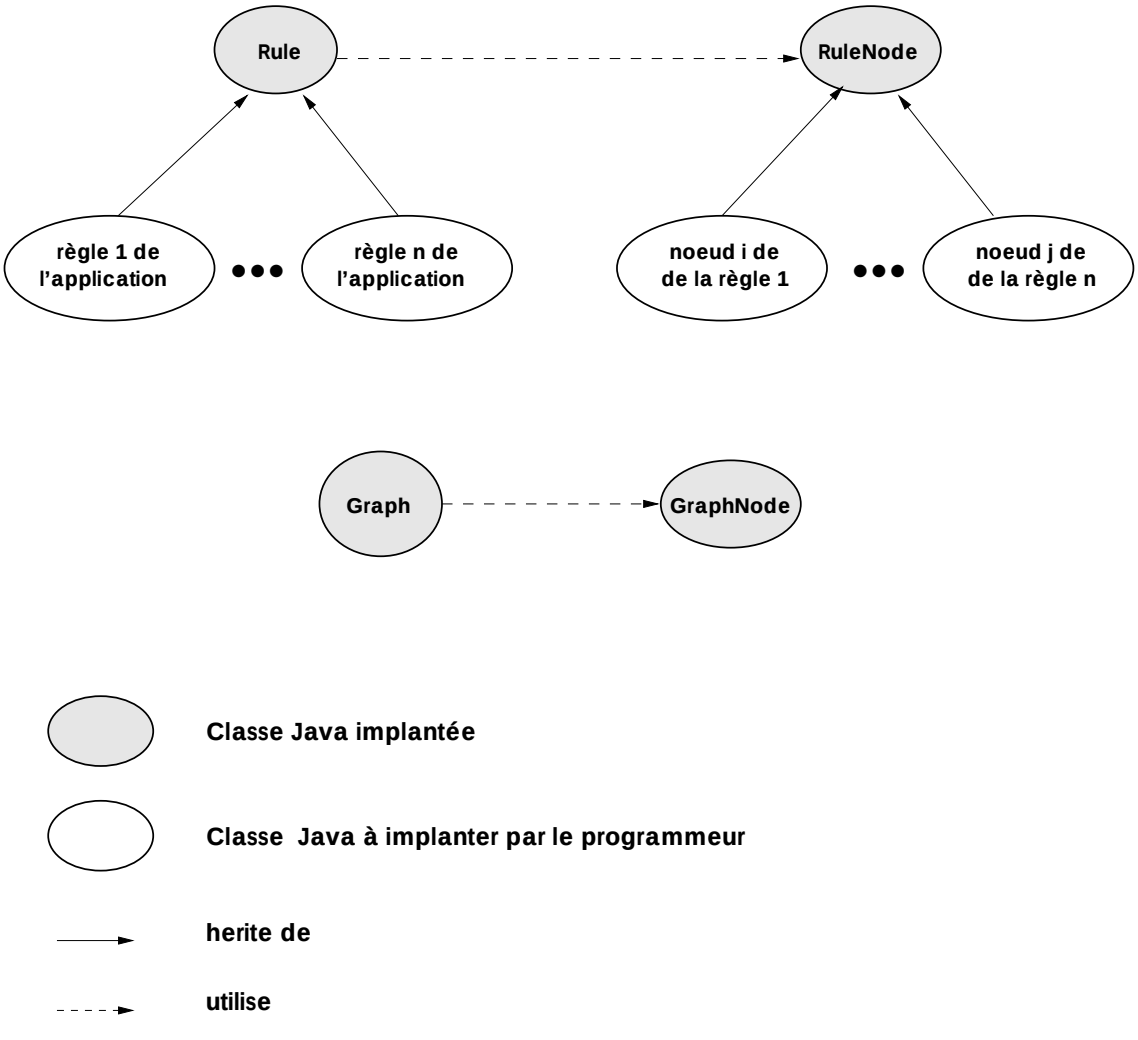


FIG. 8.9 – *Règle et graphe de coordination*

Le médium est constitué du graphe et des règles de coordination. Le graphe de coordination est construit en créant un objet Java de classe *Graph*, puis en ajoutant au graphe les nœuds et les arcs initiaux. Les classes Java des règles sont obtenues par héritage de la classe *Rule* et les nœuds de règle en héritant de *RuleNode*. Tous ces éléments sont repris par la figure 8.9. Comme nous avons pu le préciser au chapitre 2, les noeuds sont typés. La relation d'ordre sur les types est donnée en définissant une classe Java héritant de *TypesOrder* présente parmi les classes du support.

8.3 Application à l'édition coopérative

8.3.1 Description générale des règles de réservation et de libération

Les quatre règles de réservation d'une zone de texte pour l'édition sont données dans la figure 8.10.Les quatre règles de libération d'une zone sont données dans la figure 8.11.

Description des règles de transformation pour la réservation de zones

Rule1: (name = RegleCreation1)

Match Edges
Rn0 -> Rn1, Rn1 -> Rn2
with parameters
Rn1.Start == i and Rn1.End == j and Rn1.State == Free
Provided
i == d and j == f
Delete Nodes
Rn1
Edges
Rn0 -> Rn1, Rn1 -> Rn2
Add Nodes
Rn3 With Parameters Start = d, End = f, State = Busy
Edges
Rn0 -> Rn3, Rn3 -> Rn2

Rule2: (name = RegleCreation2)

Match Edges
Rn0 -> Rn1, Rn1 -> Rn2
with parameters
Rn1.Start == i and Rn1.End == j and Rn1.State == Free
Provided
i < d and j > f
Delete Nodes
Rn1
Edges
Rn0 -> Rn1, Rn1 -> Rn2
Add Nodes
Rn3 With Parameters Start = i, End = d – 1, State = Free
Rn4 With Parameters Start = d, End = f, State = Busy
Rn5 With Parameters Start = f + 1, End = j, State = Free
Edges
Rn0 -> Rn3, Rn3 -> Rn4, Rn4 -> Rn5, Rn5 -> Rn2

Rule3: (name = RegleCreation3)

Match Edges
Rn0 -> Rn1, Rn1 -> Rn2
with parameters
Rn1.Start == i and Rn1.End == j and Rn1.State == Free
Provided
i = d and j > f
Delete Nodes
Rn1
Edges
Rn0 -> Rn1, Rn1 -> Rn2
Add Nodes
Rn3 With Parameters Start = d, End = f, State = Busy
Rn4 With Parameters Start = f + 1, End = j, State = Free
Edges
Rn0 -> Rn3, Rn3 -> Rn4, Rn4 -> Rn2

Rule4: (name = RegleCreation4)

Match Edges
Rn0 -> Rn1 , Rn1 -> Rn2
with parameters
Rn1.Start == i and Rn1.End == j and Rn1.State == Free
Provided
i < d and j == f
Delete Nodes
Rn1
Edges
Rn0 -> Rn1, Rn1 -> Rn2
Add Nodes
Rn3 With Parameters Start = i, End = d – 1, State = Free
Rn4 With Parameters Start = d, End = f, State = Busy
Edges
Rn0 -> Rn3, Rn3 -> Rn4, Rn4 -> Rn2

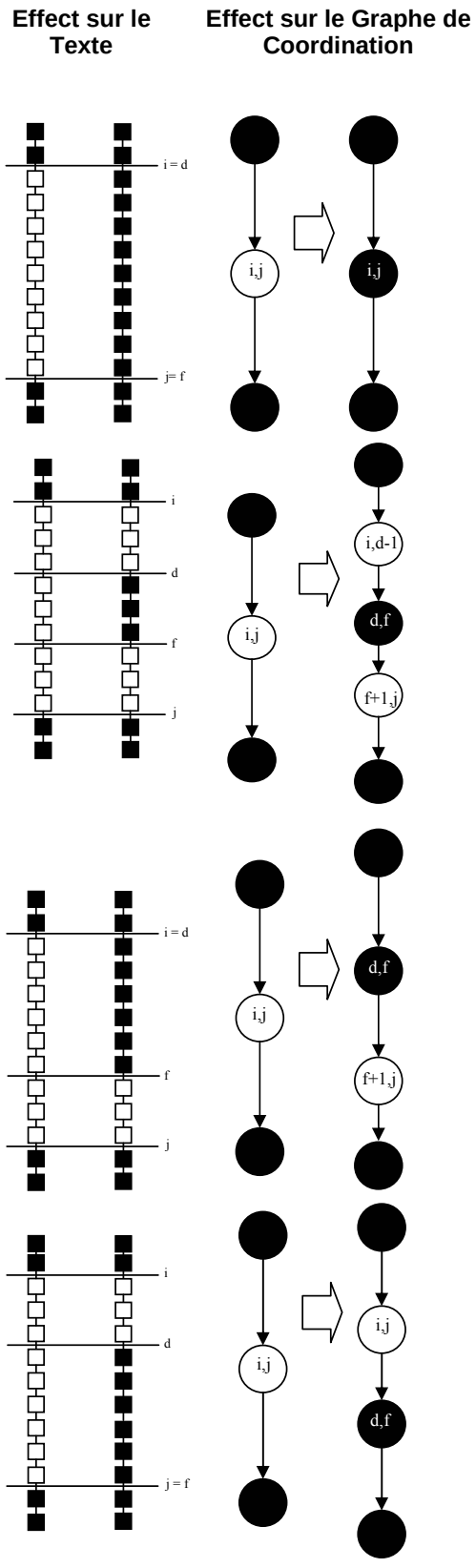


FIG. 8.10 – *Les 4 règles de transformation pour la réservation de zone dans un document à structure linéaire*

Description des règles de transformation pour la libération de zones

Rule5: (name = RegleLiberation1)
Match Edges
Rn0 -> Rn1, Rn1 -> Rn2, Rn2 -> Rn3, Rn3 -> Rn4
with parameters
Rn0.State == Busy and
Rn1.Start == i and Rn1.End == j and Rn1.State == Free and
Rn2.Start == d and Rn2.End == f and Rn2.State == Busy and
Rn3.Start == k and Rn3.End == l and Rn3.State == Free and
Rn4.State == Busy
Provided
j == d – 1 and k == f + 1
Delete Nodes
Rn1, Rn2, Rn_3
Edges
Rn0 -> Rn1, Rn1 -> Rn2, Rn2 -> Rn3, Rn3 -> Rn4
Add Nodes
Rn5 With Parameters Start = i, End = l, State = Free
Edges
Rn0 -> Rn5, Rn5 -> Rn4

Rule6: (name = RegleLiberation2)

Match Edges
Rn0 -> Rn1, Rn1 -> Rn2, Rn2 -> Rn3
with parameters
Rn0.State == Busy and
Rn1.Start == d and Rn1.End == f and Rn1.State == Busy and
Rn2.Start == i and Rn2.End == j and Rn2.State == Free and
Rn3.State == Busy
Provided
i = f + 1
Delete Nodes
Rn1, Rn2
Edges
Rn0 -> Rn1, Rn1 -> Rn2, Rn2 -> Rn3
Add Nodes
Rn4 With Parameters Start = d, End = j, State = Free
Edges
Rn0 -> Rn4, Rn4 -> Rn3

Rule7: (name = RegleLiberation3)

Match Edges
Rn0 -> Rn1, Rn1 -> Rn2, Rn2 -> Rn3
with parameters
Rn0.State == Busy and Rn1.Start == i and Rn1.End == j
and Rn1.State == Free and Rn2.Start == d and Rn2.End == f
and Rn2.State == Busy and Rn3.State == Busy
Provided
j = d – 1
Delete Nodes
Rn1, Rn2
Edges
Rn0 -> Rn1, Rn1 -> Rn2, Rn2 -> Rn3
Add Nodes
Rn4 With Parameters Start = i, End = f, State = Free
Edges
Rn0 -> Rn4, Rn4 -> Rn3

Rule8: (name = RegleLiberation4)

Match Edges
Rn0 -> Rn1, Rn1 -> Rn2
with parameters
Rn0.State == Busy and Rn1.Start == d and Rn1.End == f
and Rn1.State == Busy and Rn2.State == Busy
Provided

Delete Nodes
Rn1
Edges
Rn0 -> Rn1, Rn1 -> Rn2,
Add Nodes
Rn3 With Parameters Start = d, End = f, State = Free
Edges
Rn0 -> Rn3, Rn3 -> Rn2

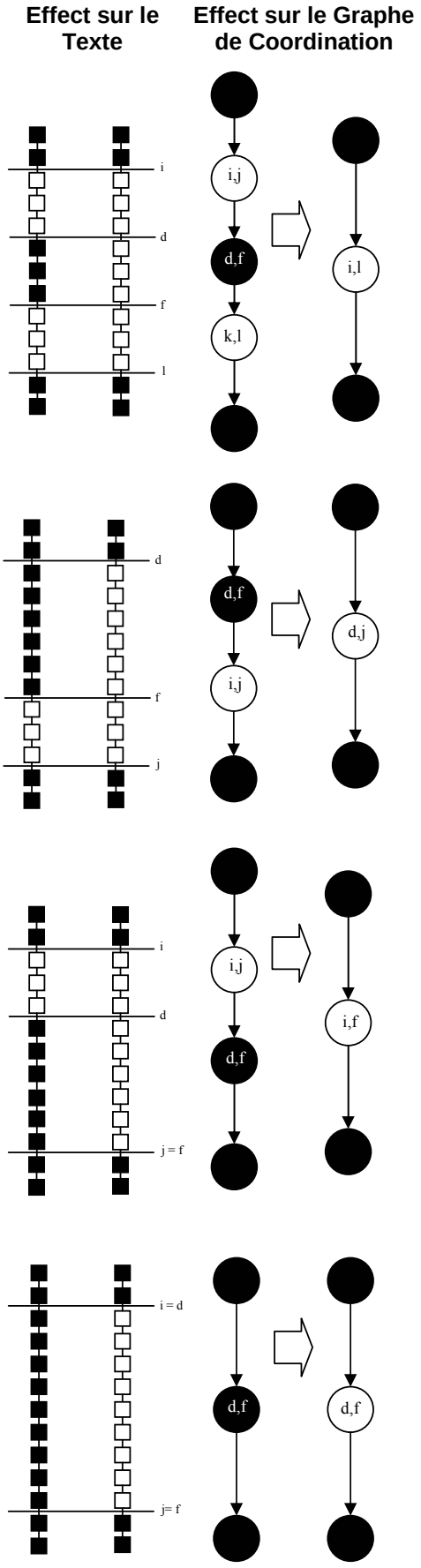


FIG. 8.11 – *Les 4 règles de transformation pour la libération de zone réservée dans un document à structure linéaire*

8.3.2 Syntaxe de la description d'une règle en Java

Les listings Java décrits par les figures 8.12, 8.13 et 8.14 présentent les notations complètes en Java de la description d'une des règles de transformation développée. Il s'agit d'abord, dans la figure 8.12, d'une partie qui décrit le graphe de la règle en détaillant ses différentes parties. Dans ce code, la significations des termes est donnée par : $contexte = R \setminus D$, $retraction = D$, $insertion = A$, où R,D et A sont les notation du chapitre 4, section 4.3.1, page 57.

