ELKAR - Ré-Ingénierie d'applications pour la mise en oeuvre de la coopération : Méthodologie et Architecture

Philippe Roose

N° d'ordre :

THÈSE

Présentée devant

l'UNIVERSITE DE PAU ET DES PAYS DE L'ADOUR

IUT de Bayonne – Pays Basque, Département Informatique

par

Philippe ROOSE

en vue de l'obtention du titre de

DOCTEUR EN SCIENCES DE L'UNIVERSITE DE PAU ET DES PAYS DE L'ADOUR

Spécialité : **INFORMATIQUE**

ELKAR

Ré-ingénierie d'applications pour la mise en œuvre de la coopération : méthodologie et architecture

---

Thèse soutenue le 14 décembre 2000

Composition du jury :

B. Causse	Professeur à l'IUT de Bayonne, Directeur de Recherche
J.P. Giraudin	Professeur à l'Université de Grenoble, Rapporteur
S. Miranda	Professeur à l'Université de Nice, Rapporteur
C. Chrisment	Professeur à l'université Paul Sabatier de Toulouse,
P. Aniorté	Maître de Conférences à l'IUT Informatique de Bayonne
M. Dalmau	Maître de Conférences à l'IUT Informatique de Bayonne

…

TABLE DES MATIERES

INDEX DES FIGURES

Index des Tableaux

Remerciements

Pour commencer, je tiens à profondément remercier l'ensemble des membres du jury :
À M. Jean-Pierre Giraudin, Professeur au laboratoire LSR-IMAG de Grenoble, pour l'honneur qu'il me fait d'être rapporteur de cette thèse. Je le remercie également pour ses remarques pertinentes qui m'ont permis d'améliorer mon travail.

À M. Serge Miranda, Directeur du DESS MBDS, Professeur au laboratoire I3S de Nice-Sophia Antipolis, envers qui je tiens à exprimer toute ma gratitude d'être également rapporteur de ce travail. Je souhaite aussi le remercier pour sa gentillesse et sa disponibilité depuis notre première rencontre au congrès SEID à Santiago de Compostella.

À M. Claude Chrisment, Professeur à l'IRIT de Toulouse, d'avoir accepté d'être au jury ainsi que pour les remarques qu'il ma soumises.

À M. Bernard Causse, Professeur au Laboratoire d'Informatique de l'UPPA, Directeur de l'IUT de Bayonne, pour présider ce jury et pour m'avoir fait confiance tout au long de ces trois années.

Enfin, les derniers mais pas les moindres comme disent les anglo-saxons, comment pouvoir remercier les deux personnes qui m'ont encadré : Philippe Aniorté et Marc Dalmau. Je pense qu'il n'y a pas de mots assez forts pour leur exprimer mes remerciements, alors tout simplement : **MERCI**.
Sans eux, cette thèse n'existerait pas. Je leur dois beaucoup et bien plus qu'ils n'osent l'imaginer. Ils ont été à l'origine de mon envie de poursuivre mes études après mon DUT d'informatique. Ils ont su me donner l'envie et le goût de leur travail. Je les retrouve à la conclusion de mes études. Il me semble que c'est une fin élégante…et j'espère, le début d'une autre aventure ! A toi Marc, une mention spéciale. Nous avons vécu ces trois dernières années dans le même bureau et avons passé, je pense, de très bons moments. Je te suis infiniment reconnaissant pour ce que tu es tout d'abord et pour tout ce que tu as fait pour moi…quelques lignes n'y suffiraient pas.

Je dois bien évidemment associer à cette thèse de nombreuses autres personnes en commençant par l'ensemble du personnel de l'IUT de Bayonne (IUT Darrigrand & Château Neuf). Je tiens tout particulièrement à remercier les CTI's boys qui font un travail de l'ombre important et sur lesquels on tombe toujours lorsque…le mail ne marche plus ! Je dois aussi remercier dans le désordre Manu, VC, Bernie, Zaza, Cathy, Pantxi, Marqua, Patchoc, Jacques (il y a du swell !), Sophie, Mitch, Jeanne-Marie, …pardon à ceux que j'oublie, la liste est tellement longue.

Mais une thèse n'est pas uniquement le travail d'un thésard. C'est un travail d'équipe. Aussi, chacune des personnes citées ci-dessus (mais aussi celles que j'ai oublié) ont contribué à ce résultat. Qu'elles en soient ici vivement remerciées, cette thèse est aussi la votre.

Afin de mener ce travail de longue haleine à son terme, il est nécessaire aussi de se sortir de cet environnement studieux pour se divertir. Un merci donc tout particulier aux Seb, Julot, Nieves, France, Fred, Kiki, David, Bonch, Wouew, Mat, Céline, Marianne, Anoulou, Christophe, Christian, Éric, Valérie… à l'équipe de rugby du B.O (! !)…pardon encore à ceux que j'oublie…Vous avez supporté, accepté mais aussi partagé mes sautes d'humeurs, mes crises, mes joies, mes peines, mes problèmes existentiels (! !), mes blagues douteuses… Vous avez contribué de manière indirecte mais tellement importante à la réussite de ce travail.

Et puis, je ne peux terminer sans donner un simple mais bigrement sincère MERCI à l'ensemble de ma famille qui m'a entouré, mené, malmené (!), encouragé, consolé, soutenu, … durant toutes mes études. Tout a été fait pour que j'étudie dans les meilleures conditions possibles. J'ai toujours été supporté dans ma passion de l'informatique. Merci à vous Maman, Mané, Mamie, Brigitte, Crapaud, Tom mon filleul.

MILESKER DENERI...

Enfin, je ne peux terminer cette thèse sans avoir une pensée profonde pour mon père trop tôt disparu. Cette thèse t'es dédié plus qu'à tout autre.

A mon père...

Introduction

1. Objectifs

De plus en plus d'applications sont constituées de modules distribués (un module correspond à une cellule de production, un automate, un composant logiciel, …) interconnectés par un réseau. Certes ces modules travaillent dans un but commun, toutefois on ne peut pas pour autant dire qu'ils coopèrent au sens où nous l'entendons. Ce type de coopération est fréquemment qualifiée de faiblement couplée.

On perçoit de manière grandissante la nécessité de faire migrer ces applications distribuées (souvent anciennes) non coopératives vers la coopération. En effet, les modules de telles applications ne savent pas se synchroniser ni échanger des informations ni surtout se constituer en groupes de travail (ensemble de modules ayant un but commun et formant une composante propre). On parvient généralement à établir des communications entre eux à l'aide d'opérateurs externes (humains ou machines dédiées de type automate ou autre – cf Figure 1), mais il est quasiment impossible d'avoir ainsi une organisation en groupes de travail capables d'évoluer dynamiquement dans le temps par l'intégration de nouveaux modules ou le départ de certains. Combien de fois ne rencontre-t-on pas dans de telles organisations des postes de travail dont tout ou partie de l'activité consiste en la saisie d'informations qui ont déjà été disponibles ou que l'on aurait pu synthétiser à un moment donné mais que l'on n'a pas su ou pas pu récupérer au moment opportun ?

Mais les responsables informatiques hésitent, voire même refusent toute profonde modification de ces applications qui tournent depuis longtemps, qui sont fiables et pour lesquelles le code n'est pas toujours modifiable pour diverses raisons : sources non maintenus ou perdus, compétences absentes, etc. La plupart du temps la seule concession réalisée est la greffe d'un réseau restant sous-exploité.

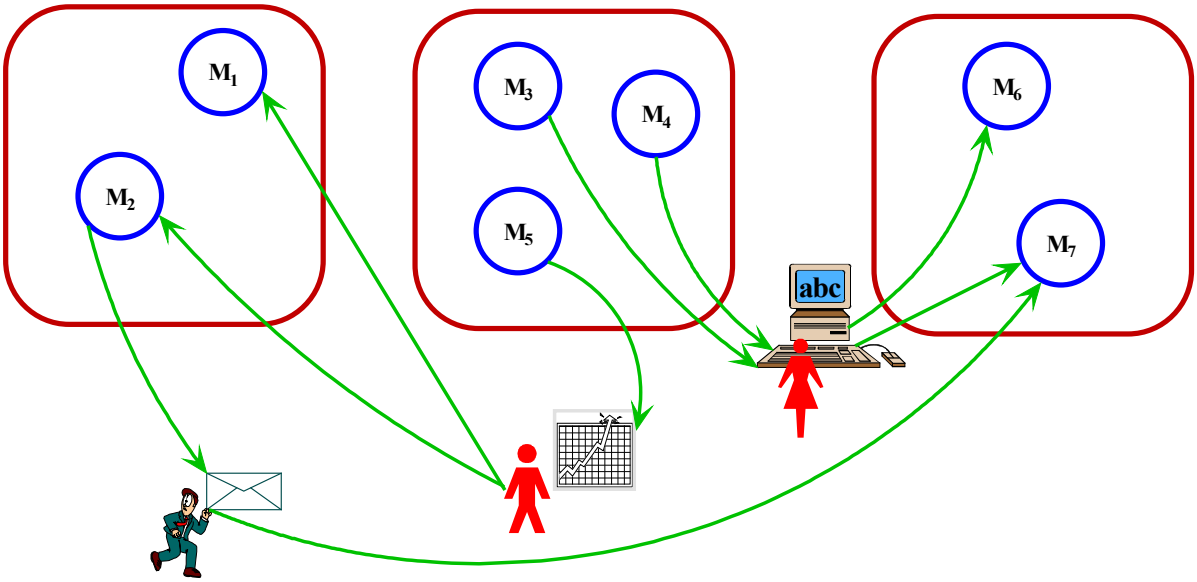


Figure 1 : Coopération par opérateurs externes

C'est ici que nous intervenons en tentant de faire disparaître tout ou partie de ces opérateurs externes assurant la coopération, c'est à dire, en gérant automatiquement la circulation des informations et l'organisation en groupes de travail dynamiques [Roose 00a] [Roose 00c]. Nous voyons sur le schéma suivant qu'après ré-ingénierie de l'application les informations, qui auparavant étaient véhiculées à l'aide d'opérateurs externes, circulent maintenant de manière automatique par le réseau allant directement des producteurs aux consommateurs.

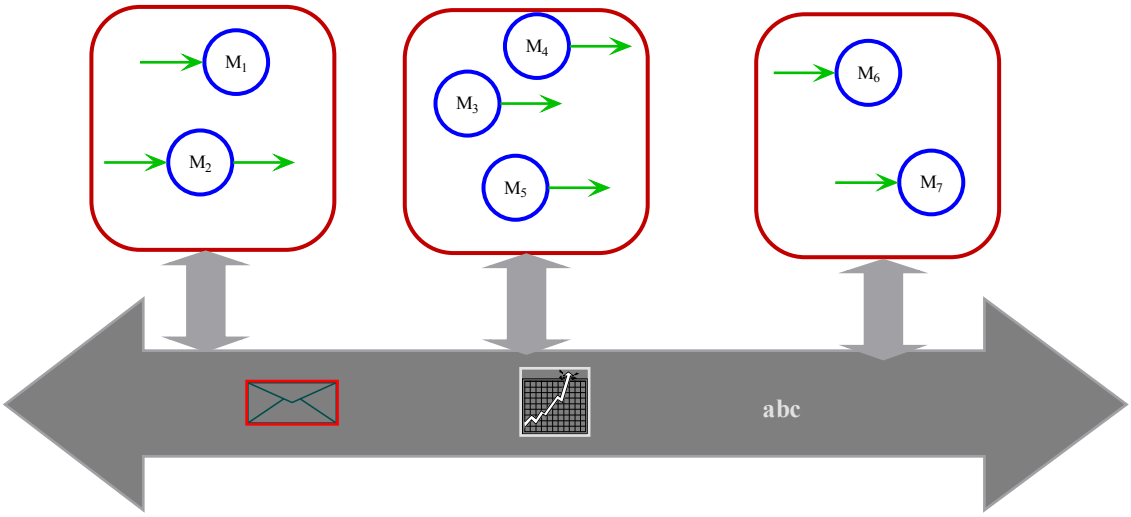


Figure 2 : Gestion automatique de la circulation des informations

Les nécessités modernes d'exploitation font de plus en plus appel à la flexibilité dans les systèmes de production [Martineau 96] [Martineau 97]. Ceci suppose que les divers modules puissent se constituer en groupes de travail évoluant au cours du temps (ordonnancement de production piloté par les commandes, insertion d'une production exceptionnelle dans une chaîne en cours, …). Une telle évolution du système ne se traduit pas par l'introduction de fonctionnalités nouvelles ou différentes dans les modules existants mais par la modification de leurs interactions.

Une mise en coopération sous entend une complète réorganisation de l'application. Dans notre méthode, nous considérons pour cela deux niveaux d'abstraction : le groupe de travail et le module opératoire et basons la coopération sur la constitution de groupes de travail et la mise en œuvre de la communication par le biais d'éléments de coopération (événements, messages et données). Les événements représentent des communications directes alors que les messages fonctionnent selon un mode différé tandis que les données sont persistantes.

Dans un souci de rendre opérationnelle notre méthode, nous proposons d'utiliser comme cible un ensemble d'outils basés sur le concept de règles ÉCA (Événement-Condition-Action) hérités des Bases de Données Actives (BDA) mais étendus aux applications coopératives distribuées. Cet ensemble d'outils a donné lieu à la réalisation d'une plate-forme inspirée du modèle d'exécution d'ADACTIF [Tawbi 96] que nous présenterons à la fin de cette introduction. Cette plate-forme aura pour rôle de récupérer les éléments de coopération lors de leur production par les entités de l'application, elle prendra en charge la circulation de ces informations et s'occupera également de la gestion des groupes de travail dynamiques et de l'état de l'application.

2. Vocabulaire du Domaine

Nous travaillons à la croisée de différents domaines et utilisons donc un vocabulaire varié. Aussi, afin de bien s'entendre sur les termes informatiques utilisés, il nous semble important de préciser un certain nombre de définitions liées au vocabulaire de base.

Objet

Selon [Muller 97], un objet est « *une unité atomique formée de l'union d'un état et d'un comportement* ». Il se doit de proposer l'encapsulation permettant ainsi d'avoir une forte cohésion interne avec un couplage faible avec l'extérieur.

Agent

Selon Jacques Ferber [Ferber 95], un agent est une entité physique (robot, avion, …) ou virtuelle (programme) capable d'agir dans un environnement. Il peut communiquer directement avec d'autres agents et possède ses ressources propres. Il ne dispose que d'une représentation partielle, possède ses propres compétences et offre ses services.

Règle

Une règle peut être vue comme une formule indiquant ce qui peut être fait dans un cas bien déterminé. Si l'on se réfère au domaine strictement informatique, une règle est « *un traitement de conditions communément observées dans une situation donnée ou bien un traitement sur une partie d'une action désignée pour obtenir un résultat donné* » [Front-Compte 97].

Les règles dont nous nous inspirons sont les règles ÉCA (Événement-Condition-Action). À l'occurrence de l'événement É, si la condition C est vérifiée, l'action A est exécutée.

Module

Le Petit Robert nous apprend qu'un module est une unité constructive d'un ensemble. Dans notre contexte d'utilisation, un module représente un dispositif opératoire (matériel ou logiciel) éventuellement complexe, pré-existant et non modifiable, susceptible de réaliser une tâche et de communiquer avec son environnement. Ce peut être un composant logiciel, un automate, une cellule de production, etc. Les modules sont récupérés afin d'être ré-intégrés dans un milieu coopératif.

Groupe de Travail

Un groupe de travail est une composante propre regroupant un ensemble de modules avec un objectif commun. Il est composé de modules, éventuellement distants sur un réseau, et travaillant à l'accomplissement d'une ou de plusieurs tâches bien déterminées. Nous utilisons plus généralement la notion de groupe de travail dynamique dans le sens où la composition (modules composants les groupes) de ces groupes peut varier dans le temps de manière à s'adapter à la demande du système afin de répondre au plus juste aux besoins du moment. Ces groupes de travail sont des entités virtuelles et représentent un niveau d'abstraction supplémentaire. Ils sont ajoutés à l'application avec l'objectif de réaliser une nouvelle structuration de celle-ci.

Événement

Anatole France disait « *Mais qu'est-ce qu'un événement ? Est-ce un fait quelconque ? Non pas ! me dites-vous, c'est un fait notable.* ». Ainsi, un événement peut se définir comme un fait important digne d'être remarqué. C'est également selon le Petit Robert, une conjoncture, une situation, une circonstance.

D'un point de vue purement informatique, un événement est un élément de notification suffisant en soi et possédant un caractère d'urgence. La sémantique d'un événement est exprimée par la simple occurrence de celui-ci, on dit qu'*il se suffit à lui même*.

Message

Les objets communiquent entre eux par le biais de messages. Dans la modélisation objet, le concept de message est l'unité de communication permettant de relier dynamiquement entre elles les différentes entités identifiées. Un message est une information éphémère (en opposition aux données ci-dessous) pourvu d'un contenu sémantique (contrairement aux événements ci-dessus dont la sémantique est leur propre occurrence).

Donnée

Selon le dictionnaire, une donnée est une représentation conventionnelle d'une information (fait, notion, ordre d'exécution) sous une forme (analogique ou numérique) permettant d'en faire un traitement automatique. On peut voir dans cette définition la notion de « *représentation conventionnelle* » permettant d'exprimer l'aspect structuré d'une donnée.

Dans le cadre de nos travaux, les données représentent les informations persistantes du système d'information (SI) qui existent déjà (on ne modifie pas le SI).

Notification

Selon le Petit Robert, une notification est un acte par lequel on fait connaître expressément une information (une décision) dans les formes légales à une personne concernée. L'acte de notification nous permet de signaler une information, qu'elle soit de type événement ou message.

Nous avons volontairement omis de donner les définitions de la coopération et de la ré-ingénierie, en effet, celles-ci seront explicitées à de nombreuses reprises et exprimées selon leur contexte d'utilisation.

Afin de mieux cerner nos travaux, nous allons maintenant entrer un peu plus dans le détail et expliquer les grandes lignes de notre méthode.

3. De la méthode…

La méthode que nous proposons permet de modéliser la coopération (circulation d'information et constitution des groupes de travail). Elle dissocie d'emblée les aspects coopératifs sur lesquels nous faisons porter notre contribution, des aspects opératoires réalisés par les modules considérés comme des composants autonomes distribuées et interconnectés par un réseau.

L'approche retenue est descendante. Nous partons du niveau d'abstraction le plus haut (constitution en groupes de travail) en allant vers le niveau le plus bas (modules composant les groupes de travail). Nous mettrons également en évidence le besoin d'avoir, et donc de gérer, des groupes de travail qui seront dynamiques. Leur composition est variable, les modules peuvent entrer/sortir de ces groupes. Puis nous mettrons en évidence à chaque étape les événements clés de la dynamique et les informations (persistantes ou non) qui doivent être échangées et partagées entre et dans ces groupes de travail.

Ces étapes préliminaires réalisées, nous verrons comment mettre en correspondance certains éléments de coopération. Nous verrons également comment créer ceux qui manquent et qui étaient auparavant crées par la coopération externe.

Enfin, nous gardons comme souci permanent celui de rendre la méthode opérationnelle et de permettre d'aller jusqu'à l'implémentation. Afin de faciliter cette implémentation, un langage de spécification a été défini afin de, premièrement réaliser une vérification formelle de l'analyse, ensuite de dériver les règles qui seront intégrées à la plate-forme coopérative.

4. …à l'implémentation…

Une fois la dérivation en règles réalisée, on peut mettre en place la couche coopérative de l'application qui laissera à chaque module sa totale autonomie de fonctionnement.

La solution que nous proposons consiste à mettre en place une plate-forme coopérative prenant en charge de façon automatique la distribution des éléments de coopération, la création d'informations synthétiques, ainsi que la constitution et la gestion de la dynamique des groupes de travail. Cette couche devra par conséquent être capable d'identifier et d'acheminer les éléments de coopération de même qu'elle devra déterminer les instants auxquels un module peut/doit intégrer ou quitter un groupe

de travail. Pour cela, la méthode proposée permettra d'identifier ces éléments clés et de les utiliser dans la mise en œuvre de la couche coopérative. Le but est d'obtenir une coopération plus fortement couplée qu'elle ne l'était auparavant.

Séparer ainsi la partie opératoire de la partie coopérative permet de faire évoluer ces modules plus librement, chacun indépendamment de l'autre, au cours du cycle de vie de l'application (mise à jour de certains modules, adjonction de fonctionnalités …) sans perturber le fonctionnement global de l'application. Chaque module sera principalement vu comme une entité nécessitant et produisant un certain nombre d'éléments de coopération.

Afin d'éviter tout malentendu, nous devons préciser un point important. Il ne s'agit pas pour nous de concevoir ex nihilo des applications coopératives. Nous proposons au contraire de définir des outils de modélisation et de mise en œuvre permettant de rendre coopératives des applications existantes. Aussi ne nous intéressons nous qu'aux aspects de constitution et d'évolution des groupes de travail, ainsi qu'aux aspects de communication entre les modules et les groupes, sans intervenir sur les fonctionnalités de ces modules ni sur le système d'information comme ce serait le cas en employant des techniques d'interopérabilité sémantique (ces techniques permettent la coopération entre des langages de modélisation différents et entre des bases de données hétérogènes). Nous supposons en particulier que les modules opératoires existent et que le système d'information (SI) de l'application est déjà défini. Notre contribution ne porte donc que sur l'interopérabilité technique.

L'architecture choisie s'inspire de différents travaux décrits dans le chapitre suivant avec entre autres ceux de [Lebastard 97] dans le sens où toute l'activité est gérée de manière locale sur chaque poste par un ensemble de mécanismes.

Des travaux antérieurs avaient porté sur la réalisation à l'aide de mécanismes de tâches et de rendez-vous, d'une couche active pour un SGBD relationnel (ADACTIF) [Tawbi 96]. Nous avons ensuite étendu cette couche pour y inclure des aspects de réseau puis nous avons incorporé les aspects coopératifs en permettant à la fois la gestion des différents éléments de coopération et des modules constitués en groupes de travail répartis sur un réseau. Ce travail réalisé, nous avions à disposition l'ensemble des mécanismes de base d'une architecture coopérative en réseau [Roose 98a] [Roose 98b].

5. …en passant par un prototype

Cette thèse, comme nous venons de le rappeler, fait suite à de précédentes recherches menées dans le cadre des Bases de Données Actives et au cours desquelles un prototype de Système de Gestion de

Bases de Données Actif (SGBD-A), appelé ADACTIF [Tawbi 96], a vu le jour. Ce travail lui même faisait suite à des travaux sur l'extension d'un SGBD relationnel aux Types Abstraits de Données [Aniorté 92].

5.1 Origine des outils

Pour réaliser la plate-forme coopérative, nous avons utilisé de nombreux outils différents issus des domaines de l'informatique distribuée, mais aussi, et peut être surtout, nous avons eu comme base ADACTIF. Bien que nous ne traitons pas des bases de données actives, nous avons retiré de ces travaux un certain nombre de concepts et d'idées plus qu'intéressantes pour la mise en œuvre de notre plate-forme. Aussi, allons nous faire une petite parenthèse et expliquer ce qu'est ADACTIF.

5.1.1 Activité dans les SGBD

Nous allons, dans ce qui suit, détailler les concepts liés à l'activité qui doivent être supportés par un SGBD pour qu'il soit actif. Ces concepts ont été mis en œuvre dans de nombreux SGBD-A comme Starbust [Widom 90] HiPAC [Mc Carthy 89], SAMOS [Gatziu 93], Sentinel [Chakravarthy 93], Ode [Gehani 91], Exact [Diaz 92]. Il faut insister sur le fait qu'un SGBD-A est avant tout un SGBD avec toutes les fonctionnalités classiques [Dittrich 93].

Le comportement actif dans les SGBD-A est obtenu par intégration de règles de type Événement-Condition-Action (ÉCA) [Dayal 88] inspirées des règles de production dans les systèmes experts. La logique des règles ÉCA est la suivante : à l'occurrence de l'événement É, si la condition C est satisfaite, alors on exécute l'action A. Ainsi les règles ÉCA rendent le SGBD réactif à ce qui se produit dans la BD et son environnement. L'idée de réactivité n'est pas nouvelle ; elle a été proposé dans le système R [Eswaran 76] sous la forme d'assertions et existe sous forme de triggers dans bon nombre de systèmes comme Sybase, Oracle 7, etc.

Les SGBD actifs sont capables de reconnaître des situations spécifiques et de les traiter sans demande explicite de l'utilisateur ou de l'application. Ils associent des *actions* (programmes exécutables contenant des opérations BD) à des *situations* (e.g. l'occurrence des opérations BD). L'existence de l'activité permet aussi de séparer le contrôle de l'application elle-même afin d'obtenir des applications plus modulaires, facilement modifiables et capables de répondre automatiquement à des situations exigeant des réactions rapides.

Afin de bien comprendre nos travaux, il nous semble nécessaire de présenter ADACTIF.