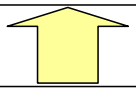
La figure 8.13 décrit ensuite le code de la "fonction garde", définie comme une fonction booléenne reliant les paramètres de la règle (notés d, f) aux paramètres du nœud libre de la règle (notés i, j). Elle doit être vérifiée après identification d'un nœud libre dans le graphe par l'algorithme de recherche d'homomorphisme.

La figure 8.14 décrit enfin le code de la classe que l'on développe pour définir les règles de création des paramètres des nœuds insérés dans le graphe, en précisant leurs types (un entier), leurs contenus (appelés "UserParameter" et qui sont dépendants de l'application et fixés par le programmeur de cette application, ici texte contenu dans la zone, et la position de la zone dans le document global) , et le comportement des objets qui leurs sont associés (classe editeur.AgentZone, dans cet exemple) et qui doivent être créés sur les sites de coopération après l'application d'une règle.

```
abstract public class Rule
{
  public Rule() {}

  public String name;
  public boolean isStar = false;
  public RuleNode centre;
  public EdgesList retractionEdges;
  public NodesList retractionNodes;
  public NodesList insertionNodes;
  public EdgesList insertionEdges;
  private short nodeCount;

  abstract public Boolean guard(dataTypes.graph.NodesList l,
                                IE.Iona.Orbix2.CORBA.Any p);
  public void initInvocation() {}
  public void recordContextNode(RuleNode rn) {}
  public void recordRetractionNode(RuleNode rn) {}
  public void recordInsertionNode(RuleNode rn) {}
  public void addContextEdge(RuleNode initial, RuleNode extremity) {}
  public void addRetractionEdge(RuleNode initial, RuleNode extremity) {}
  public void addInsertionEdge(RuleNode initial, RuleNode extremity) {}
}
```



```
public class RegleCreation2 extends Rule {
  public RegleCreation2() {
    super();
    ParametreEditeur para;
    RuleNode Rn0, Rn2, Rn1, Rn3, Rn4, Rn5;

    name = "RegleCreation2";
    Rn0 = new RuleNode( TypeNoeud.OCCUPE, new DynamicLabel("_a"), null);
    centre = Rn0;
    Rn2 = new RuleNode( TypeNoeud.OCCUPE, new DynamicLabel("_a"), null);
    recordContextNode(Rn0);
    recordContextNode(Rn2);
    Rn1 = new RuleNode( TypeNoeud.LIBRE, new DynamicLabel("_a"), null);
    recordRetractionNode(Rn1);
    addRetractionEdge(Rn0,Rn1);
    addRetractionEdge(Rn1,Rn2);
    Rn3 = new NoeudRegleCreation1( TypeNoeud.LIBRE, new DynamicLabel("_a"),
                                    null, NoeudRegleCreation1.NOEUD3);
    recordInsertionNode(Rn3);
    Rn4 = new NoeudRegleCreation1( TypeNoeud.OCCUPE, new DynamicLabel("_a"),
                                   null, NoeudRegleCreation1.NOEUD4);
    recordInsertionNode(Rn4);
    Rn5 = new NoeudRegleCreation1( TypeNoeud.LIBRE, new DynamicLabel("_a"),
                                   null, NoeudRegleCreation1.NOEUD5);
    recordInsertionNode(Rn5);
    addInsertionEdge(Rn0,Rn3);
    addInsertionEdge(Rn3,Rn4);
    addInsertionEdge(Rn4,Rn5);
    addInsertionEdge(Rn5,Rn2);
    }
   public boolean guard(dataTypes.graph.NodesList l, IE.Iona.Orbix2.CORBA.Any p)
{ // continue sur le listing suivant   }
}
```

FIG. 8.12 – *Première partie de la description complète en Java de la règle de transformation de graphe associée à l'application d'édition partagée*

```
public class RegleCreation2 extends Rule {
  public RegleCreation2() {
    // voir listing précédent
    }
  public boolean guard(dataTypes.graph.NodesList l, IE.Iona.Orbix2.CORBA.Any p)
    {
      RuleNode rn = null;
      GraphNode gn;
      ParametreRegleCreation rpara;
      ParametreEditeur paraEd;

      try
     {
       // extraire le parametre de la regle
       rpara = new ParametreRegleCreation();
       ((Any)p.clone()).extract(rpara);
       // obtenir le noeud du graphe matchant le noeud 4 de la regle
       rn = (RuleNode)l.search((short)4);
       gn = rn.associatedNode;

       // extraire le userparameter de gn noeud du graphe

        paraEd = new ParametreEditeur();
       ((Any)gn.userParameter.clone()).extract(paraEd);

       // calculer les parametres fondamentaux
       int i = paraEd.debut;
       int j = paraEd.fin;
       int d = rpara.debut;
       int f = rpara.fin;

       return ( (i < d) && (f < j) );
     }
      catch(IE.Iona.Orbix2.CORBA.SystemException e)
     {
       System.err.println("RegleCreation1: extract(paraEd)\n"+e.toString());
       return false;
     }
    }
}
```

FIG. 8.13 – *Deuxième partie de la description complète en Java de la règle de transformation de graphe associée à l'application d'édition partagée*

```
package editeur;

import coordination.*;
import IE.Iona.Orbix2.CORBA.Any;
import IE.Iona.Orbix2._CORBA;

public class NoeudRegleCreation1 extends RuleNode
{

  public NoeudRegleCreation1(short t, Label l, Any  p, int numNoeud)
    {
      super(t,l,p);  this.numNoeud = numNoeud;
      className = "editeur.AgentZone";
    }

  public final static int NOEUD3 = 1;
  public final static int NOEUD4 = 2;
  public final static int NOEUD5 = 3;

  int numNoeud;

  public IE.Iona.Orbix2.CORBA.Any getUserParameter(IE.Iona.Orbix2.CORBA.Any ruleParameter,
dataTypes.graph.NodesList nodes)
    {
      RuleNode rn3;
      GraphNode gn;
      ParametreRegleCreation rpara;
      ParametreEditeur paraEd;
      String texte;
      int i,j,d,f;
      IE.Iona.Orbix2.CORBA.Any m;

      try
       {
         // extraire le parametre de la regle
         rpara = new ParametreRegleCreation(); ((Any)ruleParameter.clone()).extract(rpara);

         // obtenir le noeud de graphe matchant le noeud de regle d'index 3
         rn3 = (RuleNode)nodes.search((short)3);
         gn = rn3.associatedNode;

         // extraire le parametre utilisateur du noeud du graphe
         paraEd = new ParametreEditeur(); ((Any)gn.userParameter.clone()).extract(paraEd);

         // parametres fondamentaux
         d = rpara.debut;  f = rpara.fin;
         i = paraEd.debut;  = paraEd.fin;

         m = new IE.Iona.Orbix2.CORBA.Any();
         m.reset();

         switch(numNoeud)
           {
           case NOEUD3: // (i, d-1)
             texte = paraEd.texte.substring(0,d-i);
             m.insert(new ParametreEditeur(texte,i,d-1,(short)0)); break;
           case NOEUD4: // (d, f)
             texte = paraEd.texte.substring(d-i,f-i+1);
              m.insert(new ParametreEditeur(texte,d,f,rpara.site)); break;
            case NOEUD5: // (f+1, j)
              texte = paraEd.texte.substring(f+1-i,j-i+1);
              m.insert(new ParametreEditeur(texte,f+1,j,(short)0)); break;
            default:
              texte = "pb";
            }
          return m;
       }
       catch(Throwable e)
       {
         System.err.println("NoeudRegleCreation1: getUserParameter\n"+
                         e.toString());
         return null;
       }
     }
}
```

FIG. 8.14 – *Dernière partie de la description complète en Java de la règle de transformation de graphe associée à l'application d'édition partagée*

9

Annexe2: Analyse de la testabilité des systèmes communicants avec traitement de la divergence

Ce chapitre présente, de façon détaillée, nos résultats de recherche sur les techniques de validation des systèmes communicants dans le cadre d'une conception basée sur les techniques de description formelle. Une nouvelle structure de représentation des comportements abstraits de systèmes communicants est introduite. Il s'agit de la structure de graphe de refus avec divergence. Une définition formelle de la notion de conformité est utilisée pour définir et analyser la dégradation de la testabilité d'un système lorsqu'il ne peut être commandé et observé que via un environnement. La conformité vue à travers l'environnement et la dégradation de la testabilité sont définies. Un algorithme d'analyse de testabilité est décrit. Notre définition de la divergence est comparée avec celle de Leduc [Leduc, 1995]. Ce chapitre est basé sur les résultats décrits dans les publications [8J] et [7J]. Ces résultats améliorent ceux obtenus dans les publications antérieures, notamment [45Ci] et [42Ci], par le traitement explicite de la divergence. En particulier, nous n'avons plus le paradoxe de l'amélioration de la testabilité lorsqu'on est en présence d'un comportement divergent dans les spécifications. Lorsqu'il y a divergence, un prédicat supplémentaire marquant les états du graphe de refus indique qu'il ne faut pas tenir compte de l'autre étiquette (l'ensemble de refus) qui n'est plus significative en présence de divergence.

9.1 Introduction

La validation des systèmes informatisés de contrôle et de communication est une importante étape de leur conception. La validation précède la mise en exploitation de ces systèmes et vérifie si leur implantation est conforme à leur spécification. Les approches utilisées diffèrent selon la nature du système à valider et selon les formalismes de modélisation et d'implantation.

Pour les systèmes de communication, on peut distinguer deux approches :
vérification de la conformité : Lorsque l'implantation peut être manipulée sans contraintes, c'est à dire au même titre que sa spécification, on parle alors de *vérification* ou d'approche *boîte blanche*.
test de la conformité : Lorsque l'implantation est considérée comme une *boîte noire* que l'on ne manipule que via ses points d'interaction avec l'extérieur, on parle alors de *test* d'une implantation.

Une validation par le test est plus générale qu'une vérification boîte blanche, mais moins discriminante pour les erreurs d'implantation. Ceci s'explique par les contraintes imposées à la

procédure de test.

Des travaux de recherche ont porté sur la capacité théorique du test de conformité à détecter les implémentations erronées. Dans le domaine des systèmes communicants, ces travaux sont exclusivement basés sur les machines à états finies et ont considéré des systèmes entièrement accessibles ([Vuong et al., 1993]).

En pratique, un système sous test peut ne pas être librement commandé et observé. C'est le cas d'un système partiellement accessible (en observation ou en commande), notamment lors du test de conformité des éléments d'un système multi-composant. Ce contexte constitue une des motivations principales de cet article. Il s'agit de formaliser la détection de la non-conformité par le test, précisément la « dégradation de testabilité », et d'évaluer cette dégradation pour un système dans son environnement.

L'approche proposée présente plusieurs originalités. Tout d'abord l'analyse n'est pas purement structurelle, c'est à dire basée sur des critères de testabilité « a priori » déduits de l'architecture du système sous test, comme dans la plupart des travaux existants ([Dssouli et Fournier, 1990, Ghedamsi et al., 1992, Petrenko et al., 1993, Vuong et al., 1993]). L'analyse n'est pas non plus purement comportementale. Elle tient compte a la fois de l'architecture du système sous test, de son comportement et du comportement de son environnement. L'approche proposée fait appel à une description formelle des comportements et à des opérations algébriques de composition. Elle peut s'appliquer, dès les premiers stades de conception, à tout formalisme qui utilise pour la composition la sémantique du rendez-vous. De plus, le non déterminisme est possible dans la description du comportement attendu. Ceci est très utile pour faire abstraction de détails ou de choix d'implémentation. Enfin, un modèle implicite de fautes est utilisé, mais celles-ci n'ont pas à être énumérées. Un ordre partiel des fautes, compatible avec la définition de conformité, conduit à une analyse progressive qui identifie les comportements erronés « limites ». Ces comportements sont détectables, ainsi que toute implémentation plus erronée; toute implémentation strictement moins erronée n'est pas détectable.

Le *test d'un système communicant à travers un environnement* sera étudié dans le cadre formel du langage LOTOS [1] [Brinksma et al., 1987]. L'analyse de testabilité proposée initialement dans [45Ci] sera étendue pour tenir compte de la présence éventuelle d'exécutions divergentes [2] pour lesquelles aucune communication avec le système n'est plus possible.

Cet article est organisé comme suit. La deuxième section introduit le graphe de refus en tant que structure de représentation du comportement d'un système communicant, et rappelle la notion de conformité dans le cadre de la théorie de test de LOTOS. La troisième section définit formellement la conformité d'une implémentation par rapport à sa spécification.

La quatrième section définit le test à travers un environnement et la relation de conformité associée. Les opérations de composition et de restriction de graphes de refus sont introduites pour décrire le comportement partiellement observable et contrôlable d'un système sous test. Ces opé-

1. LOTOS : « Language Of Temporal Ordering Specification » est un langage de description algébrique normalisé par l'OSI, l'Organisation de Standardisation Internationale. Une introduction à ce langage est fournie dans [Bolognesi et Brinksma, 1988]. Ce langage s'inspire de CCS de Milner [Milner, 1980], et de CSP de Hoare [Hoare, 1985] les formalismes algébriques les plus utilisés actuellement.

2. La « divergence » désigne l'état d'un système qui évolue indéfiniment sans que rien ne soit perceptible par l'utilisateur. Lorsqu'on représente le comportement d'un système communicant par un graphe étiqueté fini, la divergence traduit la présence de cycles de transitions non observables qui sont notées τ dans les algèbres de processus.

rations sont ensuite utilisées lors de la présentation de l'algorithme d'analyse de testabilité. La dernière section présente brièvement l'environnement logiciel qui implante la technique proposée.

9.2 Représentation du comportement des systèmes communicants

9.2.1 Description de comportement par un graphe de refus

Le graphe de refus est la structure de base pour la représentation du comportement non déterministe des systèmes communicants. Cette structure comporte un ensemble d'états et de transitions entre ces états. Chaque transition est étiquetée par l'action engagée lors de ce changement d'état. Ainsi, un protocole connecté de transfert de données (figure 9.1), de son état initial, peut recevoir une requête de connexion de l'utilisateur avant de se mettre dans un état à partir duquel l'acceptation de la requête peut être confirmée. Dans le cadre des systèmes communicants, une action modélise généralement la réception ou l'émission d'un message. En plus des étiquettes qui annotent les transitions, des listes d'actions sont associées à chaque état d'un graphe de refus (la liste associée à l'état g est notée $Ref(g)$). Chaque liste déclare les actions acceptées (par le système dans cet état) mais susceptibles d'être refusées suite à des évolutions internes du système modélisé. Ainsi un protocole d'établissement de connexion, après avoir reçu la requête (état 2), peut refuser le transfert de données si par exemple tous les ports de communication sont déjà alloués à d'autres utilisateurs. Ce refus est traduit par la présence de l'ensemble $\{transfer\}$ dans la liste des actions refusées à l'état 2. Enfin l'absence d'actions refusées à l'état 3 signifie qu'une fois entamé le transfert de données ne peut être arrêté qu'à l'initiative de l'utilisateur.

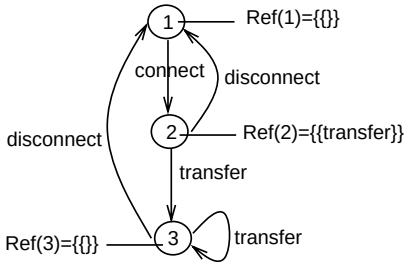


FIG. 9.1 – *Le graphe de refus d'un protocole simplifié de transfert de données orienté connexion.*

La représentation du comportement dynamique d'un système communicant par un graphe de refus présente l'avantage de la concision offerte par un haut pouvoir d'abstraction tout en étant très pragmatique pour les algorithmes de validation en général et pour l'analyse de testabilité en particulier. L'interprétation par refus est supportée par la notion de synchronisation par rendez-vous qui suppose que l'utilisateur peut se trouver bloqué si l'action qu'il veut exécuter en synchronisation avec le système modélisé (c'est à dire une communication dans le cadre de nos applications) n'est pas acceptée par ce système dans son état présent. Pour ne pas se bloquer, le testeur d'un système selon une sémantique de rendez-vous doit prendre en compte les actions offertes à chaque étape de l'exécution du test, ainsi que le « minimum » actions alternatives qu'il doit offrir ou accepter si l'action qu'il désire exécuter n'est pas possible. Le graphe de refus permet donc de fournir ces information pour la réalisation du test et uniquement ces informations sans distinguer par exemple

le blocage dû à un non déterminisme avant, pendant ou après l'exécution d'une action, contrairement aux systèmes de transitions. Les ensembles de refus permettent simplement de savoir quelles sont, parmi les actions possibles à partir d'un état, les choix qui sont offerts à l'utilisateur. De plus, les réceptions n'étant pas obligatoirement distinguées des émissions, une seule transition peut très bien décrire la possibilité d'un échange de message du système vers son utilisateur (ce que l'on appelle usuellement indication) ou bien de l'utilisateur vers le système (ce que l'on appelle usuellement requête). Dans le cas du comportement décrit dans la figure 9.1, cette abstraction permet de décrire l'acceptation de la requête de connexion sans utiliser de confirmation explicite. A l'état 2, le refus de $transfer$ signifie que la connexion n'est pas établie. Ceci peut être une décision prise par le système ou son utilisateur puisque le $disconnect$ peut représenter un message reçu par le système ce qui signifierait que la déconnexion a été décidée par l'utilisateur ou émis par le système ce qui signifierait que c'est ce dernier qui n'a pas pu établir la connexion. Dans cet état le choix entre la déconnexion et le transfert est un choix *interne* au système. Par contre, dans l'état 3, le choix entre ces deux actions est *externe* au système, ce qui signifie que seul l'utilisateur peut décider de poursuivre le transfert des données ou au contraire de l'interrompre.

9.2.2 Graphe de refus associé à un système de transitions

Les systèmes de transitions décrivant le système sous test et son environnement peuvent être obtenus à partir de formalismes plus évolués, comme une spécification en langage LOTOS. Les graphes de refus sont ensuite obtenus par déterminisation du système de transitions. Un état du graphe de refus regroupe des états du système de transitions : $g = \{s_1, s_2, \cdots\}$. L'algorithme de déterminisation est celui utilisé pour la déterminisation des automates, dont une description est fournie dans [Hopcroft et Ullman, 1989].

Le calcul des attributs propres aux graphes de refus, à savoir, le calcul du prédicat de divergence, Div, et le calcul des ensembles de refus, Ref, est maintenant détaillé.

$Div(g) \equiv_{\text{def}} \exists s \in g$ tel que $s \xrightarrow{\tau^\infty}$. Un état du graphe de refus diverge si et seulement si l'un des états du système de transitions diverge. Alors $s \xrightarrow{\tau^\infty}$ signifie qu'il est possible d'exécuter une infinité de transitions internes à partir de s sans passer obligatoirement par une transition visible. Formellement : $s \xrightarrow{\tau^\infty} \equiv_{\text{def}} \exists s' \ : \ s \xrightarrow{\tau^*} s'$, où s' est un état par lequel passe un cycle de transitions étiquetées par l'action interne τ. Formellement, $\exists n \geq 1 \ : s' \xrightarrow{\tau^n} s'$.

$Ref(g) = \{ActionsPossiblesDans(g) \setminus ActionsPossiblesDans(s), s \in g\}$. Où "*ActionsPossiblesDans(g)*" désigne l'ensemble des actions possibles à partir de l'état g (par exemple pour le graphe de la figure 9.1, *ActionsPossiblesDans(2)*$= \{ disconnect, transfer\}$). Pour les systèmes de transition la notion d'actions possibles est définie modulo l'action interne τ. C'est à dire une action est possible à partir de l'état s, s'il y a une transition qui part de s et qui est étiquetée par cette action ($s \xrightarrow{a} s'$) ou si cette action est possible à partir d'un état qu'on peut atteindre à partir de s par une ou plusieurs transitions internes ($s \xrightarrow{\tau} s'$ et $a \in ActionsPossiblesDans(s')$ ou de façon équivalente $s \xrightarrow{\tau^n} s'$ et $s' \xrightarrow{a} s''$). Pour avoir une représentation unique des comportements équivalents, un ensemble de refus est normalisé par extraction des éléments qui sont des sous-ensembles d'un autre élément de ce même ensemble de refus : $Ref(g) \ : = Ref(g) \setminus \{X \in Ref(g), \exists Y \in Ref(g) \ : \ X \subseteq Y\}$.

La figure 9.2 fournit deux exemples de construction de graphes de refus à partir de systèmes de transitions. L'on remarque en particulier la différence entre l'ensemble de refus $\{\{a\}, \{b\}\}$

et $\{\{a,b\},\cdots\}$. Le premier signifie que a et b peuvent être refusées séparément (dans l'état 2 ou dans l'état 3). Le second signifie que a et b peuvent être refusées simultanément (dans l'état 4). Remarquons aussi que le calcul des ensembles de refus de l'état $\{2,3,4\}$ du graphe $gr(S2)$ s'obtient en regroupant les ensembles refusés dans les états 2, 3 et 4; soit respectivement $\{a,b,c\}\setminus\{a\}=\{b,c\}$, $\{a,b,c\}\setminus\{a,b\}=\{c\}$ et $\{a,b,c\}\setminus\{c\}=\{a,b\}$. Ce qui donne après simplification (en enlevant $\{c\}$ qui est un sous-ensemble de $\{b,c\}$) l'ensemble de refus $\{\{b,c\},\{a,b\}\}$.

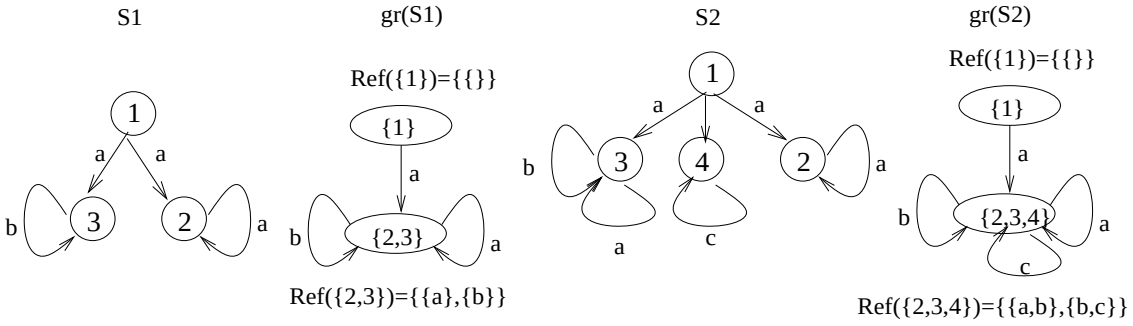


FIG. 9.2 – *Deux exemples de construction de graphes de refus.*

9.2.3 Motivations et choix pour le traitement de la divergence

Une première approche d'analyse de testabilité, implémentée dans [7], a consisté à supposer que le modèle de l'implantation ne peut pas indéfiniment évoluer sans communiquer avec son environnement de test : la divergence n'était pas admise. En effet, lorsque le choix des actions observées, (la restriction), crée une divergence après une séquence d'actions, il n'est plus possible de calculer les ensembles de refus associés à cette séquence dans la restriction d'un graphe de refus à partir des ensembles de refus initiaux. En pratique, le calcul de l'ensemble de refus d'un groupe d'états du graphe de refus restreint n'est pas forcément correct lorsqu'il y a divergence à l'intérieur de ce groupe, c'est à dire lorsqu'il y a entre ces états, un cycle étiqueté par des actions non observées.

Il est toutefois possible de prendre en compte la possibilité de divergence dans l'analyse de testabilité, au moyen d'une interprétation nouvelle de la divergence. Il s'agit d'assurer la préservation de la conformité en présence de divergence, et de conserver la composition et la restriction d'un graphe de refus. Ces deux opérations seront expliquées plus loin.

L'interprétation catastrophique de la divergence selon De Nicola Hennessy et Hoare [Nicola et Hennessy, 1984, Hoare, 1985] suppose qu'un processus qui diverge après la trace σ (en présentant un cycle d'actions non observables), sera considéré comme divergent après toutes les traces qui contiennent σ. Cette interprétation n'est pas réaliste dans le cas de notre représentation état/transition, car un même état peut être atteint par différentes séquences et pourrait ainsi être jugé divergent même s'il ne présente pas de cycle d'évolution internes. Ainsi dans le système de transitions de la figure 9.3, l'état initial, bien que sans possibilité d'évolution infinie par des transitions internes τ, serait considéré divergent selon l'interprétation catastrophique car il est accessible par la séquence $a.b$ qui passe par l'état divergent 2.

Comme Leduc [Leduc, 1991], nous choisissons ici une définition moins restrictive qui considère que la divergence est associée à l'arrivée dans un état divergent et non à la possibilité de

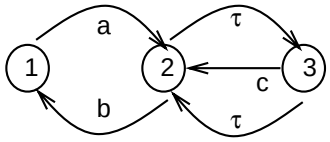


FIG. 9.3 – *Divergence dans un système de transitions. Les actions a,b et c sont des transitions externes, τ dénote des transitions internes.*

passer par un état divergent. Ainsi pour le système de transitions de la figure 9.3, seuls les états 2 et 3 sont considérés divergents.

Contrairement à Leduc, nous ne distinguons pas et pour des raisons purement pratiques, entre blocage [3] et divergence. En effet dans les deux cas, le système ne communique plus avec son environnement. Une divergence dans la spécification permettra de juger conformes des implantations qui bloquent après l'exécution de la séquence qui aboutit à cette divergence.

9.2.4 Comparaison avec les trois relations de conformité de la nouvelle théorie de test de Leduc

Dans [Leduc, 1995], Leduc a introduit trois relations de conformité comme suit :

$$I \ \mathbf{conf_L}\ S \ ssi \ \forall \sigma \in Conv(S), \forall A \subseteq L :$$
$$(i)\sigma \in Conv(I) \Rightarrow ((\sigma, A) \in Fail(I) \Rightarrow (\sigma, A) \in Fail(S)) \mathbf{et}$$
$$(ii)\sigma \in Div(I) \Rightarrow \mathbf{une\ certaine\ condition}$$

Où "$\mathbf{une\ certaine\ condition}$" peut être l'une des trois conditions suivantes : "**vrai**", "$(\sigma, A) \in Fail(S)$", ou "**faux**".