5.1.2 ADACTIF

ADACTIF est un SGBD Actif qui a été développé à l'IUT Informatique de Bayonne en collaboration avec l'université Paul Sabatier de Toulouse. Il vise principalement :

1. à exploiter le principe de rendez-vous pour mettre en place l'activité dans une BD,
2. à utiliser un langage procédural pour la spécification des règles ÉCA et de celles associées à des événements composés.

Parler de SGBD-Actif ne peut se faire sans parler des règles Événement-Condition-Action (ÉCA) aussi appelées règles actives. Avant d'aller plus, loin, nous allons nous attacher à bien définir ce qu'est une règle active et ce qui la déclenche : les événements.

5.1.2.1 Evénements

En informatique, un événement est une occurrence ou l'arrivée d'un programme dans un état particulier et significatif. Les événements sont généralement décomposés en deux catégories : les événements primitifs et les événements composés. Les premiers sont basés sur des opérations atomiques comme :

Les événements internes aux SGBD sont basés sur les transitions entre les différents états d'une base de données (suite à une insertion, à une suppression, …). Ils peuvent également êtres temporels (e.g. une date précise), un événement externes (e.g. venant de l'environnement), ou un événement abstrait (i.e. spécifié par l'utilisateur et signalé explicitement au système) [Front 94].

Les événements composés sont combinés récursivement à partir d'événements primitifs.

5.1.2.2 Règles active

Dans les règles ÉCA, l'événement indique le moment du déclenchement de la règle, la condition vérifie si l'état de la BD permet l'exécution de l'action qui à son tour exécute des opérations en réponse à cet événement.

Souvent appelées *triggers* dans les produits relationnels, les règles actives fournissent un traitement réactif. C'est une forme de traitement qui est motivée par l'occurrence de certains événements (stimuli), typiquement par des opérations BD. Les règles actives peuvent réaliser des requêtes sur la BD pour recueillir des informations sur les événements et les objets de la BD et décider si il est nécessaire de réagir. Elles peuvent alors exécuter les requêtes nécessaires ou, lancer des programmes externes.

Les règles actives peuvent aussi être utilisées selon [Front-Conte 94]pour :

- La communication avec l'utilisateur par notification d'événements,
- La structuration en procédures déclenchables sur un événement,
- L'initialisation de valeurs par défaut pour des attributs de certaines classes,
- La vérification et le maintien de contraintes d'intégrité,
- Le traitement d'exceptions,
- L'optimisation de la gestion physique.

5.1.2.3 Apports d'ADACTIF

Le principe des tâches parallèles ADA avec leur mécanisme de synchronisation (*rendez-vous*) est apparu séduisant et à donc été retenu pour implanter les règles.

Les règles dans ADACTIF sont considérées comme des tâches actives attendant un rendez-vous du système lors de l'occurrence de l'événement déclenchant. Chaque tâche est en charge de vérifier si la condition est valide, et dans l'affirmative, d'exécuter l'action associée. Cette manière d'opérer apporte d'une part une réponse aux besoins d'autonomie et de parallélisme en utilisant les tâches qui réalisent la partie C-A, et d'autre part, une réponse en terme de synchronisation des règles avec leurs événements via le concept de rendez-vous.

Les événements (ainsi que les événements composés) jouent un rôle important dans les SGBD actifs. Un événement composé est détecté par une règle détectrice en attente de rendez-vous avec les événements composants. Lorsque ces derniers sont au rendez-vous, et après la réalisation de la composition décrite par un algorithme, cette règle signale l'événement composé au système. Cette approche permettra à l'utilisateur de spécifier ses propres opérateurs de composition conformément à la sémantique de l'application (et contrairement à de nombreux autres SGBD-Actifs qui ne peuvent réaliser la composition qu'à partir d'opérateurs prédéfinis au nombre desquels on trouve les classiques opérateurs booléens).

En ce qui concerne les événements composés, au lieu d'un ensemble fixe d'opérateurs de composition, ils sont obtenus aussi par des règles (mécanisme de détection basé sur le principe de tâches et de rendez-vous). Une règle destinée à la détection d'un événement composé aura plusieurs événements déclenchants : les événements composants. Elle encapsule un corps réalisant la composition de manière procédurale puis la production de l'événement composé. Concrètement, ce type d'événement est spécifié d'abord en tant qu'événement abstrait. Ensuite, l'utilisateur définit lui-même son opérateur de composition dans une règle. La procédure de composition d'événements exécutée par cette dernière signalera l'événement afin que celui-ci soit pris en compte.

5.1.2.4 Intérêts d'ADACTIF

En ce qui concerne nos travaux, ADACTIF constitue une base intéressante. Il contient en particulier quelques idées fortes qui nous ont directement inspiré dans nos travaux.

- Le mécanisme de tâches parallèles inspiré du langage ADA.
- Les règles détectrices qui permettent de passer outre les restrictions des conditions booléennes communément rencontrées et qui permettent d'obtenir des événements composés représentant des situations complexes,
- Une modèle d'exécution particulièrement clair et modulaire sur lequel nous avons basé notre plate-forme pour la coopération.

5.2 Réalisation et test de la plate-forme coopérative

L'adaptation des concepts et outils d'ADACTIF semblait constituer une bonne base à la création d'une plate-forme coopérative. Il convenait à la fois de vérifier que cette plate-forme permettait e faire fonctionner une applications revisitée par notre méthode.

Aussi, nous nous sommes intéressés d'une part à la réalisation de cette plate-forme distribuée et d'autre part à l'étude d'un cas concret permettant de tester méthode et plate-forme sur un prototype.

5.2.1 La plate-forme coopérative

C'est donc par extension du modèle d'exécution d'ADACTIF que nos avons conçu notre plate-forme coopérative. Les axes essentiels de l'évolution devaient être :

- La prise en compte de la distribution sur le réseau,
- L'intégration des modules de l'applications à la plate-forme coopérative,
- La gestion d'éléments de coopération autres que les événements,
- La prise en compte de l'existence des groupes de travail.

Ainsi nous avons dû mettre en place :

- Un Gestionnaire de Communication capable de faire circuler les éléments de coopération sur le réseau.
- Des Gestionnaires de Modules dont le rôle est d'assurer l'interface entre les modules et la plate-forme en particulier en capturant les éléments de coopération issus des modules pour les ré-injecter dans la plate-forme et, réciproquement, en récupérant les éléments de coopération circulant sur la plate-forme pour les transmettre aux modules.

- Une extension du Gestionnaire d'Événements du modèle ADACTIF adapté à la gestion de tous les éléments de coopération, et pas seulement les événements : le Gestionnaire d'Éléments de Coopération.
- Une base de données de l'état de l'application permettant à tout instant de savoir comment sont constitués les divers groupes de travail. Des règles ÉCA particulières se chargeront de mettre à jour cette BD en fonction de l'évolution de ces groupes.

5.2.2 Le prototype

Nous avons toujours soutenu que nous voulions une méthode opérationnelle permettant d'aller jusqu'à l'implémentation. Le prototype a donc une valeur importante à nos yeux puisqu'il nous permettra de valider nos propositions. Dans un soucis de clarté, nous avons disposer d'un exemple permettant de vérifier visuellement nos dires. Le prototype devra permettre de suivre pas à pas chacun des éléments de coopération identifiés, d'où ils viennent, où ils vont, comment ils ont été créés le cas échéant, etc. Il devra également montrer l'organisation en groupes de travail et les éventuelles évolutions de leur composition.

Afin d'exploiter la plate-forme coopérative, nous avons simulé une application réelle tirée d'un cas concret que nous présenterons ultérieurement (une entreprise de messagerie composée de plusieurs agences) à laquelle nous avons entièrement appliqué la méthode. Nous n'avons pas désiré développer l'application dans son intégralité mais avons choisi les parties permettant d'illustrer au mieux nos propositions.

CHAPITRE 1

ÉTAT DE L'ART

1. Méthodes

Il nous est naturellement apparu nécessaire d'étudier un certain nombre de méthodes. Ainsi nous nous sommes intéressés à deux méthodes très orientées BD actives nous permettant d'avoir une première idée de la prise en compte des événement.

Bien que nos préoccupations ne soient pas orientées vers les agents, nous étudierons aussi une méthode centrée agents afin d'avoir un panorama le plus complet possible sur les méthodes permettant de réaliser une gestion de la coopération.

Enfin, nous étudierons une méthode entièrement centrée « événement » dans laquelle l'événement est utilisé pour exprimer le comportement de l'application.

1.1 Méthodes pour BD Actives

Les bases de données sont sorties de leur domaine d'application initial qu'étaient les applications de gestion et de comptabilité. Les SGBD relationnels sont devenus inadaptés aux nouvelles orientations comme le multimédia, la conception assistée par ordinateur, l'aide à la décision, le génie logiciel. Afin de répondre aux nouveaux besoins, de nouvelles pistes ont été explorées, parmi celles-ci : les bases de données orientées objet, les bases de données déductives et, plus proche de nos préoccupations, les bases de données actives.

1.1.1 Les événements dans les méthodes pour SGBD-A

Pour modéliser le comportement d'un système, il faut spécifier les événements pertinents ainsi que leur sémantique. Mais il faut aussi exprimer leurs conditions de synchronisation lorsqu'ils participent au déclenchement d'autres événements. L'utilisation d'événements dans les méthodes n'est pas nouveau, puisque Colette Roland [Rolland 88] l'introduit dans la conception intégrée de la structure et du comportement d'un SI. Un événement y est défini comme *quelque chose qui survient dans l'organisation, il est vu comme une cause ou une condition de l'exécution d'opérations ou d'activités réalisées par des entités de l'application ou par des acteurs extérieurs et qui modifient l'état de l'organisation*.

Certaines méthodes à objets utilisent aussi le terme d'événement avec souvent une légère nuance due à leur champ d'application. Ainsi, par exemple, dans UML [Muller 97], c'est une occurrence qui engendre un changement d'état. Dans OMT [Rumbaugh 91], un événement correspond à la transmission d'information d'un objet vers un autre de manière asynchrone.

Collette Roland quand à elle définit trois types d'événements :

- *Temporel*. Les événements temporels sont déclenchés à intervalle régulier ou bien à certaines dates.
- *Aléatoire*. Ces événement ont un déclenchement imprévisible, comme une panne.
- *État*. Ils correspondent à un changement d'état des objets par exemple lors de la réalisation d'une commande.

Ces trois types d'événements se retrouvent dans de nombreuses méthodes incluant ce concept.

1.1.2 Règles actives dans les méthodes pour SGBD-A

Agnès Front dans [Front-Conte 94] réalise un classement des différents types de règles actives dont voici la synthèse :

Rôle		Structure
Règles Actives	Interaction Utilisateur – Système	Action destinée à l'utilisateur.
	Modèle de données	Action destinée à traiter l'intégrité, les exceptions, les valeurs par défaut.
	Communication Inter-Applications	Action destinée à échanger des données.
	Gestion BD	Action destinée à optimiser la BD

Tableau 1 : Classification des différents types de règles

Cette classification des règles actives permet de montrer un ensemble de cas différents où ces règles peuvent être utilisées mettant ainsi en correspondance l'application avec les moyens disponibles. Par exemple, le concepteur d'une application voulant réaliser du contrôle d'intégrité saura qu'il peut résoudre son problème à l'aide de règles actives. Sachant que, la plupart du temps, ce même

concepteur a à sa dispositions plusieurs solutions pour résoudre son problème, le choix des règles actives sera motivé par l'envie de séparer les données de la base de connaissance. Une fois ce choix réalisé, il passera par la conception afin d'en avoir une traduction pour son implémentation.

Maintenant que nous avons expliqué le sens des règles et celui des événement, nous allons nous attacher à étudier quelques méthodes qui ont marqué ce domaine ou qui, par leur approche, apportent une idée intéressante pour notre sujet.

1.1.3 (ER)^2

Le modèle (ER)^2 [Tanaka 92] porte sur l'étude conceptuelle des problèmes liés à l'incorporation du comportement des bases de données actives sous forme d'événements et de règles dans le modèle Entité-Relation (E/R), modèle bien connu et particulièrement par la communauté française grâce à son lien avec la méthode Merise. Le modèle E/R retenu est en fait une première variante (le modèle E/R étendu) incluant les notions de généralisation/spécialisation, d'agrégation, … Le résultat est le modèle (ER)^2 (*Entity-Relationship-Event-Règle*).

Il fournit un ensemble d'outils pour assister le concepteur de la base de donnée dans les tâches de spécification, d'analyse et d'incorporation du comportement actif. La validation de l'étude du comportement actif est réalisée à l'aide d'une variante de haut niveau des réseaux de Pétri (*e/r-net* pour *event/rule net*).

1.1.3.1 Le diagramme (ER)^2

Ce diagramme représente le comportement de la base de donnée active sous la forme d'événements, de règles et de leurs interactions avec les objets. Pour éviter l'encombrement dans la représentation schématique, les spécifications des événements, des conditions et des actions sont mises à part selon une description textuelle. La même technique de mise en forme choisie pour les schémas ER est ici utilisée, gardant ainsi une certaine facilité de lecture sans pour autant perdre d'information.

Le modèle (ER)^2 peut représenter une diversité de contraintes, de comportement dans des situations ou actions ou de prescriptions restrictives. Il peut aussi renforcer une contrainte d'intégrité ou représenter le signalement d'événements d'alerte.

Ce schéma peut potentiellement représenter n'importe quel comportement d'application pouvant être géré par un SGBD-A. De plus, cette représentation peut être facilement adaptée pour une extension de l'abstraction du modèle ER comme la généralisation/spécialisation ou l'agrégation.

1.1.3.2 Exemple

EMPLOYÉ (<u>insee</u>, nom, travail, adresse, anniversaire, fonction, statut, salaire)

PROJET (nom, budget)

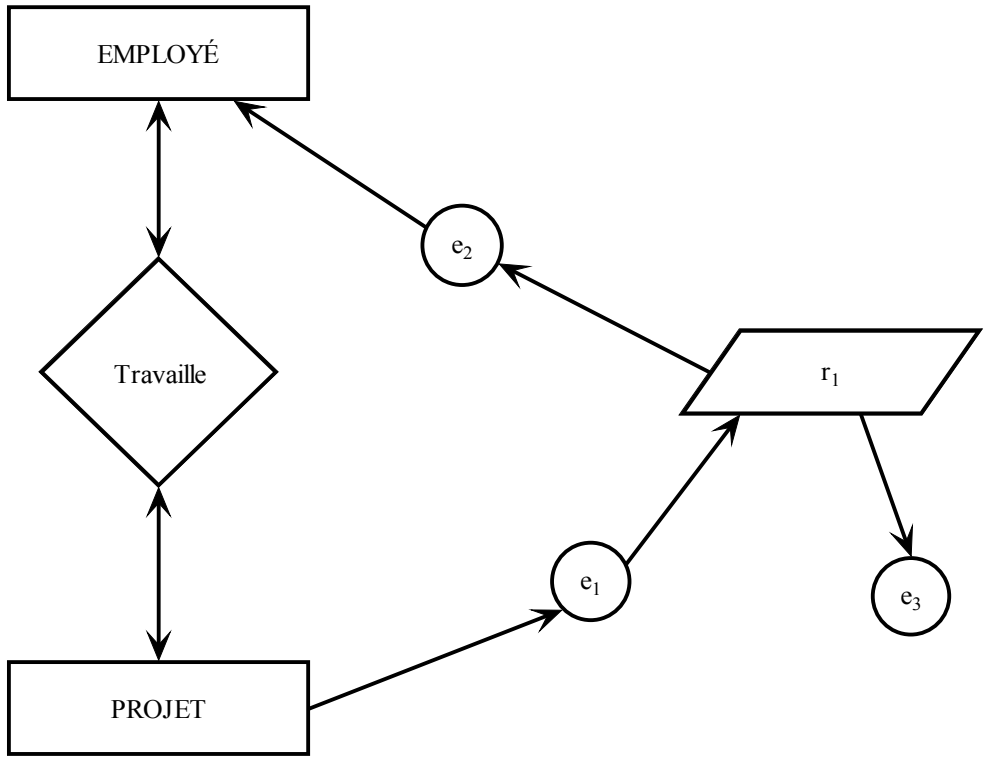


Figure 3 : Diagramme partiel de la BD d'une société

Ce schéma est une partie d'un schéma $(ER)^2$ représentant le comportement de l'application en terme d'événements et de règles attachés aux données.

A chaque événement est associé une description, sachant qu'un même événement peut avoir plusieurs descriptions différentes :

WHEN e_1 PROJET MODIFIÉ
FIRE r_1 (« Politique de réduction du budget »)
IF NEW budget < OLD budget
THEN DELETE_ENTITY EMPLOYÉ (insee = OLD EMPLOYÉ_insee,
statut = « temporaire »)(e_2),
RAISE e_3 : révision_salaire.

Les événements levés lors d'actions sont mis en correspondance avec diverses règles afin d'en contrôler l'occurrence et/ou de réaliser une réaction adéquate.

Il peut y avoir différents types de connexions mettant en œuvre les événements et les règles :

- Connexions inter-événements : Notion de précédence se reportant à l'ordre d'occurrence des événements.
- Connexions inter-règles : Notion de priorité pour déterminer l'ordre d'exécution des règles.
- Connections entre événements et règles : Le déclenchement est réalisé entre un événement et les règles associées. Un événement est levé lorsqu'il est l'un des résultats de l'exécution d'une règle.

1.1.3.3 e/r-net

La puissance de modélisation des réseaux de Pétri de haut niveau en terme de données et d'abstraction de contrôle est utilisées dans les travaux d'Asterio Tanaka afin de modéliser le comportement de la base de données active.

Schéma $(ER)^2$	Réseau de Pétri de haut niveau
Événément →	Place
Règle →	Transition
Relation « fire » →	Arc dans l'ensemble d'entrée de la transition
Relation « raise » →	Arc dans l'ensemble de sortie de la transition
Attributs des Événements et des Règles →	Tokens (jetons)
Conditions et Actions →	Enregistrements attachés à la transition

Tableau 2 : Correspondance $(ER)^2$ - Réseaux de Pétri

Dans sa thèse, Asterio Tanaka propose un algorithme (que nous ne présenterons pas ici mais que l'on retrouvera à partir de la p. 104 de sa thèse [Tanaka 92]) permettant à partir des spécifications du schéma $(ER)^2$ une dérivation afin de le valider.

1.1.3.4 Synthèse

Bien que commençant à dater un peu, cette thèse, réalisée avec le Dr. Sharma Chakravarthy bien connu dans le domaine des bases de données actives pour ses travaux sur Sentinel [Chakravarthy 93][Chakravarthy 95], est une référence dans le domaine de la modélisation pour des bases de données actives. En effet, des nombreux prototypes de SGBD-A ont vu le jour et de plus en plus, un manque cruel en méthodologie se faisait sentir afin de les utiliser de manière optimale.

Son avantage est de se baser sur un modèle mondialement connu (le modèle E/R) étendu pour l'occasion aux concepts d'activité. Le modèle $(ER)^2$ ne nécessite pas une complète remise en question

du concepteur qui assimilera aisément les ajouts proposés. De plus, la possibilité donnée de valider le schéma $(ER)^2$ par dérivation permet une vérification qui fait souvent défaut dans les modèles.

1.1.4 Projet IDEA

Ce projet planifié sur quatre ans a démarré en 1992 afin de développer un prototype industriel de « *Deductive and Object Oriented Database* » (DOOD). Ces systèmes constituent la base idéale pour le développement de bases de données intelligentes offrant à la fois les qualités d'interaction et d'efficacité de gestion des données.

1.1.4.1 IDEA en résumé

La méthode IDEA [Ceri 97] est basée sur les concepts d'objets et de règles actives et déductives et, elle se décompose selon les trois activités classiques suivantes :

- Analyse,
- Conception,
- Implémentation.

À delà de ces trois activités, IDEA inclut une phase intermédiaire entre la Conception et l'Implémentation : le Prototypage.

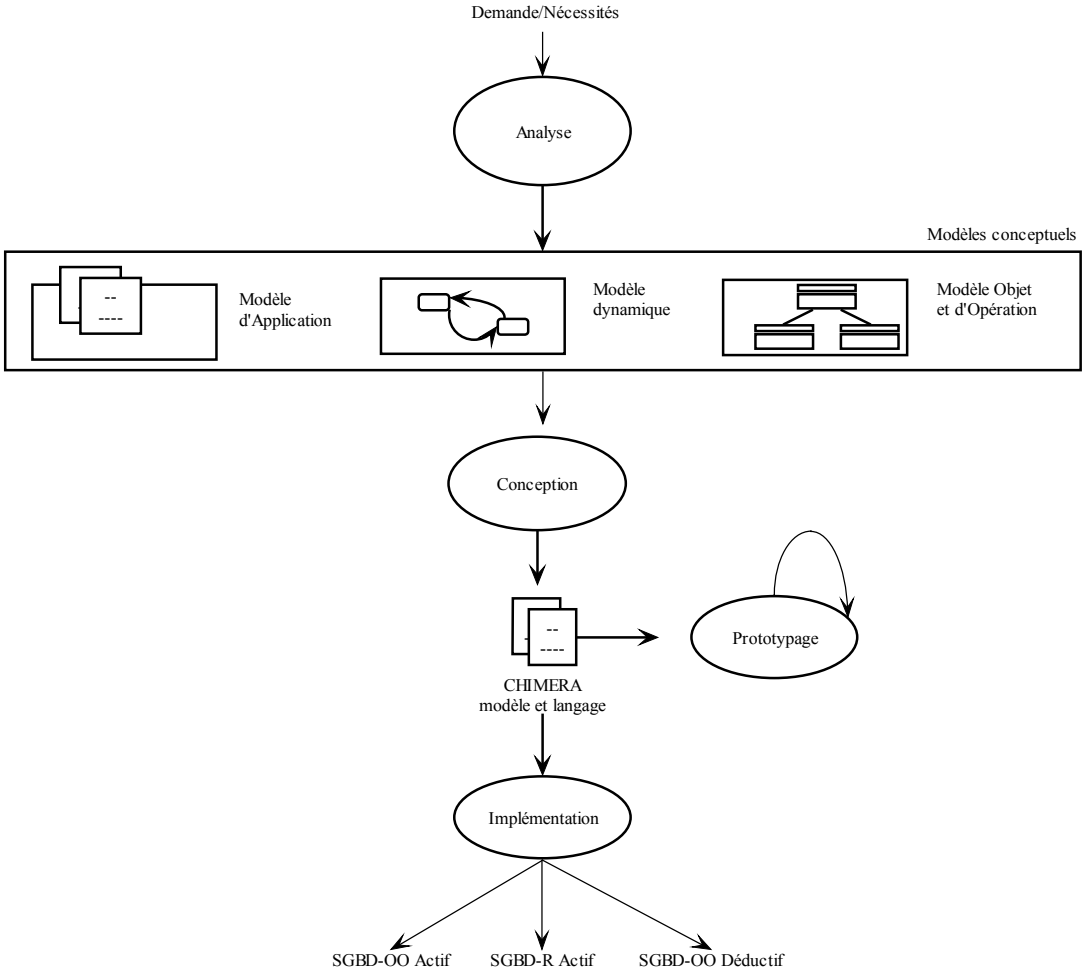


Figure 4 : La méthode IDEA

La méthode IDEA permet de se concentrer sur la sémantique des données. Cette possibilité est offerte par le modèle d'objet permettant d'exprimer d'une manière structurée certains aspects comme l'héritage, l'encapsulation, les relations sémantiques et les contraintes d'intégrité. La différence avec d'autres méthodes qui permettent la même chose est qu'avec IDEA, le résultat de l'analyse est conçu de manière structurée. IDEA n'oblige pas à de gros efforts concernant la capture des aspects fonctionnels comme le font d'autres méthodes comme [Booch 94]. Le *message passing* et les *timing diagrams* sont délibérément omis, c'est à dire, que les mécanismes d'interaction des objets principaux ne sont pas décrits.

Cette méthode se focalise premièrement sur la description des systèmes d'information et sur l'identification et le développement des systèmes logiciels partagés (*shared software system*) utilisés par toutes les applications où l'aspect fonctionnel est intégré à la description des données. Cette partie est décrite à l'aide des modèles objets, dynamiques et d'application.

1.1.4.2 L'analyse

Cette partie se décompose en deux sous parties : l'analyse grossière et l'analyse détaillée. La première étape fait un tour d'horizon des nécessités de l'utilisateur pour évaluer les limites du problème. Pour cela deux questions sont nécessaires :

- *Quelles sont les abstractions nécessaires qui constituent le « vocabulaire » du domaine du problème.*
- *Quelles sont les principales fonctions qui doivent être supportées pour couvrir les besoins.*

Dans cette étape est aussi réalisée l'identification de la cible permettant de définir la liste des concepts du domaine d'application pour fournir une première liste de classes qui seront incluses dans le futur système. Elle inclut aussi l'identification de l'application qui a pour objectif de fixer les fonctions requises par l'utilisateur pour un domaine donné. La question est : *quelles sont les fonctions développées qui incluent/dépendent des concepts partagés du domaine de l'application ?* Le résultat est une liste d'applications décrivant les fonctions principales du système à développer.

La seconde grande partie est l'analyse détaillée qui permet de définir le contenu de l'information, les propriétés statiques et dynamiques, les relations mutuelles des cibles et applications identifiées par l'analyse grossière. Les données entrées sont les besoins de l'utilisateur, et la description des destinations et des applications produites par l'analyse grossière. En sortie, nous avons un ensemble de documents constitués des modèles Objets, Application et Dynamiques, plus une version révisée de la destination et les listes d'application.

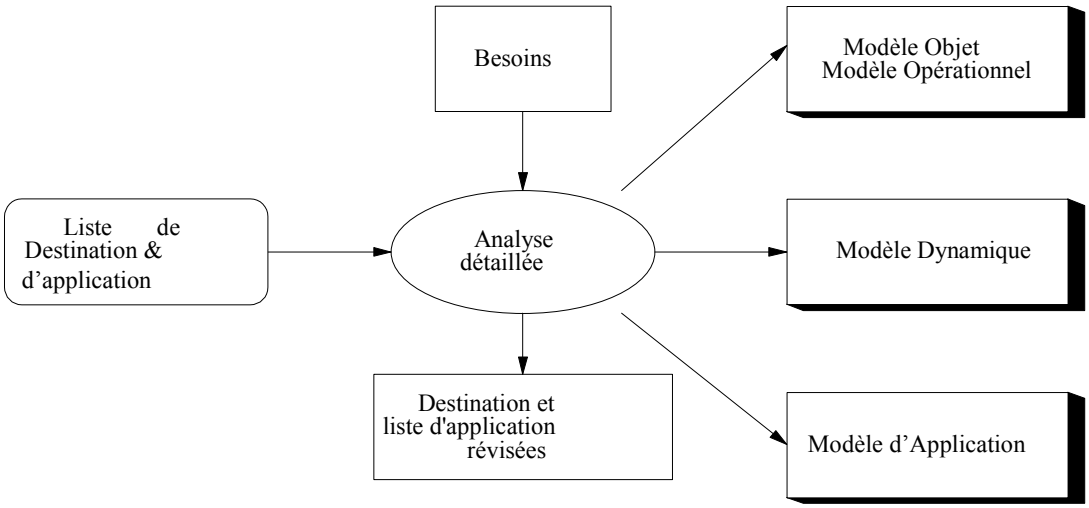


Figure 5 : Informations nécessaires/produites dans l'analyse détaillée

Le modèle objet décrit la structure interne, les propriétés, les opérations et les relations des classes. Le modèle dynamique décrit le comportement des classes en réponse aux événements, et le modèle

d'application décrit les opérations et les dépendances des données des applications. L'ensemble de ces modèles fournit les connaissances disponibles caractérisant le partage des objets persistants identifiés par l'analyse des processus.

1.1.4.3 Conception

Cette partie se compose d'un schéma de conception (*design schema*) ainsi que de l'étude des règles déductives et actives.

Elle a pour but de traduire les modèles Objets et Dynamiques en schémas conceptuels écrits à l'aide de Chimera (modèle et langage pour la conception et le prototypage) et fournissant des spécifications précises et non ambiguës du domaine d'application.

- *Le schéma de conception*. Le modèle objet étant très proche du modèle Chimera, les transformations nécessaires pour arriver au schéma de design seront simples. Elles consisteront en un passage d'une description graphique (modèle Objet) en une description textuelle (modèle Chimera). Seuls les liens de relations entre objets sont conservés. En effet, leur représentation graphique est très riche et leur traduction textuelle n'apporterait rien si ce n'est des risques de perte d'information.
- *Conception des règles déductives*. Cette partie concerne la génération des règles déductives représentant la partie déclarative de la connaissance. Cette étude est influencée par les spécifications collectées lors de la phase d'analyse. Durant cette conception, de nombreux ajouts nécessaires peuvent être identifiés. Le concepteur doit définir toutes les dérivations des données qui enrichissent la connaissance. Il doit aussi détecter toutes les informations nécessitant une vérification de consistance et qui peuvent être exprimées à l'aide de contraintes d'intégrité. Cette étape doit être exhaustive.
- *Conception des règles actives*. Cette partie concerne la définition des règles actives. Elle nécessite trois types de spécification : contraintes d'intégrité, traitements et comportement dynamique des objets. Dans cette étude, l'utilisateur peut ajouter des descriptions procédurales de l'action qui répareront la violation d'intégrité. Ces règles peuvent être facilement dérivées en combinant les spécifications déclaratives des contraintes d'intégrité produites par l'étude des règles déductives avec l'indication des actions de réparation collectées durant l'analyse.
- Enfin, les règles actives peuvent être dérivées des diagrammes d'états (*statecharts*) utilisés pour modéliser la dynamique des objets et leur évolution basée sur les événements.

1.1.4.4 Prototypage

Dans la plupart des méthodologies, la phase de prototypage dans laquelle sont testés les résultats, est réalisée en développant une première implémentation (d'une portion) du système. Elle est souvent réalisée à l'aide d'outils de prototypage rapide.

IDEA fournit des techniques et des outils pour évaluer les règles déductives et actives.

- *Prototypage des règles déductives*. Ce prototypage consiste à valider les deux propriétés de *stratification* et de *satisfaction*. La première concerne l'évaluation de l'ordre des règles déductives ; la seconde vérifie l'absence de contradiction dans les spécifications d'intégrité. Le prototypage doit valider ces deux propriétés et modifier si nécessaire les règles les violant.
- *Transformation des règles déductives et règles actives*. C'est une part importante du prototypage. Elle consiste en la transformation des spécifications déclaratives des règles déductives en spécifications procédurales codées pour les règles actives. Ces transformations ne sont pas systématiques et dépendront de la cible d'implémentation selon qu'elle supporte ou non les règles actives mais pas déductives.

1.1.4.5 Implémentation

Cette partie se charge de la traduction des spécifications conceptuelles en schémas d'objets et en règles pour des SGBD existants. Elle consistera à traduire la conception réalisée en Chiméra pour divers SGBD (commerciaux ou prototypes de recherches) comme : Oracle, Illustra, DB2, Validity et Ode. Mais la partie implémentation peut aussi réaliser des traductions dans des cadres plus généraux comme les standards SQL92 (pour le modèle relationnel) et l'ODMG Standard (pour le modèle orienté objet).

1.1.4.6 Synthèse

La méthodologie IDEA permet la conception d'applications à partir d'un ensemble d'objets et de règles en introduisant la notion de règle de gestion (*Business Règle*). Ces règles expriment les besoins de l'application en modélisant les réactions de celle-ci face à des événements.

L'avantage d'IDEA est de faire apparaître ces règles dès la phase de la conception selon la stratégie suivante :

1. Identification des tâches à accomplir et des conditions (booléennes) associées à leur exécution.
2. Pour chaque tâche, il est nécessaire de faire l'inventaire des événements la déclenchant.
3. Enfin, la génération proprement dite des règles actives qui répondront à certains événements et exécuteront la tâche associée selon l'évaluation de la condition.

Le principal reproche que l'on peut faire à IDEA est de se cantonner à un domaine de BD strict au moment où justement les mécanismes de règles actives s'intègrent dans de nombreux projets hors BDA.

Alors que les méthodes de conception ont pour but la réalisation de modèle permettant une implémentation, nous allons décrire une autre technique, diamétralement opposée, la rétro-conception dont le but est au contraire de partir d'applications existantes.

1.2 La rétro-conception

De nombreuses sociétés ont développé des systèmes d'information uniques à la fois sensibles (d'un point de vue stratégique dans l'entreprise) et non intégrés (sous formes diverses). Avec le temps, l'évolution des besoins des utilisateurs ainsi que des conditions de travail ont eu pour conséquence une évolution constante de l'intégration de l'information.

Lorsqu'au fil du temps ces modifications se multiplient, de nombreux problèmes surgissent. Ici intervient la notion de rétro-conception qui propose une méthode séduisante pour reconstituer la compréhension mais également l'organisation des données du système qui a pu être détériorée ou qui est devenue peu claire.

Le terme de rétro-conception peut être défini comme un traitement sur un produit connu fonctionnant en arrière pour définir la manière dont il a été conçu [Percival 99]. C'est un processus d'extraction des informations d'un système ou d'un logiciel. Ces informations sont ensuite conceptualisées avec l'objectif de les utiliser pour le développement de nouveaux systèmes/logiciels, pour la maintenance ou pour la gestion de projets [Glaser 96]. Notre démarche s'inscrit dans cette approche.

La popularité de la rétro-conception a crû de manière significative depuis ces 10 dernières années. La mise en œuvre d'une approche par réutilisation passe obligatoirement par une étape d'identification des ressources réutilisables. Deux tendances (éventuellement complémentaires) émergent. La première appelée analyse par rétro-ingénierie est ascendante, la seconde par analyse du domaine est descendante.

1.2.1 Objectifs

La rétro-conception est une méthode très utile pour réexaminer les systèmes existants. Elle est généralement appliquée pour améliorer les logiciels et systèmes et a pour but de déterminer ce que contient un programme. C'est souvent le dernier moyen d'obtenir des informations qui ne sont pas disponibles ailleurs.

Une autre utilité majeure de la rétro-conception est de créer des produits qui sont compatibles et qui pourront interagir avec le produit original. Le matériel, le système d'exploitation et les applications doivent tous être compatibles lorsqu'ils communiquent dans le sens où chaque composant doit comprendre le format des autres communications. Ceci est réalisé à l'aide d'interfaces communes. En plus des interfaces entre les ordinateurs, il existe un accroissement des communications entre les interfaces de l'ensemble des systèmes informatiques, entre des ordinateurs en réseau local, ou bien même entre ordinateurs distants de leur plate-forme de stockage des données.

Les fabriquants de matériel et de logiciel ne souhaitent pas fournir les spécifications techniques nécessaires à la compatibilité. Ils fournissent juste parfois des informations basiques évitant d'entrer dans le détail et permettant au fabriquant de conserver ses avances techniques.

Dans ces circonstances, la rétro-conception est l'unique moyen de découvrir l'interface ainsi que d'autres spécifications techniques nécessaires pour assurer la compatibilité entre les interfaces (une interface peut être définie comme une description précise de comment les programmes reçoivent/envoient et stockent les informations [Ignatin 92]) existantes et celles qui seront produites dans le futur.

Bien que nous n'en présenterons pas les détails ici, il est à noter qu'un certain nombre de contraintes légales, qui de plus sont différentes selon les pays, s'opposent à la rétro-conception. Le lecteur se référera à [Percival 99] qui propose une étude complète des limites légales de la rétro-conception en expliquant les variantes selon les pays et particulièrement entre les États-Unis, le Canada et l'Europe.

La ré-ingénierie est l'examen d'un système pour le reconstituer d'une nouvelle manière. Elle nécessite généralement une phase de rétro-conception. Mais contrairement à la rétro-conception, la ré-ingénierie peut changer la sémantique du système [Favre 95].

Pour résumer, la rétro-conception est l'analyse du système en ayant deux objectifs :

- l'identification des composants du système et leurs relations entre eux.
- la création d'une représentation d'un système dans une autre forme ou à un niveau d'abstraction plus élevé.

Il est à noter que [Blaha 97] réalise une distinction entre la rétro-conception des données et celle des bases de données. En effet, cette dernière a par définition une homogénéité plus importante.

1.2.2 Approche par rétro-ingénierie

La première, appelée modèle de réutilisation, analyse des produits logiciels et détermine des éléments réutilisables selon les propriétés du produit. Ces propriétés représentent le modèle de réutilisation. Ce

modèle composé d'un ensemble d'attributs permet de mesurer la réutilisabilité d'un élément. Il fournit des valeurs ou des plages de valeurs permettant de qualifier le composant candidat à la réutilisation.

La seconde approche est appelée analyse de similarités. Elle est basée sur la comparaison d'éléments provenant de produits différents candidats à la réutilisation. Cette approche est notamment utilisée dans le cadre du projet Esprit F3 [Cauvet 99] pour identifier les entités réutilisables dans la construction de schémas conceptuels Entité/Association.

Afin, la dernière approche est appelée recherche d'analogie. Les deux premières approches sont souvent perçues comme voisines du fait qu'elles visent toutes les deux à comparer des éléments communs à plusieurs systèmes. La différence apparaît lorsque l'on s'intéresse à la nature des éléments comparés. Dans l'analyse par analogie, l'idée est de rechercher les structures similaires si les instances concernées sont issues d'une même structure générique.

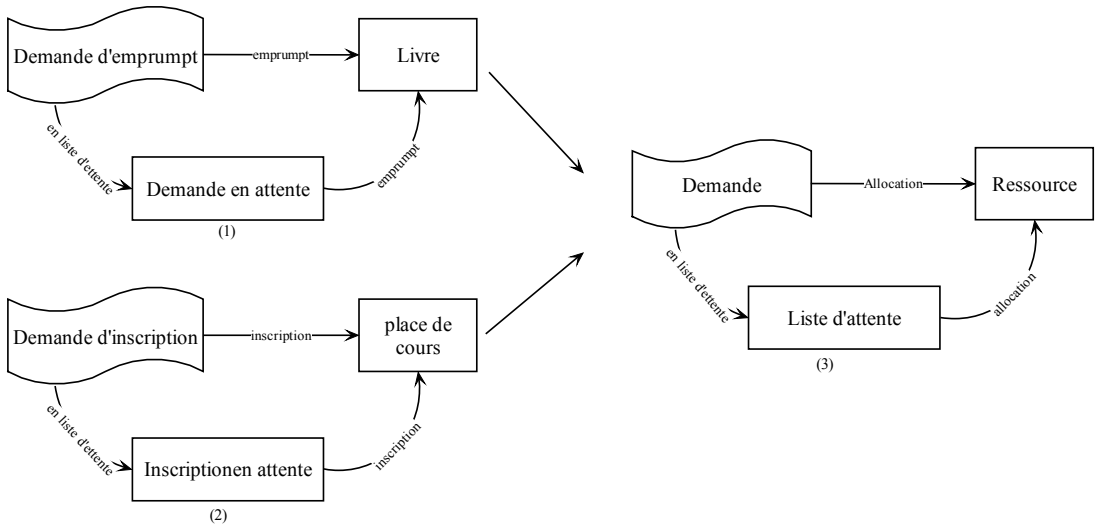


Figure 6 : Structures en correspondance analogique [Maiden 91]

Les deux systèmes représentés sont ceux d'une réservation de places à la bibliothèque (1) et au théâtre (2). L'analogie se situe ici au niveau de l'allocation des demandes en fonction des disponibilités. [Maiden 91] a d'ailleurs utilisé la correspondance analogique (3) pour développer une bibliothèque de 35 structures génériques (ou abstractions du domaine) dans le domaine de la ré-ingénierie des SI.

1.2.3 Un exemple de méthode

Nous n'allons pas davantage entrer dans le détail de la rétro-conception mais allons présenter un modèle d'analyse pour la rétro-conception de données [Aiken 98]. Bien que la rétro-conception ne se

limite pas à celle des données, nous verrons dans la méthode proposée que plusieurs points de concordance existent entre cette approche et la notre.

Le modèle proposé possède une décomposition en 13 étapes :

- *Activité 1 :Identification du système cible.*

Phase d'indentification nécessaire lorsque la compréhension de l'organisation des données du système s'est dégradée ou est devenue confuse.

- *Activité 2 : Coordination préliminaire.*

Depuis que certains systèmes sont partagés entre des composants organisationnels avec des besoins différents, il est possible d'avoir des difficultés de coordination.

Cette étape est nécessaire afin de mettre en évidence les conditions préliminaires lorsque le système sert plusieurs clients.

- *Activité 3 : Vérification des identifications et des accès.*

Elle est relativement facile à réaliser lorsque la documentation du système produit et maintenu est synchronisée en utilisant des technologie de type CASE (*Computer Aided Software Engineering*). C'est le cas le plus rare.

Lorsque la documentation existe mais qu'elle est incomplète (pas ou peu à jour), des modifications peuvent introduire des imperfections dans les données. C'est le cas le plus courant.

La pire des cas se produit lorsqu'aucune documentation n'a été réalisée, ou lorsque la documentation n'est pas disponible ou n'est plus utilisable.

- *Activité 4 : Création de l'équipe d'analyse.*

Il est nécessaire de définir le niveau d'intervention des participants dans l'analyse. Le début de leur travail sera l'analyse préliminaire des points clés (les points de survie) du système. Les membres de l'équipe d'analyse travaillent en coopération ce qui impose une compréhension globale des buts de l'analyse d'un point de vue stratégie d'intégration dans l'entreprise.

- *Activité 5 : Étude préliminaire du système*

Le but de cette étape est de déterminer la durée et le nombre de ressources nécessaires pour réaliser la rétro-conception sur les composants du système concernés. Dans cette étude, deux techniques structurées vont être appliquées : une décomposition fonctionnelle ainsi qu'une décomposition du modèle initial de données.

- *Activité 6 : Planning d'analyse*

Ce planning permet de déterminer quels sont les spécialistes disponibles, le nombre de membres dans l'équipe d'analyse, ainsi que le nombre de semaines de travail pour les membres de cette équipe.

L'équipe d'analyse va décomposer le travail en trois composantes orientées vers l'organisation spécifique des données :

- les conditions relatives ainsi le degré de confiance,
- la manipulation des données, le système d'exploitation ainsi que le facteur langage de programmation,
- la combinaison de la participation des spécialistes avec une possible automatisation.

- *Activité 7 : La démarrage de l'analyse*

Cette étape marque la transition vers l'implémentation et le début de l'analyse du système cible. A ce stade, il est nécessaire d'avoir réalisé un certain nombre d'étapes importantes comme :

- L'identification et l'implémentation de solutions nécessaires à la mise en œuvre de la coordination,
- La formation des collègues et des membres de l'équipe du projet,
- La confirmation de l'engagement des participants,
- La consensus avec les participants dans la nature de leur investissement.

- *Activité 8 : L'analyse du système cible*

Elle se répète sous forme d'un cycle jusqu'à l'obtention des résultats désirés ou dans certains cas, jusqu'à la détection de la non-faisabilité de la chose. Les étapes sont les suivantes :

Mise en évidence des liens de connexions dans le système d'information,

Mise en évidence des informations partagées,

Mise en évidence des informations structurant le système, c'est à dire le nombre et le type des relations entre les éléments à coordonner.

- *Activité 9 : Propriété des données*

Cette partie permet de s'assurer de la justesse des informations délivrées pour l'intégration dans les différentes activités de l'entreprise.

- *Activité 10 : Intégration des données*

Cela permet par exemple de retrouver les informations redondantes, synonymes et homonymes. Cette partie a pour but de résoudre les problèmes de confusion entre informations.

- *Activité 11 : Transfert des données*

Il est possible ici de permettre des évolutions comme le changement des moyens de communication, de la localisation et du format des données avec l'objectif de mieux s'intégrer aux autres activités de l'entreprise.

- *Activité 12 : Estimation de l'analyse*

Une fois l'analyse terminée, un bilan est réalisé. Il permet d'obtenir entre autres:

- Le nombre d'entités analysées,
- Le nombre de données dupliquées éliminées,
- Le nombre de données partagées identifiées.

- *Activité 13 : Affinage du modèle et de l'implémentation*

Un des points les plus important de l'analyse est de collecter et de sauvegarder les mesures d'implémentation, les procédures d'affinage, les outils et les modèles de données utilisés et enfin les concepts utilisés. Ces informations permettront des améliorations ultérieures tant en terme de modèle que de l'implémentation.

1.2.3.1 Conclusion

Nous retrouvons dans notre méthode les deux principaux objectifs de la rétro-conception. Ainsi, l'étape d'identification des composants du système est à rapprocher de l'identification des éléments de coopération qui nous réaliserons. De même, la création d'une représentation du système sera également réalisée à la fois de manière graphique par des diagrammes, mais également de manière plus formelle à l'aide d'un langage de spécification.

Il existe de nombreuses approches différentes permettant d'aborder la rétroconception, avec chacune leurs particularités afin de se rapprocher au mieux du domaine concerné. Néanmoins, il existe actuellement une convergence vers les approches à bases de patrons comme [Campo 97] [Kazman 98] [Rieu 99] s'appuyant notamment sur des méthodes d'ingénierie des systèmes informatiques comme (OMT, UML, ...) ainsi que sur celles des systèmes d'information des gestion (merise, merise objet,...).

Bien que la ré-ingénierie et la rétro-conception d'applications aient des objectifs différents, ces deux approches sont complémentaires et possèdent plusieurs points de concordance.

Alors que l'objectif de la ré-ingénierie est l'amélioration d'applications existantes, celui de la rétro-conception est de rebâtir une analyse. Ceci a pour conséquence certaines étapes communes comme le besoin de réaliser un inventaire des entités fonctionnelles présentes, et d'obtenir un inventaire des informations circulantes avec les points de connexions entre entités (activités 1 et 8). Il est ensuite nécessaire de réaliser une vérification quant à l'intégration des données (activité 10).

On peut établir un lien entre la ré-ingénierie et la rétro-conception, on pourrait dire que cette dernière est une étape préliminaire permettant ensuite de démarrer la ré-ingénierie avec toutes les informations nécessaires.

1.3 Approche par agents coopératifs

Il paraît difficile d'étudier des méthodes fortement orientées BDA sans parler des Systèmes à base d'Agents (SA). De nombreux systèmes agents sont réalisés à partir de BDA ou bien en sont fortement inspirés [Geppert 95] [Berndtsson 98]. En effet, il existe de nombreuses similitudes entre ces deux domaines, comme il apparaît dans [Bailey 94].

Malgré de nombreuses tentatives, il n'existe pas de définition largement approuvée disant de quoi est formé un agent.

Dans [Wooldridge 95] sont identifiées 4 caractéristiques clés d'un système agent logiciel/matériel :

- **Autonomie**. Les agents opèrent sans intervention directe d'humains ou autre, et ont une sorte de contrôle de leurs actions et de leur état interne.
- **Capacité sociales**. Les agents interagissent avec d'autres agents (éventuellement humains) via un langage de communication.
- **Réactivité**. Les agents perçoivent leur environnement (qui peut être le monde physique, un utilisateur par le biais d'une interface graphique, une collection d'autres agents, l'Internet, ou toute combinaison de tout ceci) et répondent au moment opportun aux changements qui se produisent.
- **Pro-activisme**. Le agents ne réagissent pas uniquement à leur environnement, ils sont aussi capables d'adopter un comportement selon le but à atteindre en prenant des initiatives.

Dans cette vue, un agent est quelque chose comme un processus système qui se suffit à lui même, s'exécute de manière concurrente, réalise son propre contrôle en encapsulant son état et communique avec les autres agents via une sorte de *message passing*.

On trouvera une description complète des agents et des systèmes multi-agents dans [Ferber 95].

Ainsi, donc, on peut faire des rapprochements entres les SA et les BDA. Voici un résumé nous montrant quelles sont ces similitudes :

SGBD-A		Systèmes agents	
	Événements	Événements	
Règles	Prédicat d'événement	Condition d'invocation	**Plans**
	Condition	Contexte de condition	
	Action	Plan	

	Transactions des utilisateurs	Buts intrinsèques	
	Base de faits	Croyances	
	Procédures de mise à jour	Plans d'invocation de buts	
	Contraintes d'intégrité et déclencheurs	Événements ou plans d'invocation de faits	

Tableau 3 : Concepts similaires entre SGBD-A et Systèmes Agents [Bailey 95]

Chaque composant (élément contenu dans la BD et représentant des éléments du monde qu'elle modélise) dans une BDA est analogue aux croyances (informations, motivation et état d'un agent [Bailey 95]) dans un SA. Le contenu de la BD représente les faits du monde dans lequel le système est modélisé, cela correspond aux croyances des agents.

Typiquement, les événements dans une BDA sont restreints aux opérations BD, mais ce n'est pas toujours le cas, et les BDA fournissent un langage riche pour combiner les événements (événements composés). De ce coté là, les SA tendent à se restreindre aux événements atomiques. Néanmoins, ils ont la possibilité de générer cours de leur exécution une variété d'événements de différents niveaux au. Enfin, le déclenchement d'un événement dans une BDA provoque l'activation d'un déclencheur qui évalue les conditions et éventuellement déclenche les actions. Les événements dans les SA impliquent l'invocation de plans exécutés une fois replacés selon les intentions.

Il existe certes d'autres points communs décrits dans [Bailey 95] que nous ne développerons pas ici, mais aussi des différences importantes.

[Gervais 98] soulève le problème de la difficulté de développer des applications coopératives. L'argument principal est la complexité des environnements dans lesquelles elles sont exécutées (systèmes interconnectés, hétérogénéité des moyens de traitement de l'information, …).

Il nous semble intéressant de proposer une sorte de bref état de l'art des méthodes orientées agents et en particulier une toute nouvelle méthode proposée par Jennings, Kinny et Wooldridge, biens connus dans le domaine et qui ont uni leurs travaux de recherche afin de proposer Gaia.