Nous prouverons dans ce qui suit que, sous certaines hypothèses de test , notre définition de conformité et celles définies par Leduc sont identiques.

Nous considérons que, dans la pratique, il n'est pas possible, en appliquant des tests, de distinguer entre la divergence (liveness) et le blocage dans l'IUT. Nous supposons alors que tous les modèles possibles de l'IUT ont seulement les traces convergentes. Formellement : $\forall \sigma \in Tr(I), \models Conv(\sigma)$ où $Conv$ défini par : $Conv(\sigma) = Conv(i)$ où i est l'état atteint par σ depuis l'état initial de l'IUT : $i : i_0 \stackrel{\sigma}{\Longrightarrow} i$; et où $Conv(i) \equiv_{def} \neg Div(i)$. Dans le contexte de l'analyse de la testabilité, la divergence est à traiter seulement dans la spécification ; et nous nous intéressons à ses conséquences sur la testabilité de l'IUT.

Vue cette hypothèse, nous pouvons ignorer la clause (ii) dans la définition ci-dessus de la conformité. Et la définition résultante de la conformité de Leduc est la suivante :

$$I \ \mathbf{conf_L}\ S \ ssi \ \forall \sigma \in Conv(S), \forall A \subseteq L :$$
$$(\sigma, A) \in Fail(I) \Rightarrow (\sigma, A) \in Fail(S)$$

3. Le blocage d'un système signifie que ce système se trouve dans un état puits qui n'offre aucune transition ni observable ni non observable.

Cette définition est logiquement équivalente à la suivante :

$$I \ \mathbf{conf_L} \ S \ ssi \ \forall \sigma \in Tr(S), \forall A \subseteq L :$$
$$\sigma \in Conv(S) \Rightarrow ((\sigma, A) \in Fail(I) \Rightarrow (\sigma, A) \in Fail(S)) \ (*\mathbf{Eq1}*)$$

En utilisant les notations de Leduc, la définition de conformité de Table 9.1 s'écrit :

$$I \ \mathbf{conf_L} \ S \ ssi \ \forall \sigma \in Tr(S), \forall A \subseteq L :$$
$$(\sigma, A) \in Fail(I) \Rightarrow ((\sigma, A) \in Fail(S) \ \vee \ \neg \, Conv(S)) \ (*\mathbf{Eq2}*)$$

L'équation Eq1 a la forme logique : $X \ \Rightarrow \ (Y \ \Rightarrow \ Z)$ qui est logiquement équivalente à : $(\neg X) \ \vee \ ((\neg \, Y) \ \vee \ (Z))$ où X dénote $\sigma \in Conv(S)$; Y dénote $(\sigma, A) \in Fail(I)$ et où Z dénote $(\sigma, A) \in Fail(S)$.

Cette équivalence est basée sur le fait que $\alpha \Rightarrow \beta$ est logiquement équivalent à $\neg\alpha \ \vee \ \beta$.

L'équation Eq2 a la forme logique : $Y \ \Rightarrow \ (Z \ \vee \ \neg \ X)$ qui est logiquement équivalente à : $(\neg \, Y) \ \vee \ (Z \ \vee \ \neg \ X)$.

En conclusion, considérant l'associativité et le commutativité de l'opérateur logique de $\vee$, nous déduisons que $Eq1 = Eq2$.

9.3 Conformité d'une implantation par rapport à sa spécification

Pour définir la conformité d'une implantation à sa spécification, plusieurs relations ont été proposées dans le cadre des processus algébriques. Elles consistent à comparer les séquences de communications possibles, ou exécutions, pour chaque système. Certaines relations définissent la conformité par rapport à la *bissimulation*, d'autres relations considèrent les séquences de communications, ou *traces*, de chaque système à partir de son état initial. Nous pouvons dire que le premier type de relations est de l'école de Milner et Park et que le second est de l'école de Hoare.

La bissimulation compare le comportement de deux systèmes en s'intéressant aux séquences de communication possibles à partir de leurs états initiaux ainsi qu'à leur comportement sur les états intermédiaires. Son grand intérêt est son utilisation dans le contexte de preuve algébrique. En particulier, des représentations simples permettent de décrire des comportements équivalents. Une comparaison intéressante de ces relations est établie par Van Glabbeek [van Glabbeek, 1990].

Le principe des traces fait abstraction des états intermédiaires et paraît plus proche d'une vérification selon le principe de la boîte noire [4]. Ainsi des relations dites de test ou d'implantation, ont été proposées pour définir la conformité des traces, ou observations. Les premiers travaux dans ce domaine sont ceux de Phillips [Phillips, 1987] et De Nicola-Hennessy [Nicola et Hennessy, 1984].

Comme Brinksma, Scollo et Steenbergen [Brinksma et al., 1987], nous considérons qu'une implantation I est conforme à une spécification S lorsque : pour toute trace σ du comportement spécifié S, si l'implantation I peut évoluer par σ alors les ensembles d'actions A qu'elle peut refuser d'exécuter après cette évolution, sont (au plus) ceux que la spécification S peut refuser après

4. Abramsky [Abramsky, 1987] a caractérisé la notion de bissimulation selon le principe de test boîte noire. Néanmoins, sa mise en oeuvre complexe l'empêche d'être raisonnablement envisageable pour une procédure de test.

σ. Nous considérons, par contre, que la divergence entraîne le refus de toute action. Outre l'intérêt théorique de la notion de conformité avec divergence, nous montrons ici son intérêt pratique dans le cas de la testabilité d'un système à travers un environnement. Cette extension permet de définir (voir section 9.4.2) la restriction d'un graphe de refus même en présence de divergence, et reste compatible avec la définition de la conformité.

La table 9.1 fournit formellement cette définition.

« L'implantation I est conforme à la spécification S » est défini par :

$$I \ \underline{\mathbf{conf}} \ S \equiv_{\text{def}}$$
$$\forall \sigma \in Tr(S), \forall A \subseteq L \ : \quad \begin{array}{ll} \mathbf{si} & \exists i \left(i_0 \stackrel{\sigma}{\Longrightarrow} i \ \mathbf{et} \ A \Subset \lceil Ref(i) \rceil \right) \\ \mathbf{alors} & \exists s \left(s_0 \stackrel{\sigma}{\Longrightarrow} s \ \mathbf{et} \ (A \Subset \lceil Ref(s) \rceil \ \mathbf{ou} \ Div(s)) \right) \end{array}$$

Où les notations utilisées ont les significations suivantes :

- s_0 (resp. i_0) désigne l'état initial du graphe de refus S (resp. I).
- L'ensemble L désigne toutes les actions utilisées par les graphes de refus.
- Pour tout couple d'états, g, g' du graphe de refus, pour toute séquence d'actions σ, la notation $g \stackrel{\sigma}{\Longrightarrow} g'$ signifie que le système peut, de son état g exécuter la séquence d'actions σ et se retrouver dans l'état g'. Il existe, dans le graphe, un chemin qui part de g et qui aboutit à g' dont les étiquettes concaténées forment σ : $\exists g_1 \cdots g_{n-1}$ tels que : $g \stackrel{a_1}{\Longrightarrow} g_1$, $g_1 \stackrel{a_2}{\Longrightarrow} g_2$, $\cdots$, $g_{n-1} \stackrel{a_n}{\Longrightarrow} g'$, avec $\sigma = a_1 \cdots a_n$.
- La notation élémentaire $g_i \stackrel{a}{\Longrightarrow} g_k$ désigne la transition de l'état g_i vers l'état g_k après l'exécution de l'action $a \in L$ par le système.
- Une séquence d'actions σ est appelée trace du graphe de refus G lorsque son exécution est possible à partir de l'état initial : $\sigma \in Tr(G) \equiv_{\text{def}} \exists g' : g_0 \stackrel{\sigma}{\Longrightarrow} g'$.
- $A \Subset Ref(g)$ signifie que les actions de l'ensemble A peuvent être simultanément refusées compte tenu de l'ensemble de refus $Ref(g)$ pour l'état g. L'ensemble A est contenu dans l'un des ensembles listés par $Ref(g)$: $A \Subset Ref(g) \equiv_{\text{def}} \exists B \in Ref(g)$ tel que $A \subseteq B$.
- La notation $\lceil \ \rceil$ désigne l'opération d'élargissement d'un ensemble de refus aux actions qui ne sont jamais autorisées dans l'état considéré. Cette opération est nécessaire pour écarter les implantations non conformes qui, dans un état donné, pourraient autoriser une action nouvelle. $\lceil Ref(g) \rceil = \{A \cup L \setminus ActionsPossiblesDans(g), A \in Ref(g)\}$.

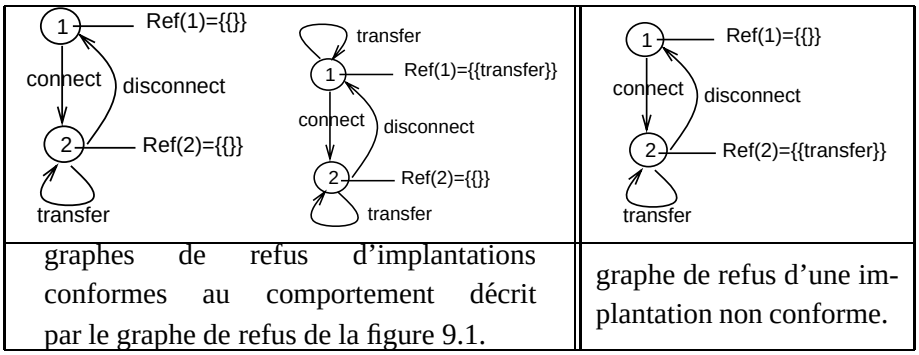
TAB. 9.1 – *Définition formelle de la conformité*

Deux raisons expliquent le choix de cette relation pour définir formellement la conformité d'une implémentation par rapport à sa spécification.

La notion de conformité retenue n'exige pas que l'ensemble des traces de l'implantation soit un sous-ensemble de celui de la spécification, ni le contraire. Ceci constitue une caractéristique spécifique à cette notion de conformité qui ne veut pas « dépasser » les limites pratiques du test

de conformité. En effet, soit une spécification qui admet une seule transition à partir de l'état initial, pour prouver qu'un implémentation n'étend pas les traces de cette spécification, il faudrait vérifier qu'il n'est pas possible d'exécuter toute autre transition à partir de cet état, soit $|L| - 1$ tests supplémentaires (où $|L|$ désigne le nombre d'actions de l'alphabet considéré).

Par ailleurs, il est possible d'identifier l'ensemble des tests qui détectent la non conformité d'une implantation erronée [Brinksma, 1988]. Cette procédure de génération de test peut être étendue aux comportements cycliques [47Ci].



graphes de refus d'implantations conformes au comportement décrit par le graphe de refus de la figure 9.1.

graphe de refus d'une implantation non conforme.

FIG. 9.4 – *Exemples de conformité.*

La figure 9.4 décrit quelques exemples de comparaison de graphes de refus pour la conformité retenue. Le premier graphe décrit le comportement d'un système de transfert de donnée orienté connexion toujours disponible, il est conforme. Le second graphe de refus décrit le comportement d'un système de transfert qui permet des transferts même sans connexion, mais qui peut ne pas être disponible et obliger l'utilisateur à établir une connexion pour poursuivre, voire, commencer un échange de données. Toutefois, si la connexion est établie, seul l'utilisateur a la possibilité d'interrompre le transfert. Ce comportement est aussi jugé conforme. Le graphe de refus de la partie de droite de cette figure est déclaré non conforme car même après l'établissement d'une connexion et à tout moment du transfert, le système peut rompre la connexion.

La section suivante explique le principe de la vérification de la conformité dans une approche de test.

9.3.1 Détection de la non-conformité par le test

Dans les théories de test de conformité qui utilisent les systèmes de transitions, directement ou au moyen de langages algébriques, pour spécifier le comportement d'un système communicant, la détection de la non-conformité se fait par la *comparaison* des *résultats* d'un ou plusieurs tests appliqués à l'implantation avec les résultats que donneraient ces mêmes tests appliqués à la spécification formelle.

La comparaison porte sur les tests que *doit réussir* ou que *peut réussir* une implantation. La réussite d'un test est définie par l'observation d'un message (ou action) dit de succès et noté « w » par De Nicola et Hennessy et δ par Brinksma, ou de manière équivalente par l'arrivée du testeur (c-à-d le système qui applique l'ensemble des tests) dans un état qui fait partie de l'ensemble des états dits de succès, noté « SUCC » par Phillips. Le résultat de l'application d'un test est alors succès ou échec. Comme les spécifications sont non déterministes, le même test peut conduire à

un succès ou à un échec, selon l'évolution interne du système sous test. Un système doit réussir un test lorsqu'aucune exécution de ce test ne peut aboutir à un échec. Un système peut réussir un test lorsque le succès du test n'est pas garanti.