Nous allons tout d'abord nous intéresser à d'autres approches issues du domaine de l'ingénierie des connaissances : CoMoMAS et Mas-CommonKADS.

Enfin, nous examinerons les langages ALBERT et i^*, qui, bien que n'étant pas des méthodes, possèdent une approche très intéressante pour nous en raison des liens réalisés avec les règles ÉCA.

1.3.1 CoMoMAS

CoMoMAS [Glaser 96] est une extension de la méthode CommonKADS [Schreiber 94] permettant la modélisation de systèmes multi-agents.

Tout d'abord, il faut savoir que CommonKADS est le standard européen dans le domaine des méthodes pour l'ingénierie des connaissances. Elle a été graduellement développée et validée dans le cadre du programme ESPRIT IT.

Nous n'allons pas décrire avec précision la méthode CommonKADS, le lecteur pourra se référer à [Schreiber 99] pour plus de renseignements sur l'ensemble de la méthode.

Cette méthode a été de nombreuses fois améliorée et étendue. Nous allons présenter deux évolutions orientées « agent » qui nous ont semblées intéressantes dans le cadre de nos travaux en commençant pas CoMoMas.

CoMoMas propose les modèles suivants :

- *Le modèle agents*. C'est le modèle de base définissant l'architecture ainsi que les connaissances de l'agent. Ces dernières peuvent être à la fois de type social, coopératif, de contrôle mais également cognitif ou réactif.
- *Le modèle d'expertise*. Il décrit les compétences (réactives ou cognitives) de l'agent en mettant l'accent sur les différences entre :
- La connaissance réactive décrivant les procédures nécessaires pour répondre à un stimulus.
- La connaissance des tâches décrivant leur décomposition dans un modèle des tâches
- La connaissance de la résolution du problème décrivant les méthodes de résolution et les stratégies de sélection.
- *Le modèle des tâches*. Il décrit la décomposition en tâches en indiquant son moyen de résolution : agent ou acteur humain.
- *Le modèle de coopération*. Il utilise les méthodes de résolution de conflits et les connaissances en terme de coopération afin de décrire la coopération entre agents.
- *Le modèle du système*. Il définit l'aspect organisationnel de la société d'agents en incluant les aspects architecturaux de ces derniers.
- *Le modèle de conception*. Il permet de regrouper l'ensemble des modèles précédents afin de les rendre opérationnels.

1.3.2 MAS-CommonKADS

Également extension de CommonKADS, son intérêt réside dans l'incorporation de techniques issues des méthodologies orientées objet comme OMT.

Dans MAS-CommonKADS [Iglesias 98], on retrouve les différentes étapes développées précédemment ainsi que certaines améliorations. Ainsi, dans le modèle agent, certaines techniques d'identification sont proposées comme l'application d'heuristiques d'identification et la réutilisation des composants des agents déjà développés.

Ce dernier point nous intéresse particulièrement et représente une approche méthodologique de plus en plus souhaitée par les concepteurs.

Le modèle des tâches permet maintenant une description par diagrammes ainsi que par patrons textuels.

Le modèle de coopération permet de décrire les conversations entre agents (les interactions, les protocoles et les capacités requises) en utilisant des modélisations formelles comme MSC (*Message Sequence Charts* [Schreiber 94]) et SDL (*Specification and Description Language* [ITU 94]). Les améliorations apportées à ce modèle permettent d'identifier les conversations et les interactions entre agents.

Enfin, un modèle de communication a été ajouté afin de détailler les interactions entre les agents et l'acteur humain avec comme objectif supplémentaire de faciliter la réalisation des interfaces utilisateurs.

Les deux méthodes brièvement décrites précédemment sont issues de travaux dont l'origine était la gestion des connaissances (*Knowledge Management*) et qui n'étaient donc pas vouées à être utilisées pour des agents. Néanmoins certains aspects comme les modèles de tâches, de coopération et de communication ont un rôle particulièrement intéressant puisqu'ils permettent de préciser les communications entre les entités ce qui dans le cadre de nos travaux est un point essentiel.

Celle que nous allons maintenant présenter de manière plus complète est une autre approche qui, cette fois-ci, a été entièrement conçue dans l'optique d'une modélisation agent.

1.3.3 Gaia

Gaia [Wooldridge 00] est une méthode orientée agent pour l'analyse et la modélisation. Elle est à la fois générale pour pouvoir être appliquée à de nombreux systèmes multi-agents, et globale dans le sens où elle peut être utilisée au niveau macro (social) et micro (l'agent).

Gaia est fondée sur la vue des Systèmes Multi-Agents (ou SMA) comme des organisations informatiques composées de différents rôles interagissant.

Les principaux modèles utilisés dans Gaia et leurs interactions sont résumés dans le schéma suivant :

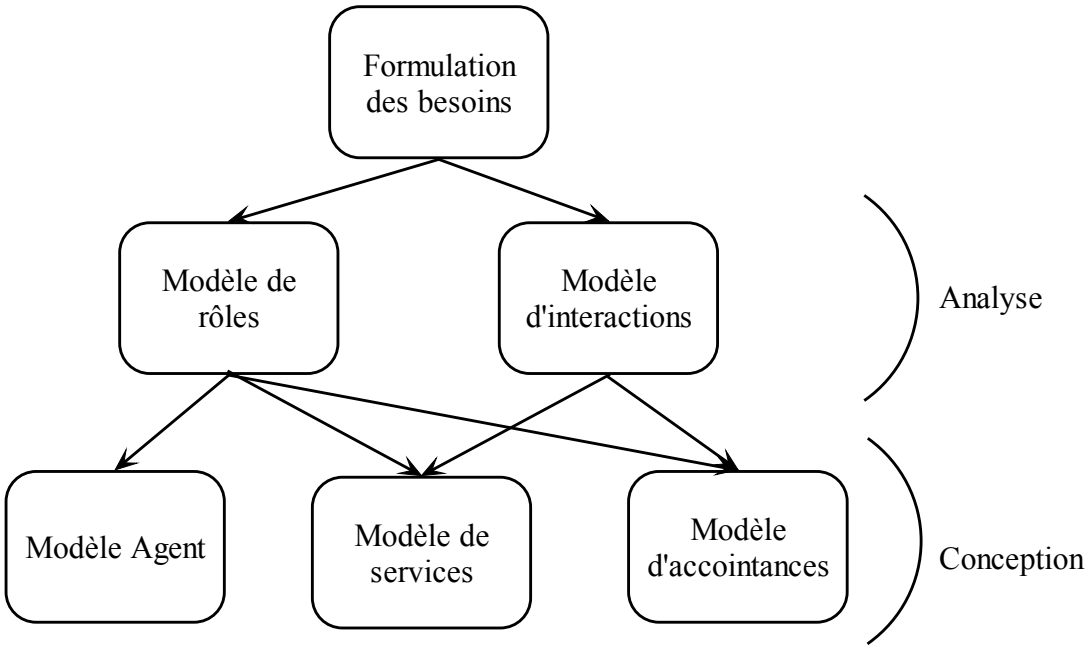


Figure 7 : Relations entre les modèles de Gaia

Les entités de Gaia sont divisées en 2 catégories : *résumé* et *concret*. Les entités de type *résumé* sont celles utilisées au cours de l'analyse afin de conceptualiser le système mais qui ne sont pas forcément réalisées par la suite. Les entités de type *concret* sont au contraire utilisées lors de l'analyse et auront leurs pendant dans l'application. Le tableau suivant présente les types *résumés* et *concrets* de Bahia :

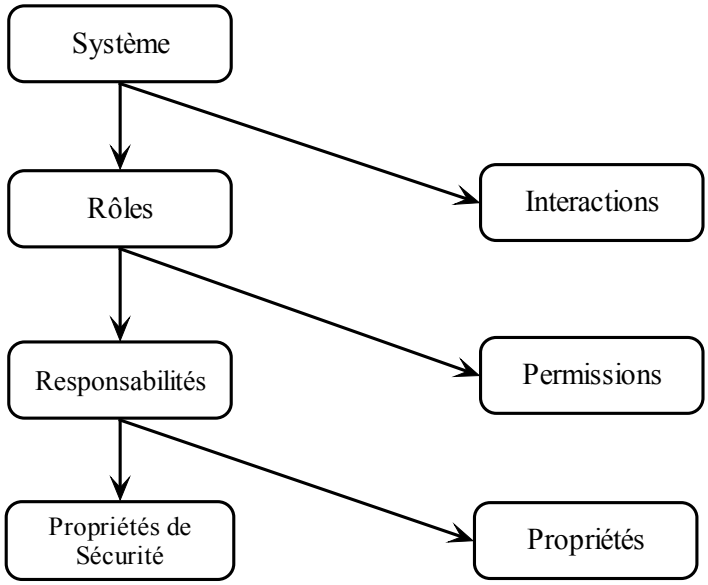
Concepts de type « résumé »	Concepts de type « concret »
Rôles	Types d'agents
Permissions	Services
Responsabilités	Accointances
Protocoles	
Activités	
Propriétés d'existence	

Propriétés de sécurité	

Figure 8 : Concepts de type « *résumé* » et « *concret* » supporté par Gaia

1.3.3.1 L'analyse

Les modèles développés dans l'analyse sont les suivants :



1.3.3.1.1 Le modèle de rôles

Il identifie les principaux rôles dans le système. De tels rôles sont caractérisés par deux types d'attributs :

- Les droits associés au rôle sont relatifs au type et à la quantité de ressources qui peuvent être exploités lors de l'utilisation de ce rôle.
- Les responsabilités du rôle sont représentées par ses fonctionnalités.

1.3.3.1.2 Le modèle d'interactions

Dans une organisation multi-agents, il existe inévitablement des interactions et des relations entre les différents rôles. Ces interactions doivent être inventoriées et représentées dans le modèle d'interactions lors de la phase d'analyse :

- *L'objectif* : brève description de la nature des interactions (demande d'informations, liste des activités, assignation de tâches).
- *L'initiateur* : le(s) rôle(s) responsable(s) de l'interaction de départ.

- *Le destinataire* : le(s) role(s) avec le(s)quel(s) le ou les initiateurs interagissent.
- *Entrées* : les informations utilisées par l'initiateur pour initier le protocole.
- *Sorties* : les informations fournies pour le destinataire.
- *Traitement* : brève description textuelle des divers traitements des initiateurs au cours des interactions.

1.3.3.2 Conception

Le but d'une étape « classique » de conception est de transformer les modèles résumés au cours de l'étape d'analyse dans un modèle suffisamment bas pour être facilement implémenté, néanmoins ce n'est pas le cas avec une conception orientée agent.

Le but de Gaia est de transformer le modèle d'analyse dans un niveau d'abstraction suffisamment bas pour que les techniques traditionnelles (incluant les techniques orientées-objet) puissent être appliquées pour implémenter des agents.

1.3.3.2.1 Le modèle agent

Le modèle agent identifie les types d'agents qui composent le système ainsi que les agents qui vont être instanciés à partir de ces types.

Les instances d'agents sont qualifiées selon FUSION [Coleman 94]. Une annotation *n* veut dire qu'il y a exactement *n* agents de ce type. Une annotation *m..n* veut dire qu'il y aura au moins m instances, mais pas plus de *n*. Il est possible également d'avoir des annotations de type * (nombre quelconque d'instances, y compris 0) et + (au moins une instance).

1.3.3.2.2 Le modèle de services

Il identifie les principaux services et les propriétés nécessaires pour réaliser le rôle de chaque agent. Selon une approche objet, un service correspond à une méthode. Dans la philosophie agent, un service est quelque chose de simple, un block d'activité cohérent dans lequel l'agent va s'engager.

Pour chaque service réalisé par un agent, il est nécessaire de documenter ses propriétés. En d'autres termes, cela veut dire qu'il faut identifier les entrées, les sorties, les pré et post-conditions de chaque service. Les entrées et les sorties peuvent être aisément dérivées du modèle de protocoles. Les pré et post-conditions représentent les contraintes sur les services. Elles sont dérivées des propriétés de sécurité du rôle (par définition, un rôle sera associé à au moins un service).

Les services réalisés par chaque agent sont dérivés de la liste de protocoles, des activités, des responsabilités et des propriétés d'existence d'un rôle.

Le modèle de service ne prescrit pas une implémentation des services qu'il documente, toute la liberté est laissée au développeur.

1.3.3.2.3 Le modèle d'accointances

Probablement le modèle le plus simple, il documente les lignes de communication existant entre les différents types agents. Ces liens ne définissent pas quels sont les messages envoyés ni quand ils le sont, ils indiquent simplement le chemin de communication.

L'objectif d'un modèle d'accointances est d'identifier les éventuels goulots d'étranglement pouvant causer des problèmes au cours de l'exécution (il est pour cela plus facile d'avoir affaire à un système faiblement couplé). Il est possible d'avoir à revenir sur la phase d'analyse pour résoudre des problèmes mis en évidence à cette étape.

Ce modèle est simplement un graphe orienté dont les nœuds correspondent aux types d'agents, et les arcs représentent les chemins de communication. Il peut être directement dérivé des modèles de rôles, d'interactions et d'agents.

1.3.3.2.4 Conclusion

La méthode résumée ici est une approche mature issue de plusieurs années de travaux que l'on peut retrouver dans [Wooldridge 97] [Kinny 97] [Wooldridge 99]. Néanmoins comme le reconnaissent les auteurs, un certain nombre de problèmes restent à résoudre comme la dynamique du système, c'est à dire, comment les entités peuvent entrer/sortir du système. Il manque également les protocoles de coopération ainsi qu'une sémantique formelle.

Les étapes actuelles semblent tout de même suffisamment abouties pour servir de base à l'évolution de Gaia dans l'espoir d'obtenir une méthode à la fois complète, robuste et simple.

*1.3.4 Le langage Albert et i**

De plus en plus, le développement de SI se réalise dans un contexte de systèmes existants et de traitements organisés établis. L'observation de l'organisation et des composants du système en tant qu'agents coopératifs offre un moyen de compréhension de leurs inter-relations et comment ces relations peuvent ou doivent être altérées lorsqu'un nouveau système est introduit.

Le langage ALBERT *(Agent-oriented Language for Building and Eliciting Real-Time requirements*) [Yu 95] est utilisé pour spécifier les nécessités en terme d'états, d'actions, d'information et de perception. Le cadre de travail *i** permet de comprendre et re-concevoir l'organisation des traitements.

Pour comprendre et caractériser les aspects coopératifs des systèmes multi-agents (SMA), il est nécessaire d'avoir des modèles exprimant et aidant le raisonnement sur « pourquoi » (*why*) les agents font ce qu'ils font.

Dans [Dubois 98] nous voyons les techniques nécessaires selon deux niveaux : un niveau de spécification décrivant ce que les agents devraient faire ou connaître (*what*), et un niveau de

« compréhension » décrivant pourquoi les agents s'associent entre eux, et pourquoi il est possible de préférer certaines configurations de relation.

La langage ALBERT a été conçu pour spécifier les nécessités (principalement fonctionnelles) des systèmes distribués temps réel. Les agents ont des états et des actions. Ils sont contraints en terme d'obligation, d'information et de perception. Les agents coopèrent en donnant aux autres des informations concernant leur état et leurs connaissances.

i* est utilisé pour obtenir une compréhension sur les relations d'organisation. Les agents ont des besoins et des capacités dépendant pour chacun des buts à atteindre, des tâches à réaliser, et des ressources à fournir.

Le modèle i* permet de supporter la génération et l'évaluation d'alternatives organisationnelles, alors que le langage ALBERT est utilisé pour produire les documents de spécification nécessaires au développement du système.

1.3.4.1 Spécifier les nécessités à l'aide du langage ALBERT

Le langage ALBERT supporte la modélisation des besoins fonctionnels en terme de collection (ou société) d'agents interagissant pour fournir les services nécessaires à l'organisation. Chaque agent est caractérisé par les actions qui changent ou maintiennent son propre état de connaissance sur le monde extérieur et/ou sur l'état des autres agents. De telles actions sont exécutées par des agents pour décharger des obligations contractuelles exprimées en terme de contraintes internes et de contraintes de coopération.

Dans ALBERT, les besoins fonctionnels sont exprimés en terme d'ensembles de déclaration formels de type logique temporelle de premier ordre.

Afin d'améliorer la lisibilité, une spécification est organisée en terme d'unités appelées agents. Les traitements logiques sont regroupés autour d'agents pour définir l'ensemble des comportements autorisés.

Les traitements logiques décrivant un agent sont classés en catégories, chacune correspondant à un modèle (pattern) de propriétés. Un tel modèle fournit un guide pour la structuration des besoins.

1.3.4.2 Déclarations

La partie déclaration consiste en une description de la structure générale du système composé en terme d'agents mais aussi en terme de structure de chaque agent individuel. Une spécification consiste en une collection d'agents.

La partie déclaration d'un agent consiste en la description de la structure de son état (i.e. la mémoire de l'agent) et d'une liste d'actions qui peuvent se produire durant sa vie et qui peuvent changer son

état. Les composants de l'état (graphiquement, des rectangles) sont typés et les actions (des ovales) peuvent avoir des arguments typés. Les types peuvent être des types de données simples ou des types de données variables (construits récursivement à partie des types prédéfinis).

1.3.4.3 Contraintes

Les contraintes sont utilisées pour réduire l'ensemble infini de comportements possibles des agents d'un système composé. La vie d'un agent est généralement une séquence infinie de changements (occurrences des actions) et de valeurs d'états. Un comportement admissible respectera :

1. les contraintes *locales* relatives au comportement interne de l'agent,
2. les contraintes de *coopération* définissant comment les agents interagissent avec les autres agents.

Les contraintes locales sont classifiées ainsi :

- *Actions et effets*. Les effets d'une action sont exprimés au travers de ses caractéristiques fonctionnelles en terme de relations mathématiques entre les différents états successifs.
- *Causalité entre actions*. Le déclenchement des actions est généralement assuré par des règles ÉCA, à n'importe quel moment, lorsqu'un événement se produit, si la condition sur l'état courant est valide, alors l'action se produit. Dans ALBERT, c'est plutôt style ce spécification opérationnelle qui est supporté tandis qu'un style plus déclaratif permet de garder les empreintes des occurrences d'actions et des causalités spécifiques entre elles.
- *Capacité.* C'est la description des règles ÉCA. En plus des circonstances selon lesquelles une action s'exécute ou non, la langage ALBERT introduit aussi des caractérisations non-déterministes où l'action est *autorisée* selon certaines circonstances (elle peut donc se produire ou non). Ce non déterminisme est très important au niveau de l'étude des nécessités induites par les aspects de modélisation du monde réel.

1.3.5 Conclusion

Les méthodes agents proposent une approche très intéressante dans le cadre de la modélisation de la coopération. Les modèles de rôles, d'interaction et de service de Gaia offrent une vue intéressante des relations entre agents et entre services proposés par les agents.

Concernant Albert et *i**, cette approche permet de répondre aux besoins en matière de modèles : modèle décrivant l'organisation du travail, et modèle supportant la ré-ingénierie [Yu 95b].

Albert insiste sur le fait que la caractérisation de la coopération nécessite l'utilisation de concepts intentionnels. Les agents et la manière dont ils s'associent entre eux doivent être caractérisés en terme

de concepts comme la connaissance, l'obligation, l'engagement et les buts. Il est à noter l'utilisation importante des règles ÉCA permettant la gestion de la communication et donc de la coopération.

Enfin, après avoir parlé de méthodes pour BDA, de méthodes agents, nous ne pouvons passer sous silence un modèle entièrement événementiel. Particulièrement abouti, il a été une source d'inspiration importante pour nos travaux, c'est le modèle IFO_2 de Maguelonne Teisseire.

1.4 Une méthode événementielle : IFO_2

Le modèle IFO_2 [Teisseire 94] est un prolongement de la méthode IFO [Abiteboul 87] (modèle de données conceptuel - représentation structurelle) dont l'objectif premier est, pour sa composante modélisation, de concilier les qualités des approches objet et des démarches de conception classiques. Cette méthode propose une perception du comportement en terme d'événements pouvant éventuellement agir sur des objets plutôt qu'une perception du comportement en terme de réaction d'objets à certains événements.

Mais IFO_2 ne s'arrête pas là et propose des mécanismes d'évolution de la représentation ainsi que des possibilités de dérivation automatique vers des modèles logiques. Enfin, et c'est l'apport principal de la thèse de Maguelonne Teisseire, ce modèle se doit d'être doté d'une composante comportementale (représentation comportementale) afin d'être un modèle conceptuel complet.

1.4.1 IFO_2 en résumé

IFO_2 est une méthode descendante : du global vers le particulier. Son approche par spécification de schémas événementiels respecte les qualités de vision globale et d'uniformité de représentation. En s'appuyant dessus, il est ainsi possible d'exprimer des conditions d'occurrences conférant ainsi au modèle un important pouvoir d'expression.

Les schémas événementiels sont faciles à appréhender car ils ne représentent pas la totalité des états remarquables d'une application. Ils permettent une spécification aussi précise mais plus simple (plus concise) que celles à base de graphes de transition. Bien que différents, ces deux modes possèdent un rôle voisin dans le sens ou chacun d'eux est chargé de représenter l'ensemble des comportements possibles du système.

Deux aspects s'imposent dans IFO_2 : l'aspect structurel dans lequel *tout est objet* et l'aspect comportement où *tout est événement* avec, comme dans les modélisations objet, le double aspect : typage (sémantique des événements) et identification (instant d'occurrence).

Les événements sont ici l'unique unité de description élémentaire de la dynamique de manière symétrique au rôle joué par les objets dans la partie statique du modèle.

La notion de comportement est au centre d'IFO_2. Elle y est perçue « *non pas en termes de réactions d'objets à certains événements, mais uniquement en termes d'événements pouvant éventuellement agir sur des objets* » [Teisseire 94]. Cette approche différente des autres a le mérite d'être tout aussi expressive tout en éliminant certains de leurs défauts.

1.4.2 Aspects du modèle comportemental d'IFO_2

Nous ne présenterons pas ici les aspects structurels puisqu'ils sont directement issus d'IFO et ne constituent pas pour nous le principal apport d'IFO_2. Nous allons directement nous intéresser aux aspects comportementaux en présentant le modèle associé.

Les aspects comportementaux dans IFO_2 sont entièrement basés sur la notion d'événements. En généralisant, il est possible de dire qu'ils sont la représentation de faits participant aux réactions du système modélisé.

De manière classique, les événements d'IFO_2 n'ont aucune durée. Mais IFO_2 va plus loin en posant une contrainte forte et simplificatrice : « *il ne peut se produire à un instant donné qu'au plus un événement* ». Cette restriction permet de s'affranchir des problèmes liés à l'occurrence simultanée d'événements, simplifiant ainsi à l'extrême la prise en compte de l'ordre (gestion de précédence) dans l'apparition des événements.

Ils peuvent être externes, temporels (événements externes) ou issus de l'application elle même (événements internes). L'ordre d'occurrence est aléatoire (dépend de l'utilisation de l'application).

Afin de simplifier les représentations, la notion de fragment (hérité d'IFO) événementiel a été introduite. Elle est articulée autour d'un type principal appelé « cœur » de manière à permettre la modularité et le ré-utilisabilité. Son rôle est de décrire un sous-ensemble du comportement modélisé pouvant dès lors être utilisé comme un tout.

1.4.3 Exemple

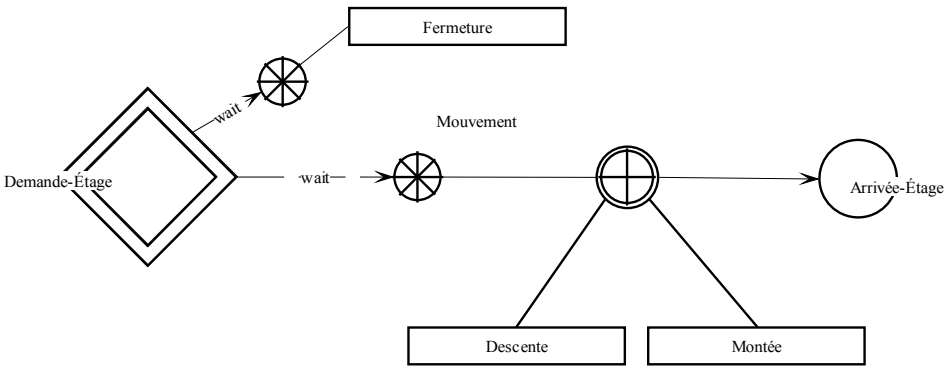


Figure 9 : Fragment événementiel "Demande-Étage"

Ce fragment événementiel représente un sous-ensemble d'une application de type « ascenseur ». Nous ne décrirons pas les différents formalismes utilisés, ce n'est pas l'objet ici, mais cet exemple permet intuitivement de comprendre le mécanisme utilisé.