L'application (ou l'exécution) d'un test se définit comme une suite de commandes en entrée et d'observations des sorties. Plusieurs sorties peuvent correspondre à la même entrée appliquée au même système dans le même état. La suite de l'exécution dépend donc de la sortie observée. Cette définition peut se formaliser en faisant abstraction de la notion d'entrée/sortie et utilisant le principe de « synchronisation » entre le test et l'implantation. Le système sous test et le test (ou testeur si on regroupe plusieurs tests) sont alors décrits par des systèmes de transitions. La synchronisation entre un système sous test S et son testeur T se définit par un opérateur de composition, détaillé plus loin en section 9.4.2, noté « | », qui décrit, sous forme de système de transitions $S|T$, l'exécution des tests. Le système sous test et le testeur peuvent évoluer de façon autonome par des transitions internes et doivent évoluer simultanément pour toute autre transition. Les évolutions indépendantes expriment des choix faits par le système sous test ou des choix faits par le testeur. Une évolution synchronisée traduit une requête du testeur vers le système sous test ou une réponse du système vers le testeur et vice-versa. Une évolution synchronisée entraîne le changement d'état des deux participants. Une évolution interne entraîne un seul changement d'état. Tous les résultats de l'application du test T au système S sont donnés par le comportement $S|T$: en particulier, échec s'il existe un état de fin d'exécution de $S|T$ qui ne résulte pas d'une décision du testeur. Ceci traduit une requête non implantée dans le système, ou une réponse incorrecte du système. Plus précisément, un état puits, c-à-d sans transition possible, (s,t) de $S|T$ indique un succès si t est un état de succès, et correspond à un échec si t n'est pas un état de succès. En réduisant les états de succès d'un testeur à l'ensemble de ses états puits, ou terminaux, tout arrêt d'exécution doit correspondre à une décision d'arrêt du testeur.

Une implantation est conforme si et seulement si elle réussit tous les tests que doit réussir sa spécification. Ceci correspond à la contraposée de la définition formelle [Tretmans, 1992]. Les différences entre théories de tests dans le contexte algébrique résultent du langage de test. Celui-ci peut être identique au langage de spécification, comme ici, ou bien faire appel à un formalisme plus évolué comme dans [Langerak, 1990, Phillips, 1987, Abramsky, 1987] où le blocage ne signifie plus « présence d'erreur », mais permet de choisir différentes poursuites du test en fonction des actions refusées.

La notion de conformité retenue ici permet d'identifier l'ensemble des tests nécessaires et suffisants pour écarter les implémentations erronées par rapport à une spécification donnée.

La dérivation de ces tests se fait par la transformation des ensembles de refus de la spécification selon [Brinksma, 1988], suivie d'une réécriture du graphe obtenu pour éliminer les cycles selon la technique des « chemins maximaux » [1T],[48Ci].

Le testeur $T(S)$ d'une spécification formelle S mène à un échec (au moins) s'il est appliqué à une implantation non conforme I, ou encore $I|T(S)$ contient un état d'échec.

Comme dans le cas des techniques basées sur les machines à états finies, l'exactitude de la méthode repose sur l'hypothèse essentielle que le nombre d'états de l'implémentation est identique à celui de la spécification, ou au moins borné par une valeur connue par le testeur. Avec cette hypothèse, des tests finis détectent les mêmes erreurs que le testeur canonique. Dans notre cas, les hypothèses sur le nombre d'états de l'implémentation et de la spécification portent sur les graphes de refus.

La figure 9.5 décrit, sous forme de système de transitions à gauche et sous forme de graphe de refus à droite, les tests que doit réussir le transfert de données orienté connexion dont le graphe de refus était décrit par la figure 9.1. Les états visualisés par des cercles en blanc représentent les copies des états en noir et à partir desquels le comportement cyclique est tronqué. Ce test détecte l'implantation non conforme, décrite par le graphe de refus de la figure 9.4.

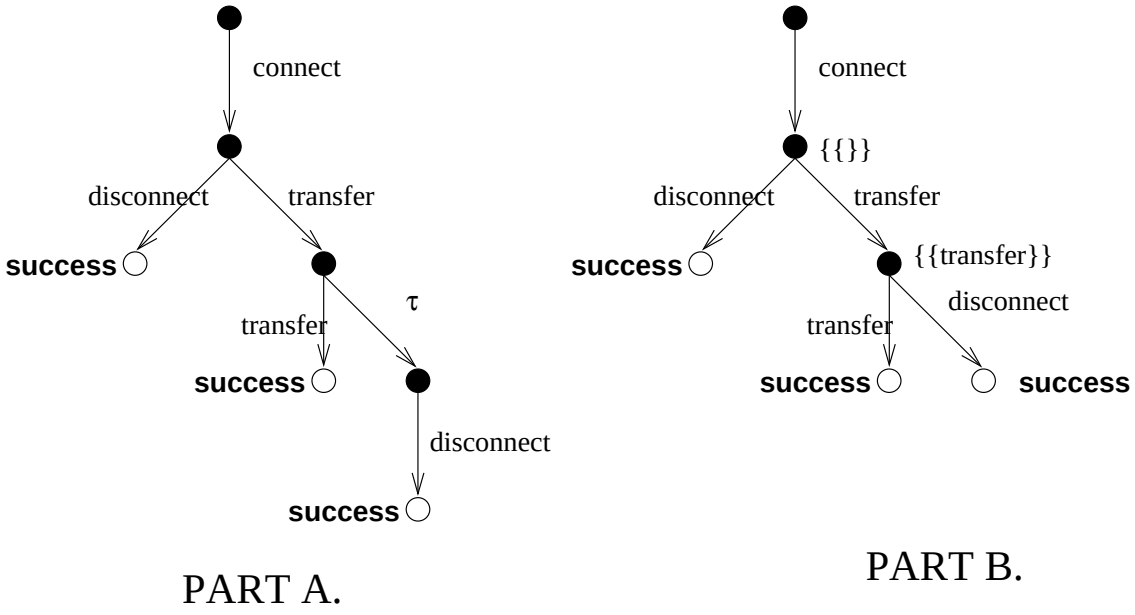


FIG. 9.5 – *Exemple de tests automatiquement déduits de la spécification*

9.4 La conformité évaluée à travers un environnement de test

La testabilité des composants d'un système communicant est usuellement évaluée sur des critères soit purement architecturaux soit purement comportementaux. Dans le premier cas, il est tenu compte de la façon d'interconnecter les composants et de l'existence de points d'observation de la communication[5]. Dans le second cas, les conditions de test d'un composant à travers son environnement ne sont pas envisagées. Les analyses architecturale et comportementale produisent des résultats partiels si elles sont effectuées séparément.

La technique d'analyse de testabilité proposée ici tient compte de critères architecturaux mais aussi de la dynamique du système sous test fournie par le graphe de refus. L'évaluation des résultats de la validation tient compte de l'environnement et des limites d'observation et de contrôle des échanges du système (Fig. 9.6). Le degré de confiance dans les résultats du test d'un système global est évalué automatiquement par rapport au test intrinsèque de chacun de ses composants, compte tenu de la définition de la conformité.

5. Une synthèse de ces méthodes dites structurelles ou a-priori, utilisées dans le domaine des composants électroniques, est présentée dans [Vuong et al., 1993]

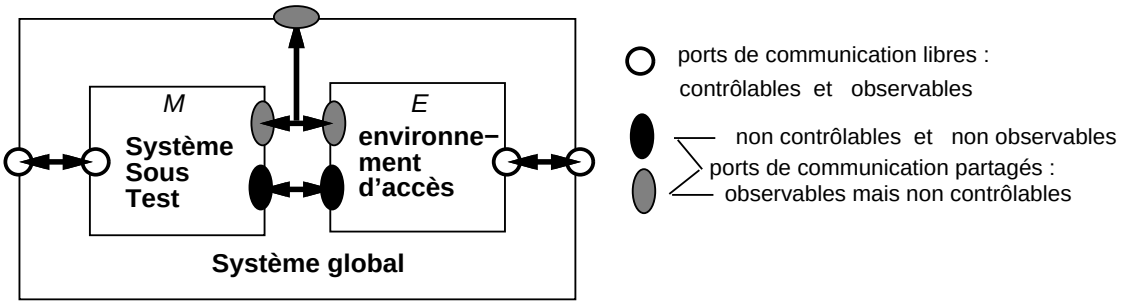


FIG. 9.6 – *Test contraint par la présence d'un environnement de test*

9.4.1 Problématique de la testabilité à travers un environnement

Tout d'abord, l'impact du manque d'accessibilité, d'observabilité et de commandabilité sur la testabilité d'un système, M, à travers un environnement, E, est expliqué informellement. Ensuite, la conformité à travers un environnement est définie.

La testabilité est la capacité de distinguer les implantations erronées d'un système M en testant le système contraint $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ I|[SynchAct]|E$. Où $SynchAct$ est la liste d'actions de synchronisation entre M et E (qui réduit l'accessibilité de M) et $NonObsSynchAct \subseteq SynchAct$ est la liste d'actions de communication non observables (qui réduit l'observabilité de M) et où I est une implantation (éventuellement erronée) de M.

La figure 9.7 illustre une dégradation dans le cas où la communication est obligatoirement non observable comme en CCS. D'après nos notations ceci correspond à : $SynchAct = NonObsSynchAct = \{\alpha, \beta\}$.

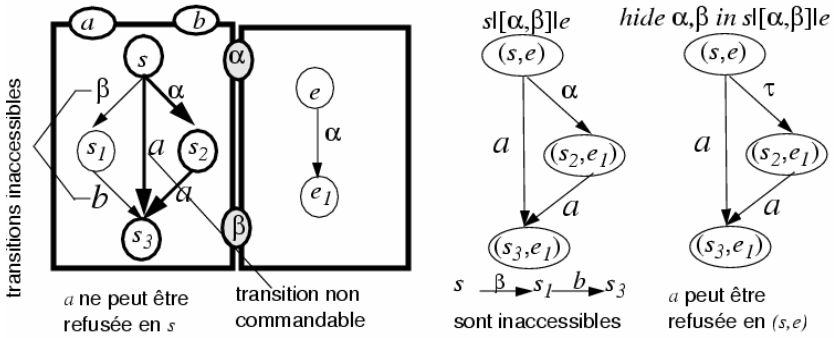


FIG. 9.7 – *Testabilité dégradée par la composition de systèmes*

La dégradation est inévitable en général, mais les règles suivantes peuvent la limiter.

loi d'accessibilité Une spécification ne doit pas contenir d'états ou de transitions inaccessibles, toutes les transitions doivent être activables à travers l'environnement. Cette loi vise à interdire toute sur-spécification des transitions de synchronisation (cf. le cas de sur-spécification de l'environnement, section 9.4.3).

loi de commandabilité Une spécification testable ne doit pas contenir de choix passif (i.e. externe)

entre une émission et une réception, ou bien
entre une action de synchronisation et une action libre.

loi d'observabilité Une synchronisation non observable doit être suivie par une transition observable. Cette règle exige en particulier une confirmation pour chaque requête.

L'analyse de testabilité tiendra automatiquement compte des contraintes d'accessibilité, d'observabilité et de commandabilité. La méthode est la suivante:

Le système, M, et son environnement, E, sont décrits par des graphes de refus. Le comportement de M contraint par E , ou de M dans E, est analysé. Les erreurs d'implémentation sont des perturbations de l'ensemble de refus d'un état. Une erreur n'est pas détectable à travers l'environnement si la substitution de l'ensemble de refus original par l'ensemble de refus perturbé n'est pas perceptible à travers l'environnement. Un ordonnancent partiel des ensembles de refus permet d'éviter l'analyse exhaustive de toutes les erreurs. De plus, cet ordonnancement définit la limite de testabilité comme le comportement erroné au delà duquel toute erreur est détectable.

9.4.2 Comportement du système contraint par son environnement

Les graphes de refus du système et de son environnement sont utilisés pour construire le comportement synchronisé sur les ports communs au système et à son environnement et libre sur les ports de communication libres. L'échange de message sur un port de synchronisation constitue une action de synchronisation. L'ensemble de ces actions sera noté $SynchAct$.

Le comportement global suit les règles suivantes :

(règle 1) *synchronisation* tout couple de transitions qui décrivent un échange de message m sur le port de communication PC entre le système et l'environnement, $s \overset{PC(m)}{\Longrightarrow} s'$, $e \overset{PC(m)}{\Longrightarrow} e'$, génère une transition $(s,e) \overset{PC(m)}{\Longrightarrow} (s',e')$.

(règle 2) *évolution libre* Le système et l'environnement sont respectivement dans les états s et e, la réception ou l'émission d'un message m sur un port de communication libre PS du système, à l'état s: $s \overset{PS(m)}{\Longrightarrow} s'$ génère la transition $(s,e) \overset{PS(m)}{\Longrightarrow} (s',e)$ dans la description du comportement global; de même $(s,e) \overset{PE(m)}{\Longrightarrow} (s,e')$

Selon la première règle, le système composé peut refuser les actions d'un ensemble A dans l'état (s,e) si cet ensemble A représente des actions de synchronisation qui peuvent être refusées dans l'un **ou** l'autre des états, s ou e, des graphes composants. Soit : si $A \in Ref(s)$ et $B \in Ref(e)$ alors $(A \cap SynchAct \cup B \cap SynchAct) \in Ref(s,e)$.

Selon la seconde règle, le système composé peut refuser les actions d'un ensemble A dans l'état (s,e) si cet ensemble A représente des actions qui peuvent être refusées dans l'un **et** l'autre des états, s ou e, des graphes composants. Soit : si $A \in Ref(s)$ et $B \in Ref(e)$ alors $(A \cap B) \in Ref(s,e)$.

Enfin, l'état (s,e) du graphe composé est déclaré divergent si l'un ou l'autre des états s ou e est divergent. Soit : $Div(s,e) = Div(s)$ ou $Div(e)$.

Les actions $SynchAct$ se décomposent en actions ni observables ni contrôlables, représentées par $NonObsSynchAct$, et celles observables mais non contrôlables représentées par $SynchAct \setminus NonObsSynchAct$. Le comportement du système final, notée à la LOTOS $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M|[SynchAct]|E$, s'obtient en « effaçant » toutes les actions non observables,

par leur renommage avec une seule étiquette, (τ en CCS et i en LOTOS). Dans le cas des graphes de refus, cette opération nécessite une étape de déterminisation. Elle consiste à regrouper les états accessibles par des transitions non observables et à recalculer les refus introduits par les évolutions non observables. Dans le comportement du système, seul le changement d'état causé par une communication est perceptible par le testeur [6].

La déterminisation du graphe de refus après l'effacement des actions non observables regroupe des états du graphe de refus initial. Lors du regroupement, le prédicat de divergence et les ensembles de refus du graphe restreint sont calculés.

Un état du graphe restreint diverge si l'un des états qu'il contient diverge ou si l'effacement crée un cycle d'actions non observables entre les états regroupés.

Le système refuse après une séquence un ensemble d'actions si l'observation de cette séquence autorise l'accès à un état « de refus » qui ne peut, lui ni aucun de ses successeurs par des transitions non observables, exécuter une des actions. Une condition plus forte permet de décider facilement ce refus. La notion d'état « de refus » est remplacée par celle d'état « stable », qui n'autorise aucune transition non observable et qui refuse toutes les actions de l'ensemble dit refusé. En absence de divergence, les deux conditions coïncident.

En cas de divergence, les deux calculs ne sont pas identiques. La figure 9.8 est un contre-exemple. Après la séquence « a », l'ensemble $\{b\}$ peut être refusé par un état de refus, ici l'état 4. Toutefois, il n'y a aucun état stable après la séquence a qui refuse b.

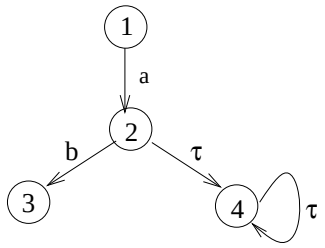


FIG. 9.8 – *En cas de divergence, la stabilité ne permet pas le calcul des refus.*

Le calcul des ensembles de refus après effacement utilisera la stabilité, même en présence de divergence. Ce choix est justifié d'une part par le fait que l'analyse de testabilité assimilera la divergence au blocage. D'autre part, il est connu d'après les travaux de Leduc [Leduc, 1991] que la représentation du comportement par un modèle de refus ne permet pas de calculer les ensembles de refus après une opération d'effacement qui crée une divergence.

Par contre les ensembles de refus sont parfaitement suffisants pour retrouver la notion de stabilité des états du système de transitions. La stabilité concerne alors les actions effacées [7]. Elle est vérifiée par tous les états internes qui n'exécutent aucune des actions effacées. L'ensemble de refus d'un état ($g_{\text{restreint}}$) du graphe restreint est alors formé de tous les ensembles de refus des états stables (g) qu'il regroupe. Cet ensemble est donné par la formule : $Ref(g_{\text{restreint}}) = \{X \in$

6. Cette hypothèse introduite par Milner est largement utilisée. Un changement d'état causé par une communication peut être précédé et suivi par une séquence quelconque d'évolutions internes sans que ceci ne soit perceptible par l'extérieur. La relation utilisée pour la conformité est alors $p \stackrel{c}{\Rightarrow} q \equiv_{def} p \xrightarrow{\tau^*} \xrightarrow{c} \xrightarrow{\tau^*} q$.

7. actions qui sont renommées τ, l'action non observables, lorsqu'il s'agit d'effacement dans les systèmes de transitions.

$Ref(g), \text{ActionsEffacées} \subseteq X, g \in g_{\text{restreint}}\}$. Dans cette formule, la condition « $ActionsEffacées \subseteq X$ » exprime la stabilité des états internes g considérés: toutes les actions effacées sont refusées.

9.4.3 Dégradation de la testabilité

Le comportement du système connecté à son environnement sert de référence pour juger si un test a été réussi ou non. La conformité d'une implantation, I, à travers un environnement, E, est définie par comparaison du comportement du système $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ I|[SynchAct]|E$ et du comportement de sa spécification, $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M\ |[SynchAct]|E$. La relation de conformité précédente conduit alors à :

$$
\begin{array}{c}
M\ \underline{\mathbf{E\text{–}conf}}\ I \equiv_{\text{def}} \\
(\ \mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ I|[SynchAct]|E) \\
\underline{\mathbf{conf}} \\
(\ \mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M|[SynchAct]|E)
\end{array}
$$

Cette définition de la conformité à travers un environnement permet d'établir des preuves sur la dégradation des tests lorsque leur mise en œuvre est contrainte par un environnement qui réduit le contrôle et l'observation du système sous test. Les systèmes qui présentent des comportements divergents peuvent être étudiés. De plus, comme il n'existe pas d'implantations non conformes qui ne soient intrinsèquement détectables par la procédure de test, cette dégradation ne peut pas être attribuée à une quelconque faiblesse de la procédure de test choisie. La dégradation de la testabilité est due à l'existence d'implémentations erronées, non détectables à cause de l'environnement.

La détection d'erreurs par le test à travers un environnement est au mieux celle du test sans les contraintes de l'environnement; la dégradation peut être réelle (cf. figure 9.7) : ce qui était détecté ne l'est plus.

Dans le cas où l'environnement est sur-spécifié par rapport au système auquel il est connecté, le test peut s'améliorer. L'environnement modifie les hypothèses de conformité. La conformité autorise l'implantation à étendre le comportement de la spécification (i.e. les traces de celle ci). Pour détecter ces implantations, le test de conformité, qui consiste à vérifier que le système ne fait pas moins que ce qui est exigé, ne suffit plus. Un test de robustesse est requis [8] qui vérifie que l'implantation ne réalise pas « plus que » ce qui est spécifié. En cas de sur-spécification de l'environnement, des implantations qui étendent la spécification peuvent être déclarées erronées.

La figure 9.9 illustre cette situation. L'implémentation I étend le comportement de la spécification M, et reste conforme car l'action α à partir de l'état initial n'entraîne pas de refus supplémentaire. Après composition avec l'environnement E et restriction (ou effacement) des actions α et β l'implémentation I dans l'environnement E offre, à partir de l'état initial $(1,1)$, une transition non observable τ qui conduit à l'état $(4,1)$ à partir duquel l'action a n'est plus possible. Ceci se traduit sur le graphe de refus par la présence, dans l'état initial, du refus $\{a\}$. Ce comportement n'est pas conforme à la spécification de M dans E, qui autorise cette action à partir de l'état initial.

8. Une implémentation I est robuste par rapport à sa spécification S si elle est conforme et si elle n'augmente pas les traces de cette spécification : $I\ \underline{\mathbf{red}}\ S \equiv_{def} I\ \underline{\mathbf{conf}}\ S\ et\ Tr(I) \subseteq Tr(S)$. La relation de réduction $\underline{\mathbf{red}}$ a été introduite dans [Brinksma et al., 1987].

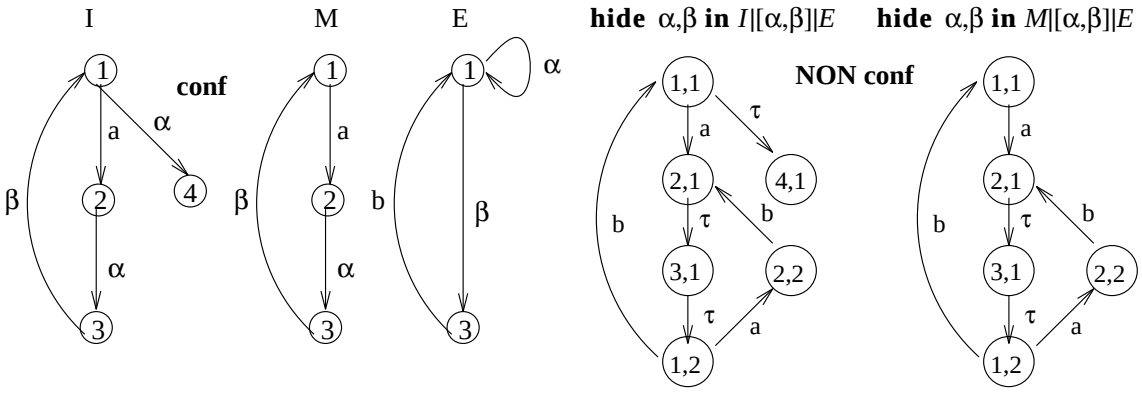


FIG. 9.9 – *Sur-spécification de l'environnement: la conformité n'est plus préservée*

Pour éviter ce cas, les traces de l'environnement relatives aux action de synchronisation ($Tr($ **hide** $(L \backslash SynchAct)$ **in** $E)$) doivent se synchroniser avec le système sous test. Autrement dit il faut que ces traces soient égales à celles du système : $Tr($ **hide** $(L \backslash SynchAct)$ **in** $E) = Tr($ **hide** $(L \backslash SynchAct)$ **in** $M)$.

La proposition suivante montre que toute implantation conforme qui n'augmente pas les traces de la spécification, sera jugée conforme à travers n'importe quel environnement des test et même en présence de divergence.

Préservation de la conformité : Si $Tr(I) \subseteq Tr(M)$ et I $\underline{\textbf{conf}}$ M alors pour tout E, on a : I $\underline{\textbf{E–conf}}$ M.

Lue dans le sens de la contraposée, cette proposition nous assure que lorsqu'une implantation est jugée non conforme par le test à travers un environnement, elle est soit réellement non conforme soit elle augmente les traces de la spécification. Dans les deux cas, ceci signifie qu'elle n'est pas robuste.

9.4.4 Analyse de la testabilité

L'analyse de la testabilité consiste à identifier les implantations non conformes mais non détectables lors du test à travers l'environnement. Même lorsque le système global contient un état divergent, l'analyse de la testabilité peut être fortement affinée grâce à la notion de limites de dégradation de la testabilité. Ceci n'est pas le cas si la divergence est ignorée ([Brinksma et al., 1987, Brinksma, 1988]) ou traitée différemment ([Leduc, 1991]).

Cette approche est basée sur un ordonnancement des ensembles de refus construits à partir des actions possibles depuis un état donné. Pour la recherche des limites de testabilité, la spécification est perturbée de manière ordonnée.

La figure 9.10 illustre, en lignes discontinues, l'ordonnancement des ensembles de refus construits sur l'ensembles d'actions $\{transfer, disconnect\}$, ainsi que les modèles ordonnées des implantations erronées. L'exemple choisi est la description simplifiée du comportement d'un protocole de transfert de données orienté connexion décrit au début de cet article. Le graphe de refus de ce modèle a comme ensemble de refus $\{transfer\}$ à l'état 2. Les implantations erronées sont

– les « distorsions », obtenues en remplaçant l'ensemble de refus par un autre qui lui est incomparable, qui refusent d'autres actions que celles qui sont refusées par la spécification.

Dans cet exemple la distorsion inverse le rôle de $transfer$ et $disconnect$ dans l'ensemble de refus de l'état 2. L'implantation erronée au lieu de refuser le transfert de données, peut refuser à l'utilisateur de se déconnecter avant le transfert.

– les « dégradations », obtenues en remplaçant l'ensemble de refus par un autre plus large, qui sont des implantations strictement moins déterministes que la spécification. C'est à dire qui refusent plus d'actions que la spécification. La première dégradation relative à notre exemple décrit le comportement d'un système qui peut refuser tantôt le transfert tantôt la déconnexion. La seconde dégradation décrit le comportement d'une implantation qui peut refuser en plus du transfert, la déconnexion et bloquer ainsi totalement son utilisateur.

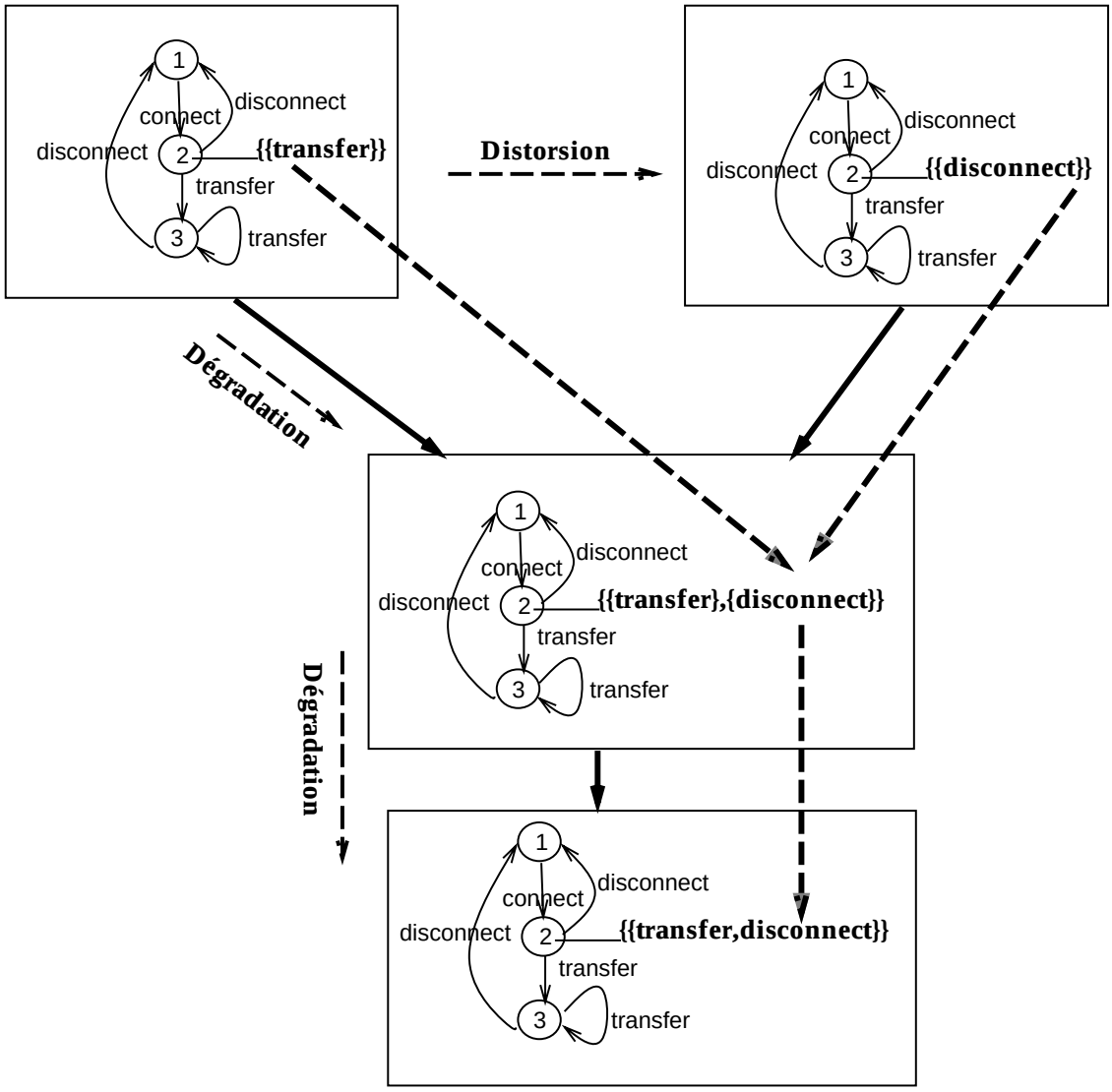
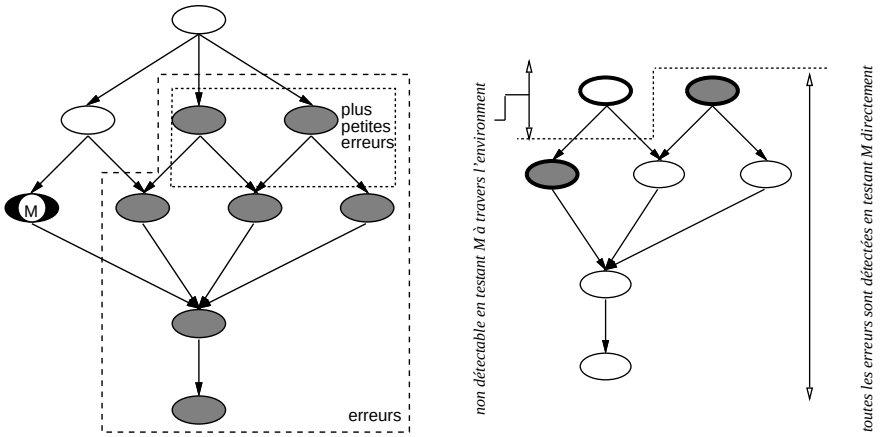