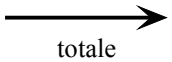
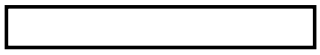
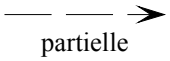
Fonctions	Opérateurs	Type d'Objets	Type d'Événements
 totale	 collection	 Arrivée-Étage représenté	 simple
 partielle	 union		 Demande-Étage externe ou temporel

Tableau 4 : Quelques exemples du formalisme IFO_2

Cet exemple illustre la richesse de représentation d'IFO_2.

1.4.4 Avantages/Inconvénients d'IFO_2

Un fragment facilite la conception de la dynamique du système en décomposant l'application afin de se focaliser sur un trait précis du comportement.

IFO_2 intègre un algorithme de transformation d'un schéma événementiel en règles ÉCA permettant d'envisager une dérivation automatique et complète des spécifications d'une application. Ainsi, cette méthode permet au développeur de définir des applications actives.

Nous avons souvent les défauts de nos qualités. IFO_2 n'y échappe pas et l'un de ses principaux défauts est la lourdeur de sa représentation. A vouloir y intégrer le maximum d'informations, chaque représentation devient d'une complexité importante rendant la méthode difficile à appréhender.

1.5 Synthèse des méthodes présentées

Historiquement, $(ER)^2$ est la première et reste toujours une référence dans le domaine. Le travail du concepteur est facilité de par les concepts de base utilisés (les schémas E/R sont connus de tous) mais aussi de par les évolutions qui en sont faites.

IDEA possède, pour sa part, une solide base ancrée dans les méthodes objets les plus renommées et une approche globale (allant de la conception à la dérivation en règles ÉCA) mais un peu trop restrictive (le champ d'application reste limité).

Les méthodes de rétro-conception proposent des solutions très intéressantes en amont de la ré-ingénierie. Leur utilisation peut permettre de démarrer ce travail dans les meilleures conditions en possédant l'ensemble des informations d'analyse nécessaires.

Les méthodes agents et particulièrement Gaia proposent une approche globale fortement inspirée des méthodes orientées objet. Mais, bien que séduisantes, les méthodes orientées agents ne peuvent être utilisées dans notre cadre d'utilisation où la ré-ingénierie sans modification de module est un maître mot. De manière plus générale, la complexité de ces approches et les modifications de l'existant qu'elles induiraient sont trop lourdes pour être utilisées pour de la réingénierie. Albert et i^* nous confortent dans l'idée d'utiliser les règles ÉCA afin d'automatiser la communication, et par extension la coopération, entres différentes entités d'un système.

Contrairement à $(ER)^2$, IFO_2 n'intègre pas les spécifications comportementales dans un schéma conceptuel (lié à des considérations structurelles). Cette méthode propose des modélisations statiques (schéma structurel) et dynamiques (schéma événementiel) bien distinctes mais néanmoins étroitement liées. IFO_2 bénéficie du fait que c'est une méthode complète allant jusqu'à la génération des règles ÉCA. Cette méthode nous a été particulièrement utile et nous en avons retiré de nombreux enseignements et inspirations. Néanmoins, IFO_2 est difficilement exploitable et difficilement adaptable à nos préoccupations. Elle contourne le problème de simultanéité des événements en posant une restriction trop forte. De plus, la lourdeur de sa représentation la rend difficilement exploitable, et encore plus difficilement extensible à l'incorporation des concepts de coopération dont nous avons besoin. Enfin, de par ses liens avec la méthode IFO [Abiteboul 87], IFO_2 est liée de trop près aux Bases de Données pour être étendue à d'autres domaines plus généraux.

Méthode	*IFO₂*	*Albert & i**	*Gaia*	*ER₂*	*IDEA*
Objet	**Oui** Les objets sont ici vu comme un concept symétrique aux événements. (modélisation structurelle)	**Non** C'est une méthode centrée agent	**Non** C'est une méthode centrée agent	**Non** Même si les entités et les relations exprimées dans le schéma ER modèlise des objets du monde réel.	**Oui** Avec support d'un modèle objet.
Évt	**Oui**	**Oui**	--	**Oui** Les événements permettent d'inclure des comportements de BDA dans des schémas ER.	**Oui** Expression dans le modèle dynamique (*statecharts*).
Règles	**Oui** Modélisation comportementale	**Oui** ÉCA pour gérer la communi-cation	--	**Oui** Elles peuvent intégrer des niveaux de priorité.	**Oui** Règles déductives pour les contraintes gérant les contraintes d'intégrité.
Coopéra-tion	**Non**	**Oui** Elle permet de réaliser la coopération entre agents.	**Oui** Au travers de différents modèles	**Non**	**Non**
BDA	**Oui** Les schémas événementiels IFO_2 sont traduits en règles ÉCA (directement inspirées de HiPAC).	**Non**	--	**Oui** Les règles ne peuvent intervenir que sur une BD (*Ingres*).	**Oui** Les règles utilisées dans IDEA ne peuvent que gérer des contraintes d'intégrité dans une BD (*Ode, DB2, Illustra*, …).
Lang. De Spécif.	**Non** Même si il existe un langage de spécification des conditions pour manipuler les traces.	**Oui** C'est le langage Albert qui permet la spécification	**À venir**	**Oui** Traduit les appels en SQL d'*Ingres* 6.3 (perdant certaines fonctionnalités au passage).	**Oui** Le langage conceptuel *Chiméra* permet un raffinement des modèles objets et dynamiques.

Tableau 5 : Récapitulatif des méthodes présentées

2. Applications distribuées : outils pour la distribution

2.1 Définitions

Les systèmes distribués émergent d'une combinaison de besoins nouveaux liés aux applications (besoins de rapidité, de fiabilité, d'autonomie, …). Les avancées techniques sur le marché des technologies de l'information ont généralisé ce type d'applications. Les communications de données sont maintenant rapides, fiables, sécurisées, et d'un coût faible.

Mais avant de décrire différents types d'architectures distribuées, voyons en une définition informelle mais néanmoins suffisamment précise pour se faire rapidement une idée :

Un système distribué est un ensemble d'ordinateurs reliés par un canal de communication. Les caractéristiques des ordinateurs ainsi que les liens de communication permettent de distinguer les systèmes entre eux. Comme les tâches peuvent être réparties sur les différents ordinateurs du système, un système peut souvent résoudre un problème plus rapidement qu'un seul ordinateur. Néanmoins, cette division des tâches introduit un problème : celui de la combinaison entre les résultats intermédiaires et les résultats finaux, nécessitant donc des moyens de coordination entre ordinateurs. Cette coordination différentie les systèmes distribués des systèmes mono-processeurs [Shands 94].

De manière plus précise, une application distribuée est une collection de *N* processus $p_1,p_2,\ldots,p_n$ reliés entre eux à l'aide d'un réseau. Le réseau permet, à l'aide de messages synchrones (l'émetteur attend que le récepteur lui réponde pour continuer sa tâche) ou asynchrones (l'émetteur envoie le message et continue sa tâche sans attendre de réponse), l'interaction des composants logiciels de l'application distribuée.

Le comportement de chaque processus consiste en un changement d'état local et un envoi de messages aux autres processus. Ces actions sont complètements contenus dans un algorithme local qui détermine aussi la réaction aux messages entrants. L'exécution concurrente et coordonnée de tous les algorithmes forme l'application distribuée. Enfin, l'état global de l'application est constitué par l'union des états des unités de traitement (dans le cas de systèmes distribués ne partageant pas de mémoire) [Bellissard 97] [Schwarz 94] [Tel 96].

Deux phrases issues de [Derycke 95] résument bien les avantages mais aussi les inconvénients que l'on peut tirer d'un tel système : « *le passage d'un individu/processeur à N individus/processeurs*

n'entraîne pas un gain de facteur N. Pour n processus, il peut même y avoir détérioration de la performance, dans d'autres cas, le gain est infini : un individu seul ne pouvant accomplir la tâche ».

Le premier problème de ces applications est celui de la cohérence des informations. Par exemple, une base de données peut être éclatée en divers sites, avec une duplication d'enregistrements pour des accès rapides et des sauvegarde en cas de problèmes. Les ordinateurs maintenant des copies doivent se coordonner afin de conserver la cohérence des informations. De ce niveau de coordination va dépendre la nature des problèmes résolvables par ce système [Shands 94].

De la division des tâches naît un autre problème nécessitant une communication et une coordination efficaces : celui de l'analyse des résultats intermédiaires et finals venant des traitements individuels.

Ce type d'application est utilisé par exemple celui de la téléconférence, des agendas électroniques, des environnements de développement, du travail à domicile par connexions distantes mais aussi des systèmes de production (pour la gestion des systèmes complexes), etc. Le champ d'application est immense et s'agrandit au fur et à mesure que les techniques et les services évoluent.

2.2 CHOOE

2.2.1 CHOOE en résumé

CHOOE [Lebastard 93]est tout d'abord un gestionnaire d'environnement distribué permettant la définition et l'évolution dynamique d'un réseau de processus. Dans cet environnement, la communication est obtenue : par l'envoi de messages synchrones ou asynchrones vers un ou des processus ou bien vers des ensembles dynamiques de processus alors même que la communication entre les éléments a déjà commencé. L'idée séduisante de CHOOE est, qu'à tout moment, un élément du réseau de processus peut se déclarer compétent ou incompétent pour un ensemble de services. Ainsi, les communications à destination d'un service nommé n'iront qu'aux processus concernés au moment de la communication.

2.2.2 Notion de réseau de processus

Le réseau est initialisé en même temps que son premier élément appelé *processus principal*. En effet, ce processus va être amené à connaître tous les nœuds du réseau et à posséder en permanence un certain nombre d'informations sur eux. Ainsi, il pourra répondre aux questions des processus sur leurs partenaires. Par exemple, lorsqu'un processus veut communiquer avec un autre non encore identifié, il va tout d'abord se renseigner auprès du processus principal qui identifiera le destinataire et retournera l'information. Il sera par la suite inutile de re-contacter le processus principal pour toute communication.

Chaque processus du système se voit attribuer un numéro correspondant à son numéro d'introduction dans le réseau en partant du numéro de processus principal (choisi de manière arbitraire par l'utilisateur) :

numéro processus entrant = *numéro principal* + *n* (n représente le nombre de processus présents dans le réseau.)

Afin de faciliter leur identification symboliquement par les programmes écrits au dessus de CHOOE, les processus peuvent aussi avoir un nom.

Une fois le processus principal initialisé, d'autres processus peuvent être ajoutés de deux manières différentes :

- Création provoquée par l'un des éléments du réseau,
- Déclaration par un processus existant.

Voyons sur la figure 1 un exemple de construction d'un réseau de processus :

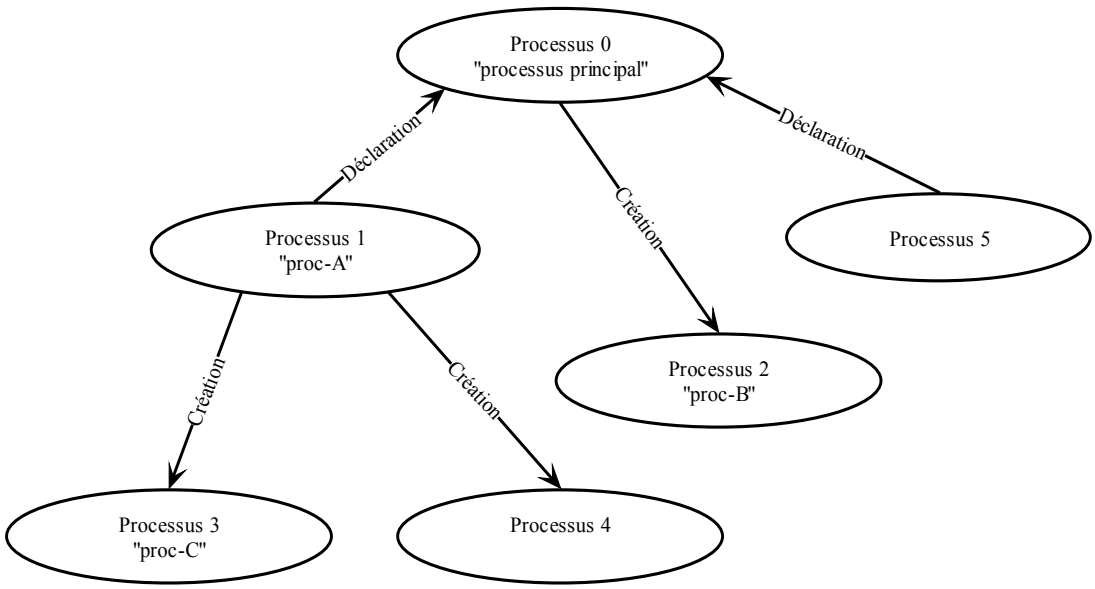


Figure 10 : Exemple de formation d'un environnement distribué

Le processus principal (numéro 0) est le premier à apparaître. Le processus suivant, le numéro 1 donc, s'est ensuite déclaré (proc-A). A la suite de cette déclaration, le processus principal crée un nouveau processus (proc-B, numéro 2). Par la suite, proc-A crée deux autres processus (numéros 3 et 4) dont un seul est nommé (proc-C). Enfin, le dernier processus à apparaître est le numéro 5 qui s'est déclaré sans nom précis.

Lorsqu'un processus devient un élément du réseau, il peut communiquer avec tous les autres partenaires, réalisant ainsi un graphe complet. Il est à noter que le réseau peut continuer à évoluer dynamiquement et ceci même si des dialogues sont engagés entre processus. Le prise en compte des modifications (ajout et retrait des processus) est immédiate.

2.2.3 Notion de services

Le processus principal connaît, pour chaque processus, l'ensemble des compétences des modules du système. Ainsi, un processus voulant envoyer un message à un service interroge le processus principal afin de connaître la liste des processus concernés. Une fois cette liste obtenue, et uniquement après cette étape, il peut leur adresser un message. Il est à noter que ce travail de récupération de liste est entièrement masqué à l'utilisateur.

Les processus faisant partie du réseau peuvent se déclarer compétents pour un ensemble de services (compétences), recevant ainsi tous les messages adressés au service en question. Lorsqu'un processus déclare une compétence, elle est prise en compte immédiatement dans l'environnement distribué et dès lors, des messages peuvent lui être envoyé.

Tout comme le réseau de processus est dynamique, l'ensemble des services déclarés par un processus peut aussi évoluer au cours du temps. Un processus peut ainsi se déclarer compétent pour un service et ceci même en cours d'exécution du système, réciproquement, il peut retirer un service de ses compétences.

Dans certains cas, un processus réalisant un service donné peut également recevoir les informations normalement adressées à un autre service (principe bien connu du « *qui peut le plus, peut le moins* »). Il est ainsi possible de réaliser des liens de dépendance entre services, des hiérarchies voire même des treillis de services.

Nous pouvons rapprocher la notion de service de celle de classe plus généralement rencontrée, de même qu'un lien de dépendance peut être compris comme un lien d'héritage.

Pour ce qui nous concerne, cette notion de service se rapproche de celle de groupe de travail dans la mesure où elle structure l'application en terme de compétences communes et où les membres d'un même groupe de travail devront recevoir toues les informations concernant ce groupe.

2.2.4 Communication

Comme pour toute communication, il est nécessaire d'identifier le ou les destinataires. Ainsi, CHOOE met à disposition une seule et unique commande *chooe-remote-call* comprenant trois arguments, le troisième étant bien évidemment le message lui même. De plus, CHOOE permet de communiquer de manière synchrone et asynchrone (différée ou pas), il faudra donc faire apparaître le mode de communication choisi.

Argument 1

L'identification du destinataire se doit d'être la plus complète possible car c'est en grande partie sur sa richesse et sa souplesse que l'efficacité du système est basée. Son rôle est d'identifier les processus cibles. Elle peut être exprimé de différentes façons, par :

- un nom ou un numéro de processus,
- un nom de service,
- un ensemble de numéros de processus et/ou de nom de processus ou de service,
- le mot réservé *all*. Tous les éléments du réseau reçoivent le message, y compris l'émetteur,
- une expression de sélection. Seuls les processus vérifiant la condition exprimée par cette expression de sélection sont concernés par la réception du message. L'expression de sélection est une fonction à un argument qui s'exécute chez le receveur. L'argument reçoit comme valeur une description du message et le corps de cette fonction peut faire appel à tout ce qui est nécessaire dans le processus receveur. Enfin, la fonction doit retourner un booléen. S'il est *vrai*, le processus reçoit le message et le traite. Dans le cas contraire, tout se passe comme il n'y avait rien eu. En fait, cette fonction se comporte comme une sorte de filtre qui décide si *son* processus reçoit le message en fonction du contenu.

Argument 2

Dans le mode de communication synchrone, le processus émetteur ne reprend son activité que lorsque tous les destinataires ont répondu. La fonction *chooe-remote-call* retourne les résultats en précisant éventuellement quel processus retourne quel résultat.

Dans ce type de communication se posent souvent des problèmes de blocage(*deadlock)*. En effet, les processus émetteurs sont verrouillés tant qu'ils n'ont pas reçu toutes les réponses. Un cas de *deadlock* immédiat peut se produire lorsque l'émetteur devient explicitement ou implicitement le destinataire. Ce cas est détecté par CHOOE lorsqu'un message est envoyé à soi même ou à un service auquel il est

membre. Il est aussi géré dans le cas ou l'option *all* est utilisée ou lorsque le destinataire est sélectionné à l'aide d'une expression de sélection.

Dans le mode de communication asynchrone, l'émetteur n'attend pas de réponse et continue son exécution dès la fin de l'appel de la fonction. Ce type de communication ne génère pas de *deadlock*.

Enfin, dans le mode différé, le principe est le même que dans le mode asynchrone, c'est à dire que le processus émetteur reprend son exécution immédiatement, toutefois, dans ce mode, des résultats sont attendus, ils arriveront de manière différée lorsque les destinataires auront déterminé et envoyé les réponses aux messages.

Lors du mode différé, *chooe-remote-call* retourne un identificateur permettant d'associer les réponses récupérées aux différents messages envoyés. L'émetteur récupère les informations dans une boîte aux lettres qu'il peut consulter de deux manières :

- par simple lecture non bloquante,
- par attente de réponse. Ce mode bloquant permet tout de même au processus de recevoir, traiter et répondre aux messages pouvant arriver pendant le blocage.

L'émetteur sait à priori s'il veut une réponse. Cela conditionne le mode de communication. S'il n'en veut pas, il émet en mode asynchrone et continue ensuite son exécution. Cependant, rien n'empêche au processus destinataire de réagir à un message en envoyant une réponse à l'émetteur.

2.2.5 Interruptions

Les processus peuvent se libérer mutuellement de tâches. Ainsi, un processus peut en interrompre un autre s'il juge que la tâche qu'il exécute n'est plus nécessaire , il est même possible d'interrompre le travail de tous les fournisseurs d'un service. Par exemple, lors de la résolution d'un même problème par différents modules, le plus rapide d'entre eux pourra stopper les autres puisque leur travail sera devenu inutile. Ils seront ainsi disponibles pour accomplir d'autres tâches.

En effet, chaque domaine de compétence est comme une sorte de *mail-liste* : un message est envoyé une fois à tous les abonnés, sauf s'il est explicitement reposté.

2.2.6 Implantation

La première implantation a été réalisée en Le_Lisp V15.25, version de l'INRIA [Chailloux 89] en utilisant la couche réseau RPC-LINK [Ilog 91].

2.2.7 Synthèse de CHOOE

L'un des principaux avantages de CHOOE en matière de gestionnaire d'applications distribuées est sa capacité à gérer à la fois des communications synchrones et asynchrones en ajoutant, qui plus est, un

mode asynchrone différé. CHOOE facilite l'identification de processus par noms symboliques, facilitant d'autant l'appel de services.

Une autre possibilité qui a retenu notre attention est la gestion dynamique des services. Un processus peut se déclarer compétent (ou inversement incompétent) en cours d'exécution à un service et donc recevoir (ou cesser de recevoir) les informations correspondantes.

Enfin, même si nous ne l'avons pas décrit ici, CHOOE possède une interface fonctionnelle (en *Le_Lisp*) contenant les fonctions de construction de l'environnement, les primitives de gestion de communication et enfin, les primitives de gestion des services (déclaration ou retrait de compétences).

2.3 Projet SIRAC

2.3.1 Introduction

Sirac a l'avantage d'une approche globale. Son champ d'activité couvre les environnements de développement d'applications, le support système pour la construction de serveurs d'informations, et les protocoles et services pour communications mobiles. Enfin ce projet bénéficie d'un soutien de nombreux laboratoires importants, citons :

- IMAG (Institut d'Informatique et de Mathématiques Appliquées de Grenoble)
- INRIA (Institut National de Recherche en Informatique et Automatique)
- Institut National Polytechnique de Grenoble
- Université Joseph Fourier, Grenoble-1

Sirac couvre un grand nombre de domaines. Nous n'allons pas décrire ici le projet dans son intégralité, mais nous allons juste prendre quelques parties directement en prise avec nos préoccupations et qui nous ont particulièrement inspirés.

2.3.2 Modèle d'événement pour la coordination

2.3.2.1 Introduction à la coordination

Comme nous l'avons entrevu dans les deux précédentes parties, il apparaît évident que la communication et plus particulièrement la coordination est un problème récurrent dans les applications distribuées.

Nous avons vu que la base de la coordination si situe dans les interactions entre les différents composants du système. Mais avant d'aller plus loin, il est nécessaire de se mettre d'accord sur le terme coordination :

Définition :

La coordination regroupe l'ensemble des techniques qui assurent la communication et la synchronisation entre des modules logiciels [Boyer 96].

Afin d'assurer la coordination entre composants d'un système distribué, un certain nombre de relations doivent être établies. Ces relations se font par le biais de connecteurs (points d'entrée/sortie) et doivent permettre :

- d'enchaîner une tâche d'un composant vers un autre,
- de diviser une tâche en *n* sous-tâches affectées à *n* composants différents,
- de synchroniser la terminaison de plusieurs sous-tâches avant qu'un autre composant ne reprenne le contrôle de la procédure,
- de « sous-traiter » la réalisation d'un service à un autre composant en attendant le résultat,
- d'informer un composant de l'accomplissement d'une tâche.

Toutes ces caractéristiques nécessaires à une coordination efficace requièrent que les entités à coordonner puissent communiquer selon deux modes :

- synchrone pour les demandes de services,
- asynchrones pour les notifications.

À delà de ces deux modes de communication, ces mêmes entités doivent pouvoir partager des données et doivent, d'une manière ou d'une autre, pouvoir s'identifier. Ces mécanismes de base, nécessaires à la communication et la synchronisation, se classent en trois familles.

- Partage et synchronisation.

Les composants du système distribué ont besoin de partager des données tout en respectant les accès concurrents. Ce partage permettra une communication cohérente par les données.

- Communication synchrone et asynchrone.

Lors d'une communication synchrone, tous les membres de l'interaction décident du moment où ils interagissent (une interaction synchrone ne nécessite pas qu'initiateur et exécutant se synchronisent, ce n'est qu'un cas particulier). La communication synchrone n'est pas le seul mode possible, il existe un autre type qualifié d'asynchrone, correspondant à l'envoi de messages et possédant un caractère plus impromptu.

- Désignation

Comme dans tout protocole de coordination, il est nécessaire de désigner le ou les destinataires. Cette désignation peut se faire par le biais d'identificateurs ou, lorsque cela est possible, par le biais de noms symboliques.

Le modèle de base utilisé, développé dans le projet Guide [Balter 94] et basé sur le partage d'objets, offre des mécanismes de partage et de synchronisation (par appel de méthode) utilisés pour la coordination. Afin d'offrir les différents types de communication nécessaires à la coordination, il est nécessaire d'inclure un mode de communication asynchrone réalisé à l'aide d'événements.

2.3.2.2 Communication événementielle

Dans la communication événementielle, le principe d'événement est un changement d'état du système entraînant l'émission d'un message par l'entité concernée. Elle est a priori (ce n'est pas toujours le cas) asynchrone car l'entité émettrice n'est pas bloquée en attente de réponse ou de délivrance du message. Ce type de communication permet l'abonnement d'entités déclarant (ou révoquant) un intérêt pour des événements déterminés. Lorsqu'une entité est abonnée à un événement, elle le recevra dès qu'il sera émis. Enfin, le dernier principe important est la réaction aux événements. La réception d'un événement par une entité abonnée a pour conséquence l'exécution d'une action associée lors de cet abonnement. L'occurrence de cette action est fonction d'une condition et de son contexte d'exécution.

2.3.2.3 Caractéristiques des événements

Les événements sont constitués d'un nom et d'éventuels paramètres en nombre variable. Lorsque la portée d'un nom d'événement est globale à l'ensemble du système, deux événements de même nom auront nécessairement la même signification. Enfin, un événement est dit « fugace » dans le sens ou il ne dure pas, il ne survit pas à sa diffusion ; il ne peut pas être sauvegardé pour une diffusion ultérieure mais peut l'être pour réaliser un historique qui servira pour des études ultérieures [Geppert 98].

Dans un souci de limitation de portée d'un événement, le mécanisme de désignation permet de restreindre la portée d'un événement à une application, une classe d'objets ou à un groupe d'utilisateurs.

2.3.2.4 Émission / Réception des événements

L'émission d'un événement est une opération asynchrone dans le sens où l'émetteur ne se trouve pas bloqué en attente d'un éventuel retour, contrairement à un appel de méthode ou à la levée d'une exception.

A la réception d'un événement est liée une réaction soumise à une condition. Pour son évaluation, la condition (qui est une séquence de code) peut utiliser les paramètres transmis lors de l'émission de l'événement mais aussi les éléments définis dans l'objet de destination.

En parallèle à cette étude sur la communication événementielle, une autre équipe de ce projet travaille sur une architecture de programmation par composants : Olan.

2.3.3 Modèle de programmation basé sur les composants : le modèle Olan

L'objectif de ce travail, mené au sein du projet Sirac, est de fournir des outils répondant à deux besoins :

1. Construire des applications réparties en combinant des techniques de programmation à base d'objets et des techniques d'intégration de composants.
2. Faciliter l'administration, la configuration et l'évolution de ces applications. Les applications coopératives, et notamment les collecticiels synchrones, constituent un domaine d'application privilégié pour l'expérimentation et la validation des outils.

Les objectifs visés par Olan doivent tout d'abord passer outre certaines contraintes auxquelles sont confrontés les concepteurs d'applications :

- La réutilisation et l'intégration de tout ou partie du code de l'application existante. Ceci est justifié par des raisons économiques mais aussi par des raisons de continuité vis à vis des utilisateurs.
- La configuration d'applications. Ce travail couvre deux étapes du cycle de vie d'une application. Il faut premièrement décrire les entités (composants logiciels) faisant partie de l'application puis les schéma de communication et de synchronisation entre ces entités.

Dans Olan, les applications coopératives permettent à plusieurs utilisateurs de travailler sur un ensemble de stations distribuées, de partager des informations et d'interagir en temps réel. Ces applications peuvent être considérées comme un ensemble d'entités (actives ou passives) que l'on peut appeler agents.

La construction de telles applications (applications coopératives distribuées) se ramène à la description des entités impliquées, à la définition des interactions entre elles ainsi qu'à l'évolution de cet ensemble au cours de son exécution.

Pour gérer tout cela, [Bellissard 95] propose :

- D'exprimer le contrôle de la dynamique de la structure des applications.
- De séparer la description des composants de celles des interactions. Selon le contexte d'exécution, l'occurrence d'un événement peut déclencher un certain nombre de traitements (appelés réactions) dans d'autres composants. Une notification d'événement peut déclencher ou non une (ou plusieurs) réaction(s).

Ainsi, dans Olan, l'interface d'un composant est décrite comme suit :

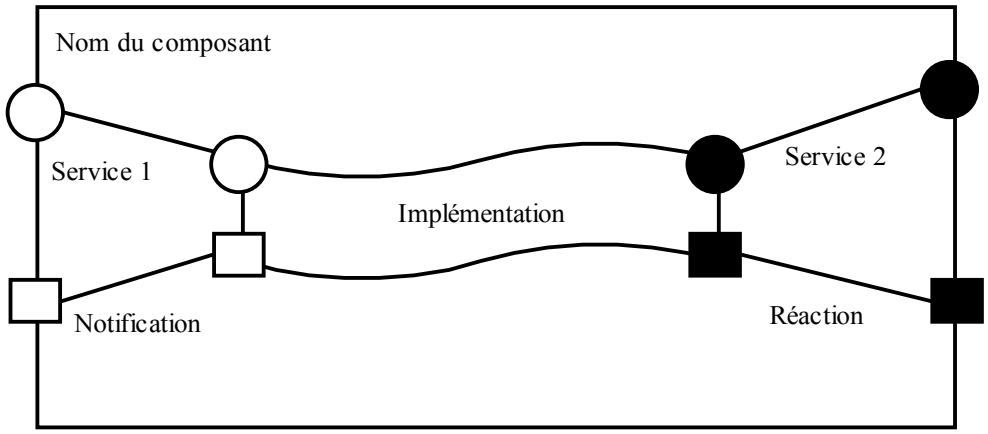


Figure 11 : Interface d'une classe "Composant" dans Olan

Les *services* sont des fonctions ou des procédures pouvant être exécutés par un composant. Les *notifications* sont des événements pouvant être diffusés pour déclencher des actions par d'autres composants. Chaque composant recevant une notification peut choisir de réagir ou non, c'est ce qui est appelé la *réaction*.

2.3.4 L'architecture CoopScan

Parallèlement à Olan, *CoopScan* a été développée. C'est une architecture définie par un ensemble de modules locaux et distants interconnectés. La réalisation de CoopScan est basée sur Olan.

L'architecture de CoopScan a été définie avec l'objectif de dupliquer une même application. Chaque nœud contient une instance de l'application partagée.

La mise en pratique de CoopScan (et par extension, de Olan) s'est réalisée au travers d'un logiciel de téléconférence. Dans ce logiciel, la collaboration se matérialise ainsi :

- Chaque action réalisée sur l'application partagée est acheminée au contrôleur de la conférence (le contrôleur fournit les services liées à l'accès aux applications partagées ainsi qu'à la gestion dynamique des intervenants). Si un utilisateur est autorisé à réaliser l'action, elle est effectivement réalisée et le contrôleur la diffuse à tous les autres contrôleurs.
- Les actions reçues des autres contrôleurs (donc des autres participants) sont exécutées sur les instances locales de l'application.

2.3.5 Conclusion

L'intérêt que nous portons à Olan et CoopScan réside dans le fait que les approches adoptent une philosophie similaire. Néanmoins, il existe suffisamment de différences pour que nous ne puissions pas réutiliser l'un et l'autre. En effet, Olan se base uniquement sur des composants logiciels développés pour l'occasion. A ce niveau là, notre approche diverge puisque nous partons sur le principe que les composants existent déjà, et qu'ils peuvent être aussi bien physiques (cellule de production par exemple) que logiciels.

Au niveau de CoopScan, même si le principe de coopération est adaptable, et c'est ce qui a motivé son étude (capture des actions réalisée pour informer le reste de l'application coopérative distribuée), les approches sont assez différentes. CoopScan est basé sur une réplication des applications partagées (applications formées de composants communiquant selon l'architecture Olan) qui communiquent entre elles. De plus, Olan ne permet de travailler qu'à partir de modules logiciels spécialement conçus pour cela alors que nous nous employons à réutiliser des modules logiciels et/ou matériels (comme des cellules de production) déjà existant. Notre approche diffère dans le fait que la coopération dans CoopScan est réalisée à partir d'applications partagées constituées de modules communicants, alors que pour nous, la coopération s'établit justement par cette communication entre les composants, et que l'application est conçue grâce à cette coopération.

Le lecteur intéressé par les travaux sur Olan et CoopScan peut se référer à [Bellisard 95b] [Bellisard 95c] [Bellisard 96b] pour Olan et [Atallah 95] [Bellisard 95a] pour CoopScan.

Enfin, le projet Sirac ne se limite pas à cette brève description et le lecteur pourra se référer à [Balter 95] [Boyer 96] [Balter 96] [Bellissard 96a] pour de plus amples explications sur le projet intégral.

2.3.6 Synthèse

Ce que nous venons de présenter permet d'éclaircir de nombreux points. Tout d'abord, dans Olan, la description des applications coopératives distribuées cadre parfaitement avec nos propres idées, nous confortant ainsi dans nos vues. De plus, Olan prône l'utilisation de composants logiciels pour la réalisation d'applications coopératives, ce qui là aussi va dans notre sens, de même que la description

des interfaces des composants. L'analogie s'arrête là puisque Olan s'oriente par la suite vers la surveillance des outils visuels pour son utilisation ainsi que la création d'un compilateur générant les squelettes des applications pour leur run-time. Néanmoins, à la manière d'IFO_2 qui nous a profondément inspiré dans notre approche méthodologique, Olan nous a apporté nombre d'éléments manquants y compris pour la réalisation de notre architecture coopérative.

2.4 EvE

2.4.1 Introduction

EvE (***Ev**ent **E**ngine*) est un moteur événementiel pour workflows dans un environnement distribué, c'est à dire que l'exécution des workflows est entièrement dirigée par les événements [Geppert 98]. Dans un tel environnement, les événements et les règles ÉCA sont des métaphores fondamentales servant à définir et à appliquer la logique des workflows.

2.4.2 Fonctionnalités

Les fonctionnalités nécessaires et supportées par EvE sont la prise en compte des événements, leur détection ainsi que leur notification à des logiciels distribués autonomes, ou à des composants logiciels représentant des entités du workflow. Enfin, EvE permet la gestion d'un historique des occurrences des événements ce qui permet le contrôle et l'analyse du workflow en cours d'exécution, mais aussi a posteriori.

La coordination par événements permet une spécification complète des traitements en utilisant des événements composés lors de traitements complexes. La coordination, si chère aux workflows, est assurée par la réactivité des entités de traitement (personnes et/ou applications). C'est à ce niveau que s'exprime toute la puissance de l'utilisation des événements et des règles ÉCA pour ce type d'application.

2.4.3 Architecture d'EvE

De nombreux chercheurs ont proposé l'utilisation des règles ÉCA fournies par les SGBD-A afin de permettre l'exécution des workflows. Bien qu'utilisant un SGBD-A centralisé, cette approche permet la distribution.

EvE représente une plate-forme d'exécution basée sur des événements capable d'intégrer des composants réactifs autonomes ou des entités de traitement (personnes, applications) communiquant au moyen d'événements. Le comportement de ces composants est exprimé à l'aide de règles ÉCA. Ces règles définissent quand et comment ces services sont exécutés dans le contexte du workflow. En réaction à des événements simples et composés, les règles ÉCA déterminent, dans leur partie action,

quels sont les brokers (programme régulant les échanges de messages entre les objets) concernés et leur notifient les événements. Les différents brokers représentent des interfaces utilisateur ou des applications externes.

Les événements simples sont des interactions entre brokers (requêtes, réponses, exceptions) et des événements temporels (absolus, relatifs, périodiques). Les événements composés, comme leur nom l'indique, servent à représenter des situations complexes en combinant des événements entre eux à l'aide d'opérateurs de séquence, conjonction, disjonction exclusive, répétition, négation (dans un intervalle de temps), concurrence (lorsque deux événement se produisent en même temps).

Cette utilisation de brokers d'événements implique qu'un certain nombre de fonctionnalités doivent être fournies par Eve :

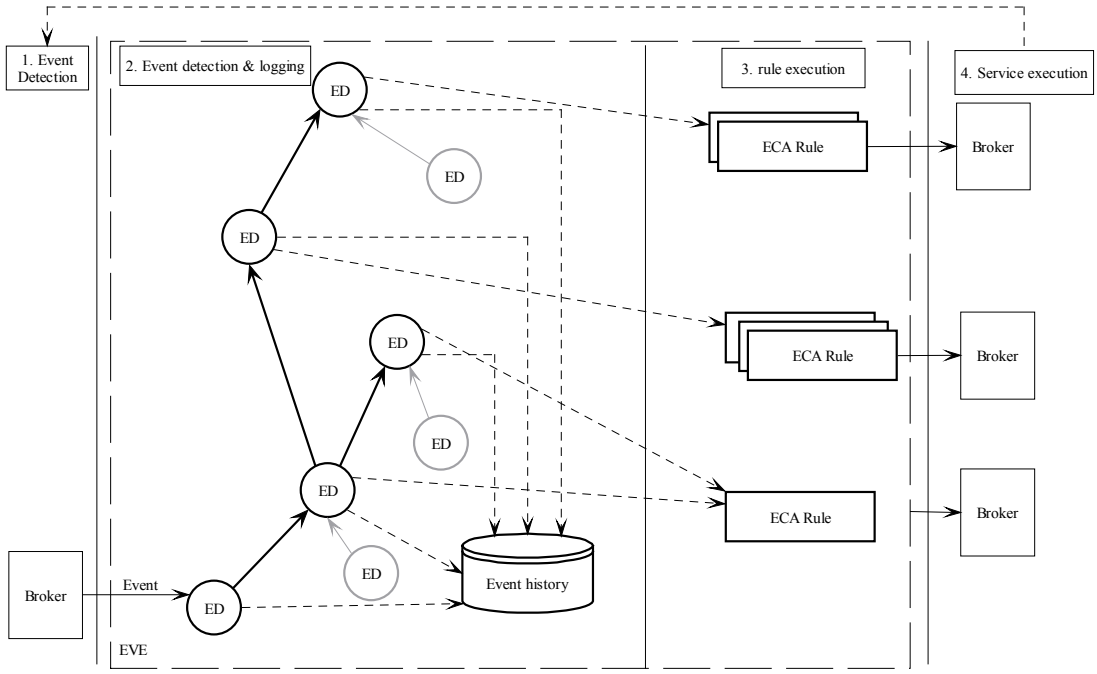


Figure 12 : Exécution des traitements dans EvE

- Les brokers étant distribués en réseau, les workflows peuvent être exécutés d'une manière distribuée. Ceci nécessite une détection d'événements distants ainsi que des facilités de communication entre le serveur de brokers (EvE) et ses clients (*les brokers*). La communication avec les brokers est réalisée à l'aide d'adaptateurs et de files persistantes d'événements, EvE se chargeant des non disponibilités et des erreurs dans la connexion aux brokers.
- Des méta-informations nécessaires pour les règles ÉCA et les brokers sont contenues dans une base d'exécution (*runtime repository*) et peuvent être modifiées au cours de l'exécution du workflow afin de permettre son évolution sans avoir à recompiler ou sans avoir à fermer puis relancer l'application.

- Afin de faciliter l'analyse à posteriori, EvE gère une base contenant l'historique persistant des occurrences d'événements (*event history*).

2.4.4 Synthèse

A l'aide de cet exemple, nous pouvons voir que les règles ÉCA ne servent pas uniquement aux BDA et qu'elles peuvent être utilisées dans des applications plus générales pouvant inclure elles mêmes des SGBD-A. Nous voyons aussi que l'idée d'utiliser des événements distants, combinée à des techniques récentes comme CORBA peut être une solution à la gestion de la coordination. Il est à noter que CORBA fournit un service de gestion d'événements (CORBA Event Services [OMG 94]) mais que celui-ci, est dans le cas présent, moins performant.

	Événements CORBA	EvE
Événements typés	Oui	Oui
Événements composés	Non	Oui
Paramètres d'événements	Un seul	Plusieurs
Diffusion	Oui	Oui
Règles ÉCA	Pas encore	Oui

Tableau 6 : Comparaison entre événements CORBA & EvE

2.5 Tempo

2.5.1 Introduction

Tempo [Belkhatir 93] est un environnement logiciel centré sur le traitement dont le but est d'assister le travail coopératif par une approche basée sur un modèle de communication. Tempo décrit un formalisme utilisé pour définir les activité de traitement. Les contraintes liées à l'utilisation des objets sont exprimées en utilisant la notion de rôle.

Le modèle de communication est basé sur des règles TECA (Temporal Event-Condition-Action). Elles ont pour rôle de contrôler l'échange des messages par un mécanisme de déclencheurs (trigger). Elles permettent de programmer des stratégies de synchronisation entre traitements en propageant les effets de l'exécution de leurs actions sur un ou plusieurs points de connexion.

L'union entre les connexions et l'environnement de travail rend possible le support de traitements coopératifs ainsi que le partage d'objets entre processus.

2.5.2 Aperçu général

Tempo est basé sur l'environnement Adèle [Belkhatir 91] et permet plus particulièrement :

- la coordination de ressources, problème lié aux objets partagés,
- la coopération entre agents partageant un modèle de traitement commun.

Tempo est composé de deux partie principales :

1. Un gestionnaire de ressources utilisant Adèle comme base d'objets persistants pour stocker les objets et les activités ainsi que pour réaliser une trace du projet en cours.
2. Un gestionnaire d'activité formé des règles TECA avec le mécanisme de déclencheurs.

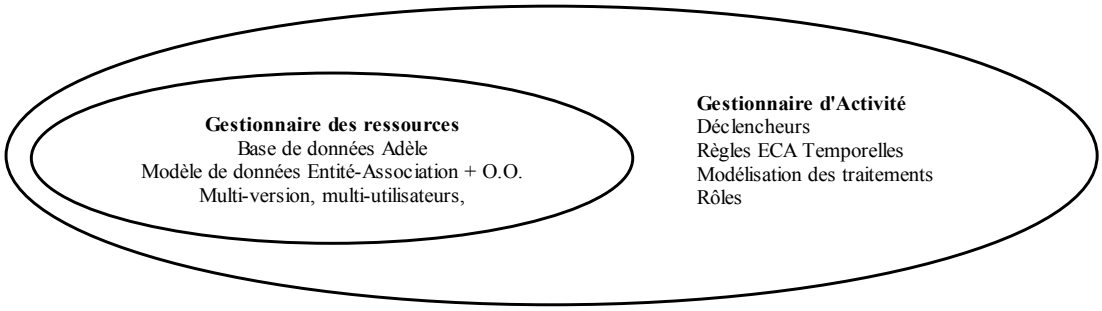


Figure 13 : Tempo

2.5.3 Contraintes temporelles

Les traitements sont contrôlés par des contraintes temporelles. Pour ce faire, il est nécessaire de les décrire et de les vérifier durant l'exécution du logiciel.

Les contraintes temporelles sont décrites dans Tempo par des règles ÉCA temporelles (TECA) similaires à celles utilisées par exemple dans HiPAC [Dayal 88]. L'interprétation et l'exécution de ces règles sont basées sur les déclencheurs de Adèle ainsi que sur son système de gestion d'objets.

2.5.4 Modèle d'exécution des règles temporelles

Les règles TECA sont définies dans le modèle de données et dans les modules de traitement. Dans le modèle de données, les règles TECA décrivent les limites d'intégrité qui sont dépendantes du contexte d'utilisation de l'objet. Elles sont aussi utilisées pour exprimer les stratégies définies dans le modèle de traitement : ordre d'exécution et synchronisation des activités, limitations d'accès à certaines ressources.

Une règle TECA est exécutée par un déclencheur Adèle à chaque fois que l'événement concerné est vrai. Une règle de ce type comporte quatre modes d'exécution :

- PRE {liste de triggers}
- POST {liste de triggers}
- AFTER {liste de triggers}
- ERROR {liste de triggers}

Le premier mode permet de réaliser une pré-condition (avant l'action principale), alors que le second agit comme une post-condition. Si la transaction est validée (commit), les règles associées au bloc AFTER seront exécutées, dans le cas contraire, celles associées au bloc ERROR seront exécutées.

2.5.5 Rôle des objets

Comme nous l'avons déjà dit, Tempo intègre la notion de rôle pour ses objets. Cette notion permet pour chaque occurrence de processus d'avoir des contraintes locales et des propriétés spécifiques pour chacun de ses objets.

L'incorporation de cette notion de rôle possède plusieurs avantages. Tout d'abord, elle permet l'intégration de divers types de comportements et de propriétés provenant de plusieurs types d'objets avec des caractéristiques statiques et dynamiques différentes mais avec une perspective unique. Le rôle permet d'associer à chaque occurrence de processus des contraintes et des propriétés locales. Cela permet aussi d'unifier le traitement d'ensembles hétérogènes d'objets. Enfin, il est à noter qu'un même objet peut jouer divers rôles en même temps pour des traitements différents.

2.5.6 Protocoles de communication

Dans de tels types de travaux, proches de l'ingénierie des logiciels, il existe une forte demande de coordination et synchronisation puisque les objets sont utilisés par plusieurs utilisateurs. Afin de pouvoir répondre à ces critères de coordination, de synchronisation et de multi-utilisateurs, il est nécessaire de se poser un certain nombre de questions :

- Quand, pourquoi et par qui ces objets ont-ils été modifiés ?
- Comment et quand ces changements doivent-ils être notifiés aux utilisateurs qui les partagent ?
- Quels sont les effets causés par ces changements ?
- Dans quels cas ces modifications doivent-elles être acceptées ou refusées ?

Ces problèmes ont fait l'objet de nombreuses recherches et particulièrement dans le domaine des bases de données. Plusieurs mécanismes ont été proposés et nous présenterons ici la solution développée dans Tempo [Belkhatir 93].

2.5.6.1 La coopération

Afin de permettre le transit d'informations entre utilisateurs, des mécanismes aidant et stimulant ces échanges doivent être fournis. L'environnement doit inclure un protocole de communication permettant aux utilisateurs d'être tenus au courant de l'activité, sachant que le nombre de processus exécutés concurrentiellement varie. Enfin, l'activité même liée à la collaboration se déroule en trois étapes :

- Notification à l'aide de laquelle les utilisateurs peuvent savoir avec qui, quand et sous quelles conditions ils peuvent échanger des données.
- Décision, résultat de la condition ci-dessus.
- Echange de données.

2.5.6.2 Synchronisation

Elle est nécessaire afin de contrôler les communications entre les utilisateurs. Ils doivent être capable de se synchroniser alors qu'ils poursuivent chacun leur activité. Sans elle, l'application coopérative ne peut pas assurer que les résultats échangés entre les utilisateurs sont corrects, en effet, au delà d'un « *certain temps* » il existe des divergences qui, si elles ne sont pas mises en évidence, empêcheront la poursuite du travail.

2.5.7 Proposition de support de la communication

L'attention a été portée sur la coordination, la collaboration et sur les stratégies de synchronisation entre les activités. Afin de permettre tout cela, Tempo inclut le concept de connexion à l'aide duquel les protocoles de communication entre les processus peuvent être décrits. Cette description autorise les différentes activités à se synchroniser contribuant à améliorer le niveau de coopération et de collaboration.

Les connexions permettent à deux logiciels de se synchroniser durant leur exécution en créant un canal de communication.

Chaque opération sur un objet (mise à jour, ajout, …) associée à des règles TECA va déclencher les règles correspondantes en réponse à ces événements. Lorsque deux applications sont interconnectées, leurs processus peuvent s'échanger des résultats, sans pour autant qu'il y ait de relation père/fils entre ces processus.

Par exemple, la mise à jour d'un objet O_1 de l'application A_1 peut déclencher des opérations sur un objet O_2 de l'application A_2 lorsque ces deux applications sont connectées. Ainsi, les connections peuvent être utilisées pour réaliser la collaboration entre deux ou plusieurs logiciels [Melo 93].

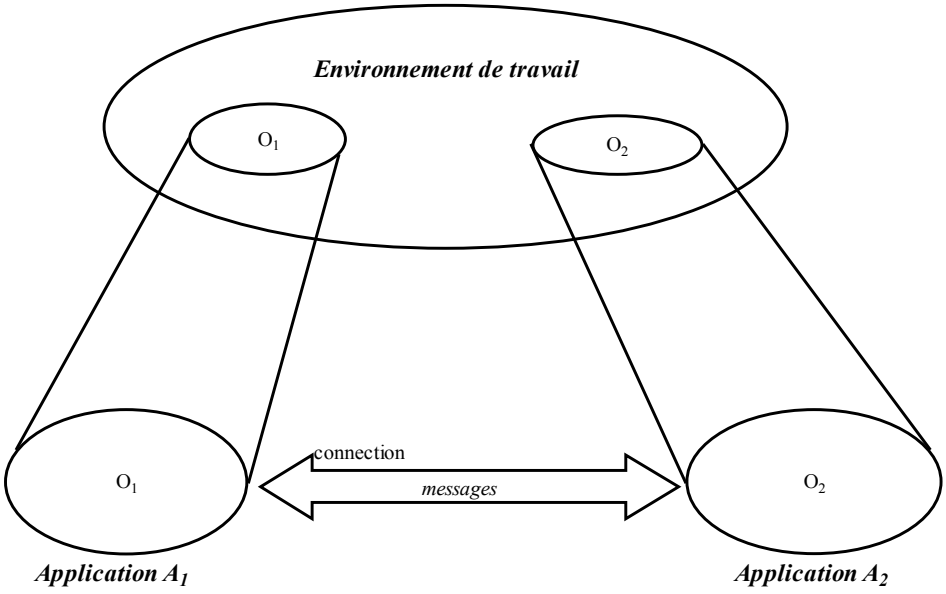


Figure 14 : Connections des entités dans Tempo

Pour chacune des connexions, il est possible de définir un ensemble de règles TECA permettant l'échange des données entre les entités concernées. Pour rendre cela possible, les règles collaboratives (règles TECA) doivent avoir accès aux objets manipulés par ces entités. Les connexions doivent aussi pouvoir suivre les opérations réalisées par ces objets.

Les règles ne sont pas simplement utilisées pour réaliser le contrôle de l'exécution. En fait, elles explicitent clairement les différentes activités qui peuvent ou ne peuvent pas être automatisées ou réalisées par les humains [Belkhatir 94].