FIG. 9.10 – *ordonnancement des ensembles de refus et des implantations*

Les implantations sont classées des moins erronées aux plus erronées (Figure 9.11.a). Ces implantations représentent des erreurs détectables par le test de conformité du système-sous-test,

mais, par définition, seulement s'il était isolé. Des opérations de substitution et de comparaison, décrites dans la prochaine section, sur les ensembles de refus du système et de son environnement permettent de décider si une implantation erronée sera détectée lors du test de ce système à travers l'environnement. Ceci est rendu possible grâce aux opérateurs de composition et de restriction de graphes de refus définis en §9.4.2.

Les implantations les moins erronées qui restent détectables dans l'environnement sont les limites de la testabilité (Figure 9.11.b).



(a) *ensemble ordonné d'implantations* (b) *Limites de la testabilité*

FIG. 9.11 – *Les ovales grisés représentent dans la partie (a), les implantations erronées et dans la partie (b), les limites de testabilité.*

Un algorithme [45Ci] a été proposé pour la recherche ordonnée de ces limites. Nous proposons ici de le généraliser pour tenir compte des cas où la restriction crée des états divergents dans la description du comportement du système contraint par l'environnement.

9.4.4.1 Algorithme généralisé de l'analyse de testabilité

Le principe de l'algorithme consiste à perturber l'ensemble de refus d'un état quelconque et à vérifier si cette perturbation est ou non détectée.

Pour tout état i du graphe de refus, notons (G, V) le graphe représentant, selon la convention de Hasse, l'ensemble de tous les ensembles refus possibles pour l'état i, $(\mathcal{P}(\mathcal{P}(ActionsPossiblesDans(i))))$, ordonné par la relation d'ordre partiel $\Subset$ qui définit l'inclusion pour les ensembles de refus. Où $\mathcal{P}(\bullet)$ désigne l'ensemble des parties d'un ensemble donné. $ActionsPossiblesDans(i)$ désigne l'ensemble des actions possibles à partir de l'état i du graphe de refus de la spécification. La partie minimale d'un ensemble (d'ensembles) est celle qui ne contient que des ensembles $\mathcal{X}$ qui vérifient la condition de minimalité suivante : $A, B \in \mathcal{X}$ et $A \subseteq B$ impliquent $A = B$. Et enfin la relation $\Subset$ décrit l'inclusion des ensembles de refus et est définie par : $\mathcal{X} \Subset \mathcal{Y} \equiv_{\text{def}} \forall A \in \mathcal{X} \exists B \in \mathcal{Y} : A \subseteq Y$.

Les points suivants explicitent les notations de la représentation de l'ensemble ordonné selon la convention de Hasse.

• G est un ensemble de noeuds représentant les éléments d de l'ensemble ordonné que l'on vient de décrire.

Ce sont les ensembles de refus qui peuvent remplacer l'ensemble de refus $Ref(i)$ noté e_i.

• V est un ensemble d'arcs représentant l'inclusion stricte de 2 ensembles de refus.

• (G, V^*) est la fermeture transitive de (G, V).

Pour tout couple de noeuds distincts (e, e'),

• $(e, e') \in V$ signifie : $e \subset e'$ et $\nexists e''$ tel que $e \subset_s e'' \subset_s e'$

• $(e, e') \in V^*$ signifie : $\exists e_1, .., e_k \ : e \subset_s e_1 \subset_s \cdots \subset_s e_k \subset_s e'$

Pour $\mathcal{E} = \{e_1, e_2, .., e_n\} \subseteq G$, **inf**$(\mathcal{E})$ désigne l'ensemble $\{e_k \in \mathcal{E} \ : \ \nexists e \in \mathcal{E} \ : \ (e, e_k) \in V\}$

La recherche des limites de dégradations et distorsions non détectables à travers l'environnement peut se résumer par l'algorithme qui calcule **Limits**, décrit dans la table 9.2, et dans lequel **VerifyBySubstitution**(i,e) est la procédure qui retourne :

- **VRAI** s'il existe un état j de $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M|[SynchAct]|E$ qui n'est pas divergent **et** pour lequel si l'on substitue e à $Ref_M(i)$ dans le calcul de $Ref_E^M(j)$, on trouve un ensemble r tel que $r \not\subset Ref_E^M(j)$.

- **FAUX** sinon.

Où $Ref_M(i)$ désigne l'ensemble de refus associé à l'état i dans le graphe de refus du système M ; $Ref_E^M(j)$ désigne l'ensemble de refus associé à l'état j dans le graphe de refus de la restriction du graphe de refus obtenu par composition du système M et de l'environnement E, c'est à dire le graphe de refus noté
$\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M|[SynchAct]|E$.
Et où j est un état (de $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M|[SynchAct]|E$) qui contient un couple de la forme $(i, *)$, où « $*$ » désigne n'importe quel état de l'environnement E.

Nous proposons maintenant une description informelle de l'algorithme d'analyse de testabilité.

Notons $Deg(i) = successeurs(Ref_M(i))$, $Imv(i) = predcesseurs(Ref_M(i))$, $Dis(i) = \mathcal{T} \setminus (Deg(i) \cup Imv(i) \cup \{Ref_M(i)\})$, et $Err(i) = Deg(i) \cup Dis(i)$

Perturber la spécification M consiste à :

- choisir un état, i, du graphe de refus

- remplacer l'ensemble de refus de cet état, $Ref_M(i)$, par l'une des bornes inférieures des dégradations et distorsions : $A \in \mathbf{inf}(Err(i))$

Vérifier si cette perturbation est détectable par le test de conformité à travers E consiste à :

- chercher un état de $\mathbf{hide}\ NonObsSynchAct\ \mathbf{in}\ M|[SynchAct]|E$ qui pourrait être « perturbé » par la substitution. C'est à dire un état, j, qui est non divergent et qui contient un couple de la forme $(i, *)$, où $*$ désigne un état quelconque de E.

- calculer le nouvel ensemble de refus résultant en remplaçant dans le calcul de l'ensemble de refus de j, $Ref_M(i)$ par A.

Distorsions := $G \setminus \{e_i\} \setminus \{e \ : (e, e_i) \in V^*$ or $(e_i, e) \in V^*\}$
Degradations := $\{e \ : (e_i, e) \in V^*\}$
OrderedErrors := Distorsions $\cup$ Degradations
Limits := Ø
PotLimits:= **inf**(OrderedErrors)
while PotLimits $\neq$ Ø **do**
  forall $e \in$ PotLimits **do**
    detected(e) := **VerifyBySubstitution**(i,e)
    if detected(e) **then**
      OrderedErrors:= OrderedErrors$\setminus\{e\}\setminus \{e_k \ : (e, e_k) \in V^*\}$
      Limits := Limits $\cup\{e\}$
    else
      OrderedErrors:= OrderedErrors$\setminus\{e\}$
    endif
  done
  PotLimits:=**inf**(OrderedErrors)
done

TAB. 9.2 – *Algorithme de recherche des limites*

– vérifier si l'ensemble obtenu est contenu ou non (au sens de $\mathbb{C}$) dans l'ancien ensemble. S'il n'est pas contenu, on arrête l'exploration. La perturbation est détectable (ainsi que toutes ses dégradations, c'est à dire ses successeurs dans le graphe). Le nouvel ensemble de perturbations à explorer sera $Err(i) \setminus (successeurs(A) \cup \{A\})$.
  Dans le cas contraire (A non détectable pour tous les états s) seul A est retiré de l'ensemble des perturbations à explorer.

9.4.5 Validité de l'approche d'analyse

Nous proposons dans cette section, d'examiner la validité des solutions calculées par l'approche d'analyse de testabilité ainsi que la couverture du modèle de faute utilisé.

9.4.5.1 Exactitude des limites calculées

Nous proposons d'abord de définir de façon générale la notion de limite de testabilité. Nous nous baserons ensuite sur cette définition pour évaluer l'exactitude des limites telles qu'elles sont calculées dans notre approche.

Définition 9.1 (Limites de testabilité) *Une implémentation erronée, I, est une limite de détection de non conformité si :*
(i) *La non conformité de I est détectée (à travers l'environnement).*

(ii) *La non conformité d'implémentations plus erronées au sens large (i.e. $I' \leq I$) est détectable à travers l'environnement.*
(iii) *La non conformité d'implémentations (strictement) moins erronées (i.e. $I' > I$) n'est pas détectable à travers l'environnement.*

La relation $<$ (plus erroné que) est définie à partir de la relation inverse $\geq$ (plus déterministe que) qui à son tour est définie à partir de la relation de conformité par : $p \geq q \equiv_{def} p \underline{\mathbf{conf}}\ q\ et\ Tr(p) = Tr(q)$. C'est à dire p est une implémentation conforme à q et qui implémente toutes, et seulement toutes, les séquences de q (notées $Tr(q)$).

Toute implémentation plus erronée qu'une implémentation erronée détectable (en particulier une limite) est aussi détectable [1T].

Proposition 9.1 (détection de non conformité)
Etant données deux implémentations, I et I', d'une spécification M
si $(I \neg \underline{\mathbf{E\text{–}conf}}\ M)$ *alors* $(I' \leq I) \Rightarrow (I' \neg \underline{\mathbf{E\text{–}conf}}\ M)$ ■

La clause (i) est vérifiée par construction de la limite. La condition d'arrêt de l'algorithme d'analyse de testabilité vérifie que la solution proposée comme limite est effectivement détectable. La proposition 9.1 assure la validité de la deuxième clause (ii).

Dans le cas général la validité de l'assertion (iii) est vérifiée sous réserve qu'une erreur n'augmente pas le nombre d'états de l'implémentation. Sous cette hypothèse, le modèle de fautes utilisé est exhaustif en ce sens qu'entre deux implémentations erronées (au sens de la relation $<$) figurant parmi les implémentations analysées, il ne peut pas exister d'implémentation erronée ayant le même nombre d'états que la spécification[9]. Sans cette hypothèse, pour toute limite, il est possible de construire un modèle d'implémentation strictement moins erronée mais détectable. Il suffit de dupliquer un état qui est atteint par deux séquences et de réduire l'ensemble de refus de l'un des deux états.

La figure 9.12 illustre l'obtention d'un tel modèle. A partir de l'implémentation (erronée) I, une implémentation strictement moins erronée $I' > I$, est construite par duplication de l'état i et en choisissant comme ensemble de refus de l'état créé i', un prédécesseur immédiat de $Ref(i)$ dans le treillis des ensembles de refus (un modèle par prédécesseur immédiat est alors possible). Un autre modèle, I'' est obtenu en interchangeant les ensembles de refus de i et i'. Le nombre d'implémentations intermédiaires que l'on peut construire augmente avec le nombre de séquences qui arrivent vers l'état concerné par l'analyse.

En supposant que I est une limite de testabilité, les deux implémentations I' et I'' qui bien que strictement moins erronées que I pourraient être détectées par le test de conformité. L'une, au moins, d'entre elles doit remplacer I dans l'ensemble des limites de testabilité si l'on admet l'augmentation du nombre d'états.

En admettant par exemple qu'une erreur peut augmenter d'un état le graphe de refus de la spécification alors le « quadrillage » de l'espace des implémentations n'est plus isomorphe au treillis des ensembles de refus. L'ensemble des implémentations simulées n'est plus alors suffisant pour trouver les limites qui vérifient l'assertion (iii) de la définition 9.1. La limite de testabilité doit en effet être fournie en fonction des modèles obtenus sous la nouvelle hypothèse de test.

9. En considérant que les implémentations et la spécification sont décrites par des graphes de refus minimaux, qui ne contiennent pas d'état équivalent à un autre état au sens de la relation d'équivalence « $\leq \cap \geq$ ») d'un autre état.

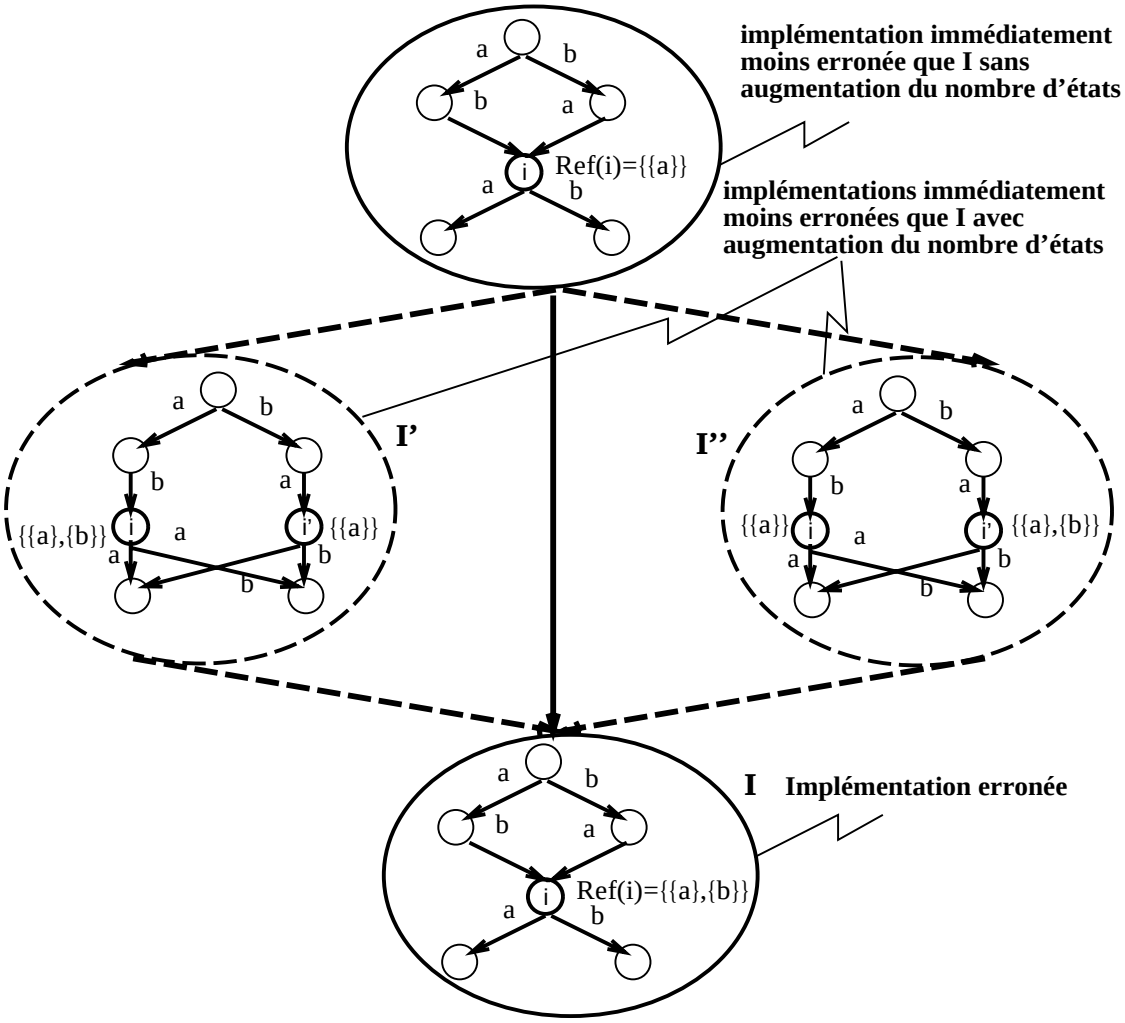


FIG. 9.12 – *Implémentations erronées obtenues par duplication d'un état et en changeant son ensemble de refus.*

Dans une spécification qui contient des cycles, on peut indéfiniment augmenter le nombe d'état d'une implémentation erronée, tout en gardant ses traces et tout en réduisant les refus associés à ces traces. Sans l'hypothèse de test que nous avons retenue, les limites de testabilité n'existeraient pas au sens de l'assertion (iii). Il est intéressant de souligner ici que ceci explique pourquoi nous préférons parler de « *limites de dégradation de la testabilité* » pour rappeler que les solution calculées par notre approche sont valides vis-à-vis de l'assertion (ii) quelle que soit l'hypothèse que l'on associe à l'implémentation erronée.

9.4.5.2 Couverture du modèle de fautes

Dans ce paragraphe, l'exhaustivité de la méthode est analysée dans le contexte classique des modèles de fautes qui (toujours sous l'hypothèse qu'une erreur n'augmente pas le nombre d'états) peuvent se réduire à trois types d'erreurs d'implémentation : « erreur de transfert », « omission de transition », ou « erreur de codage ».

La comparaison d'implémentations basée sur la relation $\geq$ restreint le modèle d'erreurs aux implémentations qui ont les mêmes traces que la spécification. Mais ceci ne nous empêche pas de juger la testabilité d'implémentations erronées ayant moins de traces que la spécification. En fait, ce qu'on appelle « omission de transitions » dans les modèles de fautes basés sur les machines à états finis est un cas particulier qui est pris en compte par le graphe de refus. En effet la notion de refus est utilisée au sens le plus large du terme. C'est à dire « refuser toujours » (ou omettre d'implémenter) une action peut aboutir au même verdict de testabilité (et conformité) que « refuser parfois » ou « pouvoir refuser ». Ainsi (comme l'illustre la figure 9.13), dans le modèle décrivant une implémentation erronée (notée I), détruire toute transition $i \stackrel{a}{\Rightarrow} j$ étiquetée par une action a figurant dans tous les ensembles de refus de l'état i (i.e. $a \in X, \forall X \in Ref(i)$) donnerait une implémentation erronée (notée I') ayant moins de traces (ou transitions) que la spécification et qui ne sera pas distinguée par le test de conformité de l'implémentation erronée de laquelle elle est déduite. Elle réussira en effet tous les test que doit réussir celle-ci. Ceci se démontre facilement en remarquant que la transformation décrite préserve les ensembles de refus et par conséquence la conformité : $I' \ \underline{\mathbf{conf}} \ I$ et $I \ \underline{\mathbf{conf}} \ I'$.

En conséquence, nous pouvons dire qu'une omission de transition « a » à partir de l'état i n'est pas détectable à travers l'environnement si l'ensemble de refus $\{X \cup \{a\}, X \in Ref(i)\}$ se trouve au delà des limites de testabilité. De façon équivalente l'omission de « a » n'est pas détectable si la substitution de $Ref(i)$ par $\{X \cup \{a\}, X \in Ref(i)\}$ n'est pas détectable à travers l'environnement.

Les erreurs dites de transfert sont couvertes par le modèle de fautes utilisé dans notre approche sous la même hypothèse que nous avons posée précédemment. Si la transition $i \stackrel{a}{\rightarrow} j$ est implémentée par $i \stackrel{a}{\rightarrow} k$ et $k \neq j$ alors, sous l'hypothèse de minimalité du graphe de refus, deux cas sont possibles. Dans le premier cas, $Ref(i)$ est modifié par cette erreur de transfert. L'erreur est alors détectée si l'ensemble de refus de l'état k est au delà des limites de testabilité. Le second cas suppose que l'ensemble de refus est inchangé malgré l'erreur de transfert (i.e. $Ref(j) = Ref(k)$). La minimalité du graphe de refus implique alors qu'il existe une séquence d'actions $\sigma = a_1.a_2 \cdots a_n$ que chacun des deux états peut exécuter mais qui aboutissent à des états (k' et j') qui n'ont pas les même ensembles de refus. La possibilité de détection de cette erreur dépendra alors de la position

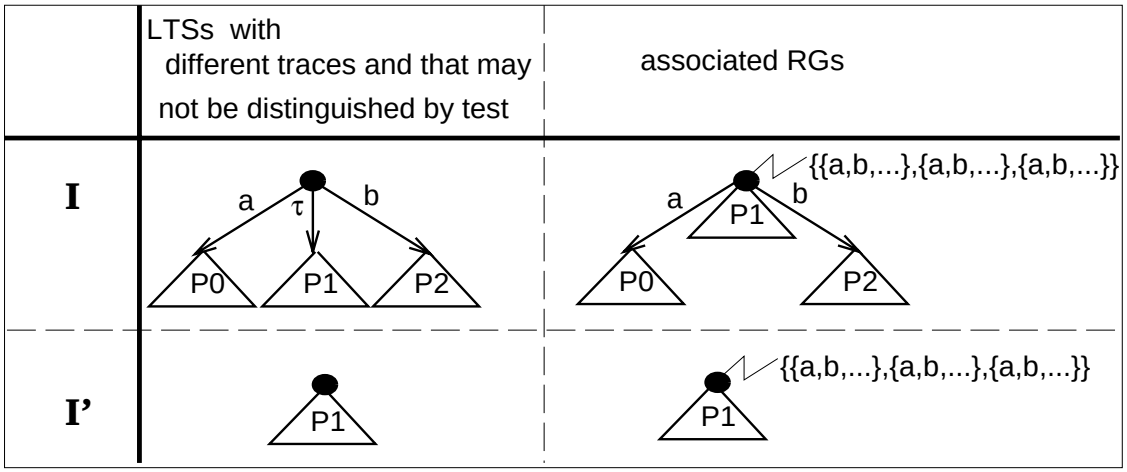


FIG. 9.13 – *Détruire les transitions associées aux actions a et b ne change pas le verdict de testabilité qui concerne I. Dans cette figure P_0, P_1 et P_2 représentent des états où seules des actions observables sont présentes.*

de l'ensemble de refus de l'état atteint par erreur (k') par rapport à celui de l'état que l'implémentation aurait du atteindre (j').

Les erreurs de codage peuvent concerner une erreur dans l'ensemble de refus d'un état, ce que l'analyse considère. Elles peuvent aussi concerner « l'oubli » d'un état ce qui est équivalent à l'omission de toutes les transitions que l'état oublié devait offrir. La discussion de la couverture de cette erreur est alors identique au cas d'omission de transition.

9.5 Conclusion

Ce chapitre a présenté une nouvelle structure de représentation des comportements abstraits de systèmes de communication. Cette représentation tient compte de la présence de comportements dits divergents qui représentent la possibilité d'une évolution permanente du système sous test sans communication avec l'extérieur. En particulier, la technique proposée d'analyse de testabilité s'appuie sur un modèle de fautes partiellement ordonnées qui permet la définition de limite de testabilité et la recherche sélective de ces limites.

Un environnement logiciel a été réalisé [7]. Il implante en langage C, l'algorithme d'analyse de testabilité et les opérations nécessaires, à savoir les transformations entre systèmes de transitions et graphes de refus ainsi que la composition et la restriction des graphes de refus. Des options d'exécution permettent de choisir, entre autres, la limite du nombre d'actions possibles dans la construction des ensembles de refus. En sortie, les états du graphe de refus qui présentent des problèmes de testabilité sont listés, et un modèle « système de transitions » de chaque limite de testabilité est fourni à la demande de l'utilisateur. Les états inaccessibles sont aussi énumérés. En entrée, seuls les noms des ports de synchronisation (décomposés en observables et non observables) sont demandés à l'utilisateur. Les systèmes de transitions du système et de l'environnement sont générés à partir de modèles LOTOS ou Réseaux de Petri Communicants. Un traitement supplémentaire permet de reconstituer l'alphabet de synchronisation et de restriction selon la description de règles

de composition et de restriction précises.

L'approche proposée repose sur un modèle ordonné de fautes isomorphe à l'ordre partiel des ensembles de refus construits sur l'ensemble des actions. L'analyse effectuée grâce à ce modèle fournit les limites « supérieures » de la testabilité, ou « limites de la dégradation de la testabilité », qui vérifient l'assertion suivante : toute implémentation plus erronée qu'une limite est détectable. Les limites calculées ne sont pas des limites « inférieures » dans le cas général : toute implémentation strictement moins erronée qu'une limite n'est pas forcément non détectable. Ceci n'est vrai que si l'on suppose qu'une erreur n'augmente pas le nombre d'états de la spécification. Les perspectives théoriques de ce travail peuvent viser la généralisation de l'approche d'analyse de testabilité par une solution algorithmique paramétrée par le nombre d'états qu'une erreur peut ajouter au modèle de la spécification.

Les perspectives d'ordre pratique de ce travail concernent l'expérimentation de cette approche. Cette expérimentation servirait à vérifier des perspectives d'ordre plus théorique sur le développement de règles de conception de bonne testabilité. Les perspectives plus immédiates de ce travail visent son extension aux interfaces asynchrones de test de conformité.

Table des figures

Liste des tableaux

Glossaire

ADL : Architecture Description Langage
CDK : Coordination Development toolKit
CSCL : Computer Supported Cooperative Learning
CSCW : Computer Supported Cooperative Work
DAI (IAD) : Distributed Artificial Intelligence
DSE : Distributed System Engineering
IETF : Internet Engineering Task Force
IHM : Interface Homme Machine
IMPP : Instant Messaging and Presence Protocol
ISO (OSI en français) : International Organization for Standardisation IUT : Implementation Under Test
LOTOS : Language Of Temporal Ordering Specification
RMS : Responsibility Management System
SID : Systèmes Informatiques Distribués
SMS : Session Management System

Table des matières