2.6 Synthèse des outils présentés

Les outils proposé dans la section précédente nous permettent d'entrevoir ce à quoi pourra ressembler notre plate-forme coopérative. Ainsi, CHOOE présente une gestion dynamique (et distribuée) de services que peuvent utiliser des processus. Pour leur part, Olan et CoopScan nous montrent une réalisation basée sur les composants logiciels communicants spécifiquement développés pour les besoins de l'application. Enfin, Eve et Tempo proposent une approche originale basée sur les règle ÉCA (et leur pendant, les règles TECA) chargées de réaliser la communication. Ces différents outils préfigurent à eux cinq ce à quoi pourra ressembler notre architecture coopérative.

CHAPITRE 2

DES IDEES…A LA METHODE

Comme nous l'avons déjà mentionné dans la première partie de cette thèse, nos ambitions consistent en grande partie à proposer une méthode permettant la migration d'applications distribuées non coopératives déjà existantes vers des applications coopératives distribuées. Nous nous appuyons pour cela sur une plate-forme coopérative basée en partie sur le mécanisme de règles ÉCA développé pour les Bases de Données Actives.

Les applications que nous considérons sont composées de modules opératoires. Sont concernées, par exemple, les applications industrielles et plus généralement toute application distribuée composée de modules matériels et logiciels. Ce peut être par exemple une application de gestion d'alarmes où chaque partie logicielle gérant un ensemble de capteurs représente un module opératoire. Ce peut être aussi une application de gestion utilisant de nombreuses interfaces distribuées.

Afin de lever toute ambiguïté, le paragraphe suivant va brièvement préciser ce que nous entendons par travail coopératif.

1. Caractéristiques du travail coopératif

Selon le dictionnaire *Le petit Robert*, la coopération se définit par « *agir conjointement avec quelqu'un* » dans le but de réaliser un travail en commun. Cette opération ne peut se faire que par un échange d'informations entre différents opérateurs, entre opérateurs et systèmes ou bien entre systèmes. Elle suppose donc la formalisation d'une activité collective à la recherche d'un but commun comme par exemple la conception commune d'un produit.

Il est à noter que la coopération n'est pas forcément un processus engagé entre des personnes afin de régler un problème. Elle est le plus souvent mise en œuvre entre des fonctions [Roubellat 98].

L'apport la coopération peut se faire sentir à différents niveaux :

- Elle améliore la qualité de travail. C'est le cas par exemple lorsque plusieurs personnes travaillent ensemble afin de confronter leurs idées et de choisir la meilleure.
- Elle augmente les rendements en autorisant une cadence plus soutenue.

- Elle permet la réalisation d'opérations qui ne pourraient l'être sans elle. Lorsque plusieurs machines différentes participent à la réalisation d'une tâche elles coopèrent de manière à atteindre un objectif fixé. C'est en particulier le cas lorsque plusieurs compétences différentes sont nécessaires. Sans cette mise en coopération, la tâche ne pourrait être réalisée.

Un ensemble de processus participant à une même sous-tâche constitue *groupe coopératif* ou *groupe de travail*. Les processus de chaque groupe de travail communiquent essentiellement entre eux, formant ainsi un groupe fermé [Kaashoek 91]. Par ailleurs, une communication entre groupes doit être établie autorisant des travaux de plus grande envergure.

Cette notion de groupe, fondamentale dans le travail coopératif permet d'appréhender les interactions en terme de groupe et non plus en terme d'individualités. Plusieurs structurations de différents niveaux peuvent être envisagées dans la gestion des groupes : organisationnelle, décisionnelle, fonctionnelle, etc. Il est donc nécessaire de modéliser ces paramètres de coopération [Kaashoek 91] sachant par ailleurs que « *devant un domaine d'application aussi vaste, il est difficile de trouver un modèle formel qui modélise les différents types de collecticiels* » [Kanawati 97].

Les termes TCAO (Travail Coopératif Assisté par Ordinateur) et CSCW (Computer Supported Cooperative/Collaborative Work) ont été introduits pour désigner les applications développées pour des groupes d'utilisateurs communiquant au travers d'un réseau. On se référera à [Owezarski 96] pour une description complète du TCAO.

Dans notre cadre de recherche, ce qui nous intéresse dans ce type d'applications, ce sont les caractéristiques de coopération qui distinguent les différents collecticiels (logiciels de TCAO).

Certaines applications nécessitent la présence de l'ensemble des membres d'un groupe de travail Pour d'autres, cette présence instantanée n'est plus obligatoire. C'est le cas typique d'une coopération par mail où les membres peuvent travailler en différé.

Toutefois, dans la plupart des applications, on trouve à la fois des besoins en terme de coopération directe et différée.

D'un point de vue conceptuel, un système coopératif doit offrir un certain nombre de fonctionnalités :

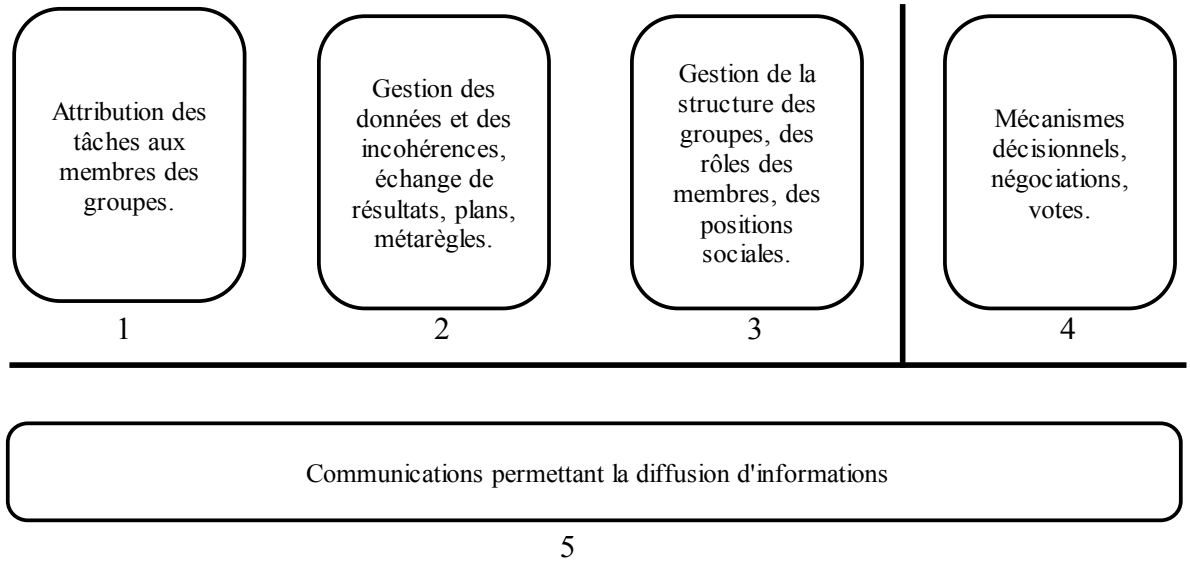


Figure 15 : Principales fonctionnalités d'un système coopératif [Villemur 95]

Les différents groupes identifiés chargés de la gestion des tâches, des données, des structures ainsi que des mécanismes décisionnels ne peuvent se réaliser que moyennant des services de communication efficaces permettant aux entités en coopération d'échanger des informations. Nous voyons sur ce schéma que les quatre structures fonctionnelles reposent sur une cinquième chargée de la communication.

L'exécution d'une tâche ❶ dépend de la configuration et des positions des membres du groupe coopératif. Elle nécessite de plus l'accès à certaines informations nécessaires à son exécution (données ❷, informations de contrôle ❸, …). Enfin, chacun des trois premiers modules peut utiliser le quatrième pour de la prise de décision ❹ ou pour gérer des conflits éventuels.

Les aspects de communication entre groupes ❺ ainsi que leur gestion sont directement issus du domaine de l'informatique distribuée.

« Entre l'informatique distribuée et le TCAO, la grande différence se situe dans le fait que pour ce dernier l'espace de travail est le moyen de communication. Un TCAO doit être distribué et utiliser une répartition par groupes de machines. Il doit aussi capturer et supporter les activités concernées par le traitement coopératif plutôt que de les cacher. » [Villemur 95].

La mise en œuvre de la coopération augmente fortement la quantité des contrôles concurrents nécessaires. Les systèmes existants fournissent un support adéquat et encouragent le développement de systèmes strictement asynchrones. Les problèmes surviennent lorsque l'on s'intéresse à des systèmes hautement synchrones. La distinction entre les systèmes locaux et distants n'est pas aussi

importante que la concurrence, néanmoins, elle introduit le besoin de supporter des plates-formes hétérogènes.

On qualifie généralement les applications coopératives par leur mode de couplage. Ainsi, dans les applications à coopération faiblement couplée, les utilisateurs tendent à travailler sur des objets partagés à différents moments. Les échanges sont généralement asynchrones avec peu de conflits d'accès et peu de concurrence.

Les applications à couplage modéré impliquent que les utilisateurs travaillent indépendamment, mais avec une connaissance des actions des autres utilisateurs sur les objets communs pouvant créer des conflits. Il est ainsi nécessaire de traiter les problèmes de concurrence, de même qu'il est nécessaire de prendre en compte la façon dont les actions des utilisateurs sont signalées au groupe.

Dès lors que le couplage devient fort, les problèmes sont nettement plus importants et les contrôles de concurrence deviennent plus conséquents prenant en compte le style des interactions à réaliser, la quantité de structure dans les interactions des groupes, et le type des objets partagés.

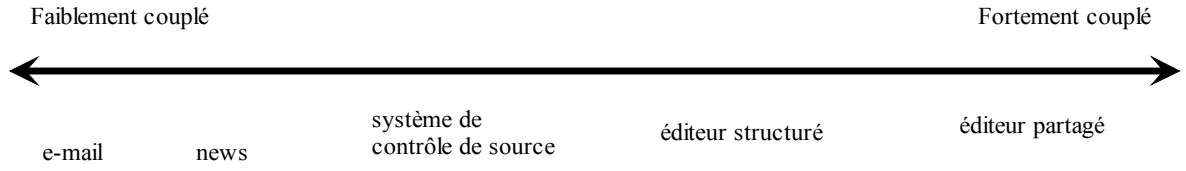


Figure 16 : Couplage de la coopération

Il paraît évident que plus la coopération est fortement couplée plus elle est efficace et, bien évidemment, plus on augmente le couplage plus les problèmes rencontrés s'accentuent. Dans notre optique, les applications auxquelles nous avons initialement affaire sont plutôt de type faiblement couplées, notre objectif étant de les amener à être le plus fortement couplées possible.

2. Les modules et leur environnement

Tout au long de cette thèse nous utilisons le terme de modules, il est primordial de bien cerner cette notion. Il nous semble important de préciser ce terme afin de lever toute ambiguïté sur la notion de module

Ce qui caractérise un module est sa possibilité de communiquer avec son environnement par quelque moyen que ce soit. Nous obtenons ainsi la définition suivante :

Module

Nous entendons par module tout dispositif opératoire (matériel ou logiciel) plus ou moins complexe, susceptible de réaliser une tâche et de communiquer avec son environnement.

Cette définition englobe aussi bien des éléments très simples comme des automates, des actionneurs ou des capteurs capables de produire et de recevoir des informations, que des éléments complexes comme des logiciels (ou des composants logiciels) ayant ces même compétences mais pouvant en outre utiliser un SI.

Un module se caractérise par son coté opératoire (il réalise quelque chose) ainsi, que par sa capacité à produire et à recevoir des informations à l'intérieur de son environnement.

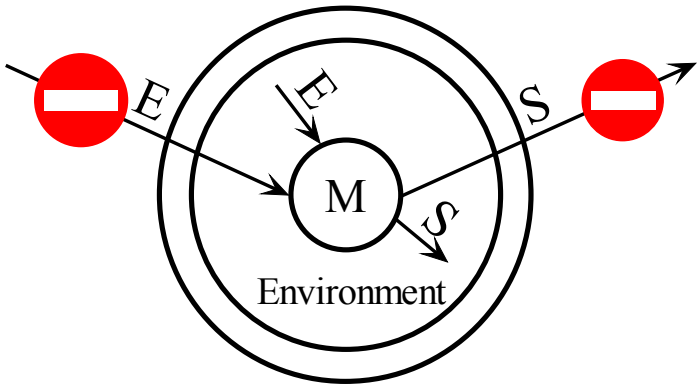


Figure 17 : Le module et son environnement

Comme nous pouvons le voir sur le schéma précédent, du fait de sa conception non vouée à la coopération, le module est totalement isolé à l'intérieur de son environnement et ne peut donc recevoir ou émettre directement d'informations hors de dernier .

Dans un cadre plus industriel, si l'on considère des capteurs sur une chaîne de montage, chaque information produite restera locale à son environnement, chaque information acquise sera puisée dans cet environnement.

De plus en plus, le besoin d'importer et d'exporter des informations au delà de l'environnement local se fait sentir. C'est ainsi qu'on a vu l'émergence des bases de données qui ont permis de répondre à ces besoins de partage d'informations entre environnements mais qui ont aussi apporté une réponse structurée et sécurisée au partage de données. Néanmoins, on ne peut encore parler de coopération à ce niveau dans le sens où les modules ne communiquent entre eux que par le biais du système d'information. Ils s'exécutent chacun de leur coté sans avoir ni connaissance, ni conscience de

l'existence d'autres modules. Ils n'ont, en aucun cas, une vision globale de l'environnement dans lequel ils évoluent. Il faut pour cela obtenir d'autres types informations reflétant l'état des modules existants dans le système, ce qui revient à les relier entre eux.

Dans une application non coopérative, même si les modules concourent à un même objectif, ils ne coopèrent pas et n'échangent pas d'information pouvant dynamiquement influer sur le travail des autres.

A défaut de communiquer entre eux, les modules communiquent (échangent des informations) avec leur environnement et parfois même d'environnement à environnement à l'aide d'interventions extérieures (personnes, programmes ou matériels spécifiques). Certaines de ces interventions extérieures peuvent être simplifiées par des procédures externes à l'application. Par exemple, de nombreuses sociétés utilisent des boîtes à lettres (type « Calvacom ») afin d'y déposer des fichiers qui seront récupérés à distance. L'utilisation de ces boîtes à lettres est une manière de faire communiquer deux environnements différents et, qui plus est, distants.

Prenons l'exemple d'une application logicielle non prévue pour fonctionner en réseau. Elle ira chercher ses données et les produira sur le poste où elle s'exécute. Les interactions sont réalisées au travers du système d'exploitation qui constitue l'environnement des modules formant l'application. Bien que les modules agissent sur le même système d'information par le biais du système d'exploitation, ils ne communiquent pas directement entre eux.

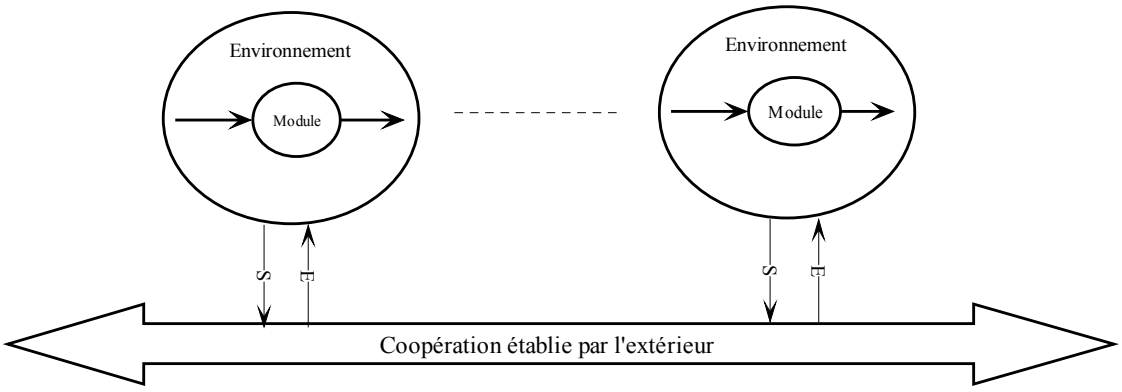


Figure 18 : Coopération établie par l'extérieur

Ce que l'on appelle la coopération établie par l'extérieur (ou coopération externe) est en fait la récupération d'informations produites par des modules et leur transmission par un quelconque moyen à d'autres modules concernés éventuellement après les avoir transformées ou en avoir fait une synthèse. Cette jonction peut être assurée par un opérateur ou un programme dédié. Si un réseau est

présent, cette opération peut être réalisée par un module dont l'unique but est d'acheminer l'information à d'autres modules. Dans le cas contraire, ce travail est généralement accompli en utilisant des boîtes à lettres consultables via un modem, par une copie sur un support amovible, ou bien par simple re-saisie par un opérateur. Cette coopération s'apparente à de la coopération de type faiblement couplée.

Dans un cadre « classique », cette liaison est fréquemment réalisée par des opérateurs extérieurs. Il est très fréquent de voir des factures sorties sur papier par le service commercial puis re-saisies par le service comptable, ainsi l'opérateur humain réalise la jonction entre les informations mais ceci oblige à des manipulations supplémentaires sources d'erreurs et de perte de temps. C'est aussi le cas avec des agences distantes qui placent/consultent des fichiers dans une boîte aux lettres ouverte ou encore lorsqu'un opérateur constate une anomalie sur un tableau de bord et qu'il actionne une commande afin de la corriger.

Une partie de nos objectifs est d'éviter que les communications d'environnement à environnement ne se fassent par des moyens « de secours » comme ceux cités précédemment.

Ces solutions complexes, lourdes et d'une fiabilité contestable ont été développées au coup par coup pour répondre rapidement à un besoin grandissant de coopération entre des applications existantes et sans avoir à réaliser de grosses modifications tant logicielles que matérielles. Elles deviennent maintenant un existant lourd à gérer et ne permettent plus de répondre aux évolutions nécessaires comme les besoins grandissants en coopération. Par conséquent, elles deviennent un fardeau pour l'entreprise. Néanmoins, ces mêmes entreprises ne veulent généralement pas se défaire de ces applications qui ont souvent été développées en interne depuis longtemps, qui sont fiables (la fiabilité de l'application est souvent grévées par le peu de fiabilité des coopérations externes) et pour lesquelles il est fréquent que les sources n'existent même plus ou ne soient plus raisonnablement maintenables (peu ou pas de commentaires, langages anciens que peu de personnes maîtrisent). Ce qui est vrai pour les applications l'est aussi pour le matériel que l'on a appris à connaître, à maintenir, à programmer et pour lequel les opérateurs hésitent à réinvestir le temps nécessaire sans avoir la certitude que le changement provoquera un gain substantiel rapport à l'investissement fourni.

Notre second objectif est non seulement d'améliorer les applications existantes en permettant un travail plus rapide et de meilleure qualité, mais aussi d'offrir de nouveaux services pour les applications à venir et pour l'évolution des applications actuelles.

La difficulté majeure de notre travail réside dans le fait qu'il est impossible de modifier profondément l'existant. Donc, la seule possibilité que nous ayons, est d'intervenir sur les informations nécessaire/produites par les modules. En modifiant la circulation des informations nous pourrons atteindre notre objectif et établir une vraie coopération. De plus, en agissant sur les applications par le biais de leur interfaces ou de leurs interactions avec leur environnement, nous pourrons fournir de nouveaux services.

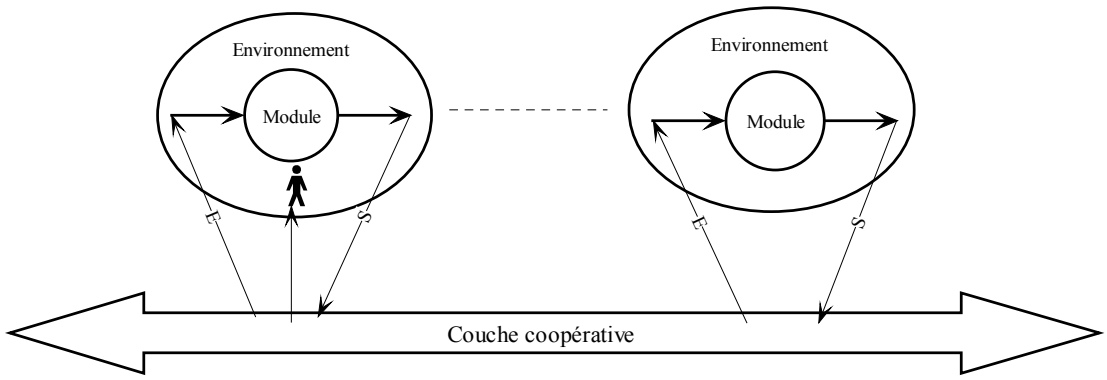


Figure 19 : Mise en coopération

Ce dessin illustre ce que nous désirons faire. Nous faisons disparaître la coopération extérieure en nous appuyant sur une couche coopérative. Son rôle est de capturer/fabriquer les différentes informations produites/consommées par un module dans son environnement (événements, E/S dans des fichiers, interactions sur l'interface, …) et de les incorporer (avec éventuellement des modifications, une complétion, …) dans l'environnement des autres modules. Plutôt que de manipuler les informations au travers de l'environnement, nous les traitons directement à la base, c'est à dire au niveau même du module ce qui permet de réagir au plus vite et au plus juste.

En agissant directement sur l'interface des modules, nous offrons à l'opérateur, lorsqu'il y en a un, la possibilité de réagir modifiant ainsi le cours de l'exécution et provoquant un changement dans le comportement de l'application. Le fait d'intervenir sur les interfaces des utilisateurs est un moyen d'accès indirect à des entrées de module par le biais de l'opérateur. C'est aussi un moyen d'ajouter des entrées à ces modules.

Ainsi, la couche coopérative permet une automatisation et une optimisation des échanges entre modules, et donc de la coopération en général. Elle permet en outre d'améliorer la coopération en fournissant des informations pertinentes impliquant une éventuelle réaction de l'opérateur, ceci pouvant par exemple faciliter l'aide à la décision.

En conclusion, nous dirons que nous nous efforçons de faire interagir autant que possible les modules entre eux, soit par l'échange direct d'informations au travers de la couche coopérative, soit par le biais de l'opérateur humain lorsqu'il y en a un. Cette automatisation va permettre un décalage sur l'échelle du couplage de la coopération en faisant glisser l'application d'une coopération (très) faiblement couplée vers une coopération « la plus fortement couplée possible ».

3. Organisation en groupes de travail dynamiques

Nous avons expliqué qu'une coopération efficace passe autant par une bonne communication que par une bonne organisation en groupes de travail. La composition de ces groupes est aussi importante que le travail lui même réalisé à l'intérieur par les modules.

Dans une entreprise, les factures éditées par le service commercial intéressent le service comptable (et peut-être d'autres), mais n'intéressent pas tous les services de l'entreprise comme par exemple le service d'entretien.

Afin de mettre en place une bonne communication, il est nécessaire de réaliser un découpage par zones d'intérêt (par groupes de travail). Cette idée, que nous reprenons, a été largement développée dans des travaux portant sur le CSCW (*Computer Supported Collaborative Work*). Ils ont montré son bien fondé ainsi que les avantages qu'il est possible d'en tirer (meilleure organisation, optimisation des ressources et des communications, flexibilité accrue, …). Bien entendu, un groupe de travail peut s'étendre sur plusieurs sites géographiques au travers d'un réseau tout en ne formant qu'une seule et unique entité virtuelle [Boyer 96] [Roose 98b].

Groupe de travail dynamique

Un groupe de travail dynamique est composé de modules, éventuellement distants sur un réseau, et travaillant à l'accomplissement d'une ou de plusieurs tâches bien déterminées. Sa composition (modules composant le groupe de travail) peut varier dans le temps de manière à s'adapter à la demande du système afin de répondre au plus juste aux besoins du moment.

Par exemple, lorsqu'une commande urgente arrive, il est nécessaire d'adapter rapidement la production. Pour cela, des cellules de production peuvent être totalement ou partiellement soustraites d'une activité moins impérative pour être assignées à une chaîne de fabrication plus prioritaire. C'est ce principe de groupes de travail, et plus encore, de groupes de travail dynamiques que nous retenons.

Une partie du comportement de l'application est contenue dans la cohérence de la composition des groupes de travail. À l'heure de la flexibilité dans les systèmes de production, les structures de groupes de travail rigides ne sont plus souhaitables. Nous proposons donc une gestion dynamique de la composition des groupes de travail permettant d'assurer au mieux la coopération. Outre la mise en

œuvre d'une communication entre divers modules, nous allons plus loin en permettant la gestion des entités virtuelles que sont les groupes de travail dynamiques. Ainsi, les modules ne recevront que les informations utiles à leur travail actuel, respectant ainsi le fameux adage qui dit que trop d'information tue l'information. Au delà de la caricature de la phrase précédente, cette organisation en groupes, totalement transparente aux modules, permet de réguler la circulation des informations, et de mieux cibler les destinataires potentiels. Une surabondance d'informations implique des tris et donc ralentit les détections et par conséquent les réactions face aux données pertinentes. Une bonne organisation des groupes permet ainsi une gestion optimale des modules évitant la perte de temps issue d'un nombre trop important d'informations devant être analysées puis traitées. Une bonne gestion de l'acheminement des informations permet de n'avoir à prendre en compte que ce qui est pertinent et de répondre ainsi dans le temps le plus court possible (ceci restant dans nos préoccupations de temps-réel).

Au delà du simple gain de temps occasionné par ce filtrage d'informations, il convient de noter que c'est le rôle même du module qui se verra modifié par son appartenance à un groupe de travail ou à un autre. Considérons par exemple un module chargé de piloter un dispositif physique. Si on le place dans un groupe de travail auquel n'appartient pas ce dispositif, les sorties qu'il produit ne sont plus envoyées au dispositif physique mais peuvent être dirigées vers un autre module. On peut ainsi lui faire jouer un rôle de simulation ou encore un rôle de contrôle vis à vis de celui qui pilote actuellement ce dispositif physique (tolérance aux pannes).

Enfin, les groupes de travail ainsi constitués peuvent à leur tour échanger des informations entre eux. Ceci signifie que la couche coopérative doit permettre la constitution et la distribution d'informations de groupe. Ces informations, mises en évidence lors de l'analyse de l'application en vue de sa mise en coopération, devraient être constituées à partir de celles disponibles au niveau des différents modules du groupe. Ce sont des informations synthétiques reflétant l'activité du groupe tout entier. Elles permettent la mise en relation des groupes et l'enrichissement du système d'information de l'application par des contenus d'une sémantique supérieure plus à même d'être utilisable ultérieurement. Ces informations seront, par la suite, exploitables pour gérer des problèmes de traçabilité ou de capitalisation de savoir faire de l'entreprise.

Nous pouvons maintenant représenter notre système par le schéma ci-dessous dans lequel chaque module adhère à un ou plusieurs groupes de travail. L'appartenance d'un module à un groupe se traduira par le fait qu'il deviendra, dès lors, destinataire des informations relatives à ce groupe.

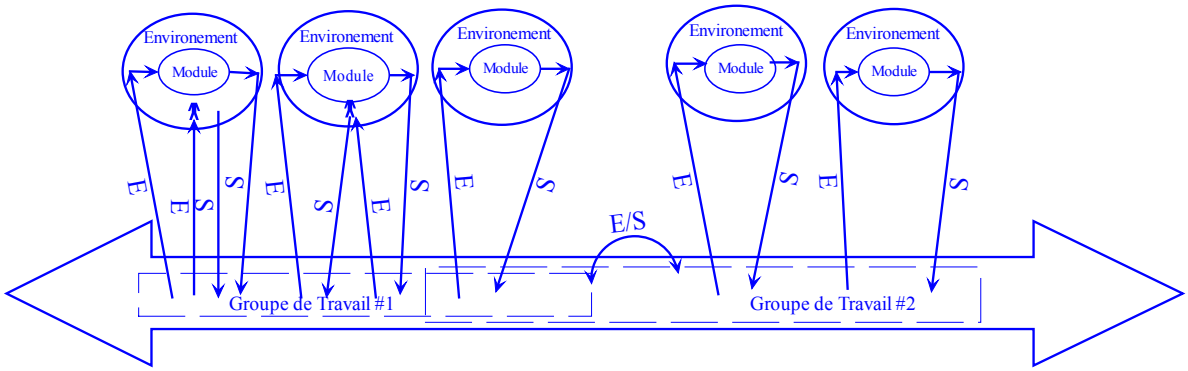


Figure 20 : Organisation en groupes de travail dynamiques

Lors de la création de ces groupes, il est nécessaire d'établir des règles précises. Un module ne peut pas anarchiquement entrer dans un groupe n'importe quand. Selon les entités, il y a des critères à respecter. De la même manière, la sortie d'un groupe se fait en respectant certaines règles établies en fonction, là encore, de l'entité concernée, mais aussi de l'état global de l'application que seule la plate-forme coopérative peut connaître.

Exemple . Un robot ne pourra être assigné à un autre travail qu'une fois la pièce ou le lot de pièces en cours terminés.

Exemple . Prenons un autre exemple et considérons un groupe formé de modules représentant des interfaces graphiques assignées à des postes de travail. Si trois interfaces (et donc postes de travail) sont suffisants pour réaliser l'activité, il est inutile et même dommageable d'en inclure d'autres.

Il faut donc assurer la cohérence dans la composition des groupes de travail dynamiques, de manière à ce que les entités présentes ne puissent pas se gêner entre elles (interblocages, sur ou sous-emploi, …).

Concernant nos travaux, ne pouvant intervenir sur les modules que nous réutilisons tels quels, l'entrée ou la sortie de l'un d'eux d'un groupe ne peut donc se faire à sa propre initiative. C'est l'environnement coopératif qui sera chargé de réaliser ce travail de manière totalement transparente pour les modules. Les groupes de travail ainsi formés représentent des entités entièrement virtuelles.

La gestion des entrées/sorties des modules de chaque groupe tient compte du niveau local (les modules) et global (les groupes) :

- Chaque module de manière individuelle interagit avec son environnement par des informations que la couche coopérative capture. En analysant les informations produites/reçues par les membres du groupe, nous pouvons connaître son activité et, par extension, son état.

Exemple . La couche coopérative peut décider d'exclure un module d'un groupe si celui-ci y possède une activité trop faible pour justifier de son appartenance.

- Comme nous l'avons précédemment dit, une partie du comportement de l'application est contenue dans la cohérence de la composition des groupes. En généralisant le point précédent à l'ensemble des groupes, les conditions d'entrées/sorties des modules dans un groupe peuvent ne plus être uniquement fonction du module mais de l'ensemble des modules du groupe, et de manière plus générale, de l'ensemble des groupes de l'application, c'est à dire de l'état de l'application.

Exemple . L'environnement peut décider de ne pas incorporer deux modules particuliers dans un même groupe pour des raisons de conflit entre eux.

En résumé l'organisation des groupes de travail répond à plusieurs préoccupations :

- Filtrage des informations. On ne fait parvenir aux modules que les informations dont ils ont besoin dans leur activité courante. On évite ainsi un trop plein d'informations reçues à un moment inopportun.
- Création d'informations synthétiques. En récupérant les informations produites par certains des modules d'un groupe, il est possible d'en créer de nouvelles reflétant l'activité du groupe tout entier, et non plus d'un seul de ses constituants.
- Spécialisation des modules. En modifiant les informations reçues et en filtrant celles émises, ou en faisant réagir l'opérateur par le biais de l'interface utilisateur, il est possible de modifier le rôle d'un module.

Jusqu'à présent, nous nous sommes attachés à montrer à quels niveaux nous intervenions pour gérer l'information nécessaire/produite par les modules, et comment nous l'utilisions dans les groupes de travail. Mais la communication est un vaste domaine ; il n'existe pas un seul moyen de communiquer mais plusieurs, chacun permettant de répondre à un cas précis d'utilisation. De même, il n'existe pas un seul type d'élément de coopération, mais plusieurs possédant chacun leurs caractéristiques propres. C'est ce que nous allons nous efforcer de décrire dans la partie qui suit.

4. Éléments de coopération

« *Le passage d'un individu/processeur à N individus/processeurs n'entraîne pas un gain de facteur N. Pour n processus, il peut même y avoir détérioration de la performance ; dans d'autres cas, le gain est infini : un individu seul ne pouvant accomplir la tâche* » [Derycke 95]. Cette citation d'Alain Derycke

met en évidence qu'en informatique (et ce n'est pas une exception), le gain n'est pas forcément proportionnel au nombre d'entités concernées. D'autres facteurs interviennent, dont un que tout un chacun s'accorde à placer en tête : la *communication*.

Dans toute coopération, l'aspect de communication joue un rôle essentiel. Il n'est pas concevable d'envisager une coopération quelle qu'elle soit et dans quelque domaine que ce soit, sans une communication efficace. L'accroissement des ressources impose une gestion plus délicate de celles-ci expliquant que ces gains ne soient pas linéaires (deux individus ne réalisent pas forcément un travail deux fois plus rapidement ou deux fois plus efficacement qu'un seul).

Ceci met bien en évidence les avantages que peut apporter la coopération mais soulève aussi le problème de l'efficacité de la communication pour que le travail en commun soit un gain et non une perte. Nous devons tout d'abord tenir compte des différents types de communication pouvant être rencontrés ainsi que de la manière de les aborder et de les prendre en compte.

Nous travaillons dans le cadre d'applications existantes. Les modules peuvent communiquer selon trois schémas différents : par événement, par message et par partage de données (stockées dans une base de données du système d'information).

Élément de coopération
Nous appelons éléments de coopération toute information (événement, message ou donnée) susceptible d'être utilisée pour mettre en relation les modules et les groupes de travail de l'application coopérative.

4.1 Communication par événements

Lorsque l'on parle de temps réel, l'idée qui vient tout d'abord à l'esprit est la notion de vitesse. Ainsi une communication de type temps réel veut dire que l'information transite dans un délai suffisant au bon fonctionnement de l'application, ce qui n'implique pas forcément une grande vitesse d'ailleurs, mais une transmission en temps voulu.

Plus le contenu d'un message est bref, moins de temps il mettra à circuler, et donc à arriver à destination. Dans les systèmes dits « critiques », on parle souvent de communication événementielle. L'événement se suffit à lui même, c'est à dire que son occurrence même est une information suffisante. On retrouve ce cas de figure fréquemment dans la programmation événementielle ou dans le domaine des systèmes industriels.

Exemple . Dans un environnement de production, à chaque fois qu'une pièce passe devant un capteur, celui-ci lève un événement afin de signaler ce passage. Ce signal sert à compter les pièces produites ainsi qu'à contrôler la cadence en temps réel. Ce simple signal de passage de la pièce suffit au traitement qu'il engendre.

4.1.1 Les événements dans la programmation événementielle

Dans la programmation événementielle, la circulation des événements issus des interactions de l'utilisateur se fait selon une organisation hiérarchique. Lorsqu'un événement est détecté par le système d'exploitation (S.E.), la widget (*WInDows gadGET* - Élément d'une Interface Graphique) concernée est prévenue par celui-ci.

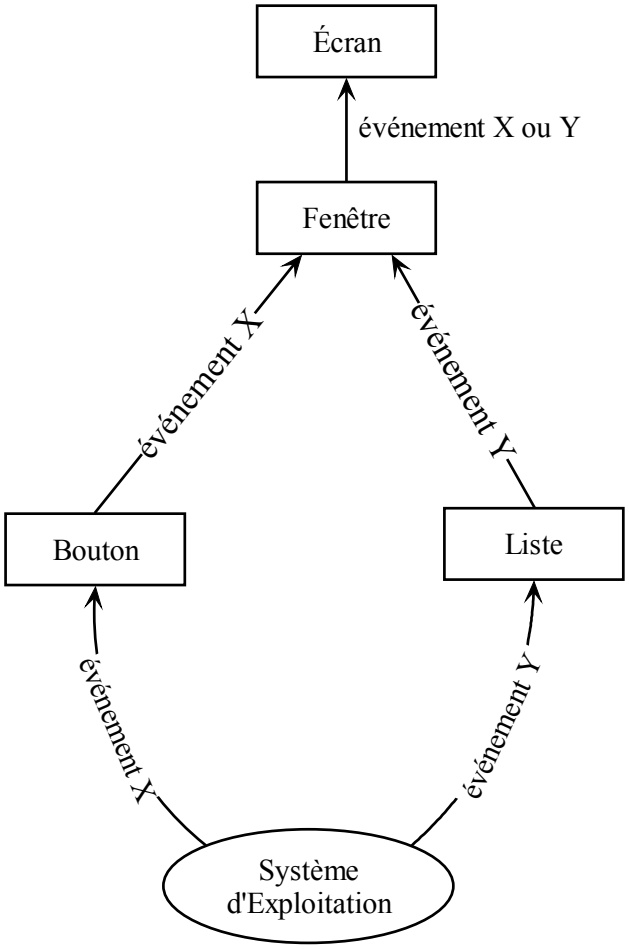


Figure 21 : Événements dans les interfaces graphiques

Lors d'un clic sur un bouton, celui-ci reçoit un événement le prévenant. S'il ne le gère pas, la fenêtre le recevra. Si elle ne le gère pas non plus, c'est l'écran contenant la fenêtre contenant ce bouton qui recevra cet événement. On remonte ainsi dans l'arbre, jusqu'à arriver au sommet (l'écran contenant cette fenêtre) où l'événement se perd s'il n'est pas traité.

Dans cette organisation, le schéma de communication est prédéfini. C'est selon ce schéma que l'événement est transmis aux objets susceptibles de le traiter. Cette organisation hiérarchique convient

relativement bien aux applications centrées sur l'interface, mais est insuffisante pour des applications coopératives générales où cet aspect hiérarchisé n'est en général, ni présent ni souhaitable.

Dans ce type de gestion des événements, il est très difficile de « flécher », de diriger l'occurrence d'un événement hors de ce schéma, c'est à dire de lui faire suivre un autre chemin que celui pré-défini. L'autre point important dans cette approche est qu'il n'y a pas de certitude qu'un événement produit soit traité.

4.1.2 Les événements dans les systèmes industriels

Dans les systèmes industriels, la gestion des événements est différente. Lorsqu'un événement est déclenché, il est traité directement par la procédure/tâche concernée. Contrairement à l'approche choisie dans les IHM (Interface Homme-Machine), un événement signalé à une tâche doit obligatoirement être traité.

Lorsqu'un capteur signale une température critique, une vitesse trop élevée ou le passage d'une pièce, la procédure de traitement ou la tâche associée reçoit cet événement et réagit en conséquence.

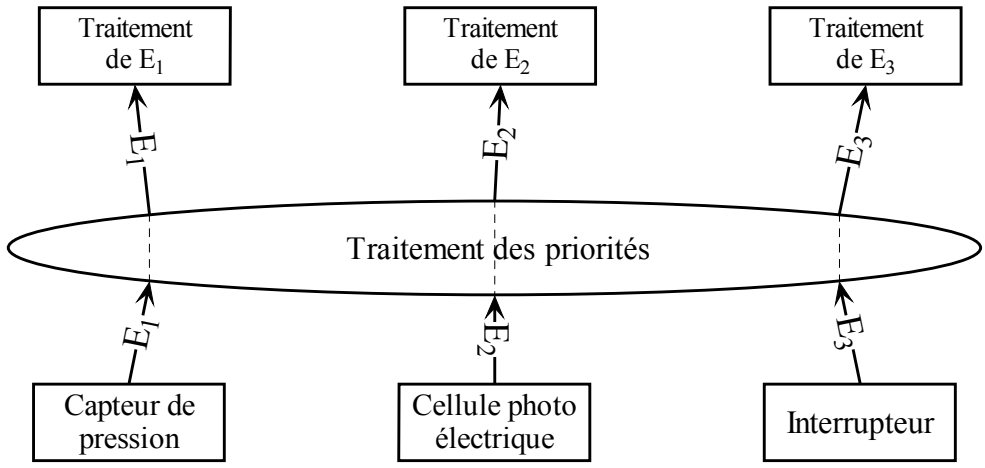


Figure 22 : Événements dans les systèmes industriels

C'est une circulation à plat avec une gestion de priorité permettant de réaliser une hiérarchie de traitement (hiérarchie non pas au niveau des entités de réception mais au niveau de l'ordre des traitements). La gestion des priorités dans les systèmes industriels est faite de manière explicite. Un numéro de priorité est donné à chaque événement. Lorsqu'il se produit, il est tout d'abord pris en compte par le gestionnaire de priorité qui réalisera si nécessaire l'ordonnancement entre les différents événements signalés. Ceci veut dire que lorsque deux événements E_1 et E_2 sont produits au même instant (l'instant est défini selon la granularité de temps nécessaire ou possible), si E_2 possède une priorité supérieure à E_1, E_2 sera traité le premier.

Dans les systèmes événementiels, lorsqu'une widget reçoit un événement, si elle ne le traite pas, l'événement remonte dans la hiérarchie. Cette gestion implémente de manière implicite une hiérarchie dans les objets. L'objet qui reçoit en premier l'événement est prioritaire par rapport aux autres. Plus on remonte dans la hiérarchie, plus la priorité est faible.

4.1.3 Approche hybride

Dans notre manière de gérer les événements, nous nous situons à la croisée des deux domaines cités ci-dessus. Nous ne faisons ni une circulation hiérarchique des événements, ni une circulation « à plat ». Nous procédons par diffusion en parallèle d'événements. Nous avons, de plus, une gestion dynamique permettant de faire varier les modules potentiellement récepteurs des événements.

Lorsqu'un événement arrive sur un module, ce n'est pas parce qu'au niveau inférieur il n'a pas été traité (programmation événementielle) mais parce ce que ce module en a besoin et donc qu'il contient les procédures de traitement correspondant à l'événement (systèmes industriels).

Toutefois, contrairement au schéma habituel, les événements sont diffusés et distribués dans toute l'application de façon à parvenir à tous les modules concernés qui les traitent en parallèle.

Ainsi, dans notre cadre de travail, les événements sont produits par les modules. Ils sont ensuite captés par l'architecture coopérative qui va les acheminer à d'autres modules pour lesquels cette information a une signification, c'est à dire pour lesquels ces événements constituent une entrée utile à la coopération.

4.2 Communication par messages

Les événements sont liés à la dynamique du système tandis que les messages sont liés aux données. Les messages représentent des informations notifiées par un module et, contrairement à la communication événementielle, dans la communication par message c'est le contenu qui représente la sémantique, et non la simple occurrence de l'élément d'information. La communication par message est moins tributaire du temps que celle par événement, sans en être totalement dégagée.

Ce type de communication est, là aussi, très fréquemment utilisé :

- par le mail, IRC – *Internet Relay Chat*, ICQ – *I Seek You*)
- dans le domaine des réseaux (trames diffusées pour la recherche de serveurs bootables),
- entre objets (appels de méthodes),
- client/serveur
- …

La distinction entre message et événement vient de la sémantique même des deux types d'information. Là où la sémantique de l'événement est contenue dans son occurrence, celle du message est encapsulée en son sein, permettant d'avoir un contenu sémantique riche pouvant exprimer des informations complètes et complexes.

Les messages font partie du système d'information, ils en constituent la partie non persistante. C'est un moyen de partager de l'information éphémère ou incomplète.

4.3 Communication par partage de données

Notre ambition dans la migration d'applications non coopératives vers des applications coopératives est de supprimer certains intermédiaires. Or, dans de nombreux cas, leur rôle est uniquement de réceptionner des informations, d'éventuellement les compléter et de les stocker et/ou de les faire transiter. C'est à ce niveau que nous nous situons en ce qui concerne les données.

Dans un système modulaire non coopératif, le SI peut être utilisé comme moyen de communication entre modules. Certains y introduisent des informations que d'autres iront chercher, modifieront, compléteront, voire même supprimeront.

La mise en coopération des modules et la création de groupes de travail permet de faire collaborer des modules à la création d'informations représentatives, non pas de l'activité de l'un de ces modules, mais de l'ensemble du groupe de travail. Dans une application non coopérative, de telles informations doivent être fabriquées à la main par saisie (ou re-saisie) d'informations synthétiques ou par des modules externes assurant un post traitement périodique de la BD.

Ainsi, par exemple, dans une usine, lorsque l'ensemble des pièces nécessaires à la fabrication de l'un des produits est réuni, le contrôleur de fabrication met à jour la base de données contenant l'état du stock.

A l'aide de la mise en coopération, les machines produisant ces pièces constitueront un même groupe de travail. A chaque pièce fabriquée, l'architecture coopérative récupérera les informations nécessaires se présentant sous forme d'événements ou de messages (pièces usinées, utilisées, …) pour mettre à jour elle même directement le Système d'Information. C'est ce que nous appelons la production coopérative d'informations.

5. Gestion de la communication

Concernant la communication événementielle et par messages, il existe deux manières de l'aborder, de manière synchrone ou asynchrone. Afin de lever toute ambiguïté, nous allons nous attacher à préciser ce que nous entendons par communication synchrone et asynchrone.

5.1 Communication synchrone

La communication synchrone permet de répondre aux besoins de synchronisation entre modules. De manière classique, lors de l'émission d'un message, elle implique un acquittement du destinataire. Tant que cet acquittement n'est pas parvenu à l'émetteur, celui-ci reste bloqué.

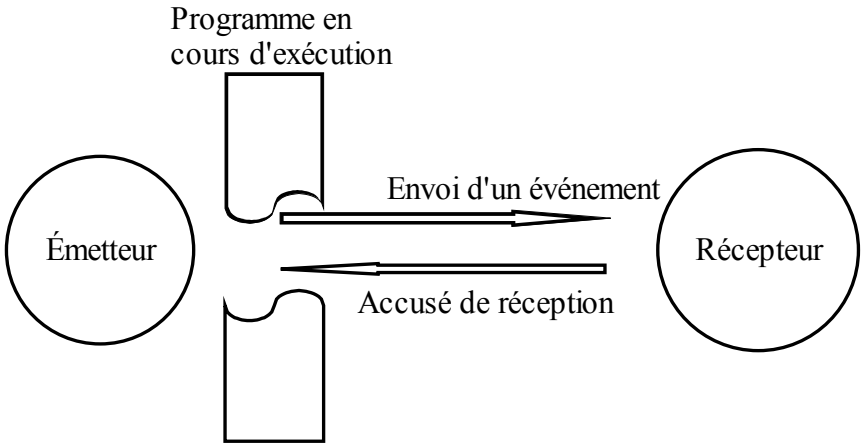


Figure 23 : Communication synchrone

Dans le schéma présenté, l'émetteur se synchronise au récepteur. Lorsque le récepteur renvoie l'acquittement, l'émetteur peut poursuivre son exécution.

Ce type de communication est fréquemment utilisé afin de synchroniser des processus dans des applications temps-réel. Prenons pour exemple la notion de *rendez-vous* du langage ADA. La tâche émettrice prend rendez-vous avec la tâche réceptrice et se met en attente jusqu'à ce que celle-ci accepte le rendez-vous.

Nous situant dans un cadre de réutilisation, nous n'avons en général pas la possibilité d'utiliser la communication synchrone au sens strict du terme. En effet, intervenant sur de l'existant par capture des échanges entre les modules et leur environnement, la gestion d'une communication synchrone (au sens envoi bloquant avec acquittement) n'est réalisable que dans la mesure où les modules prennent des rendez-vous avec leur environnement. Ce qui n'est généralement pas le cas dans une application non coopérative.

Aussi allons nous nous intéresser à une autre forme de communication : la communication asynchrone.

5.2 Communication asynchrone

Contrairement à la communication synchrone, l'émetteur de l'information n'est pas bloqué après l'émission, et n'attend donc pas une confirmation pour continuer son exécution.

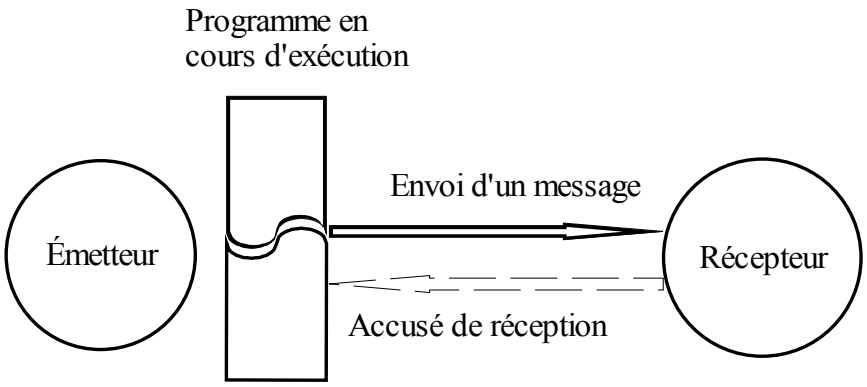


Figure 24 : Communication asynchrone

La différence avec la communication synchrone est que l'émetteur ne se bloque pas en attente d'un éventuel acquittement. En effet, celui-ci n'est pas obligatoire. Il dépend en fait du cadre d'utilisation.

Comme nous l'avons montré précédemment, il nous paraît peu probable pour l'instant d'avoir à assurer une communication synchrone, en effet, lors de l'émission d'un message, comment bloquer le module émetteur si celui-ci n'a pas été prévu pour cela ?

Nous préférons alors faire porter nos efforts sur la communication asynchrone tout en couvrant l'ensemble des communications possibles : événementielle, par messages et par partage de données.

Une remarque toutefois, si dans l'application deux modules sont synchronisés à l'aide d'une procédure externe, l'environnement coopératif assurera la circulation des événements et des messages permettant de réaliser l'acquittement nécessaire et donc permettant la synchronisation des modules. Nous pouvons donc affirmer que nous pouvons tout de même assurer une forme de synchronisation, car l'environnement coopératif que nous proposons (nous le développerons en détail plus loin) gère (prise en compte, diffusion) en temps réel et en parallèle les informations circulantes afin de répondre aux critères de vitesse souvent nécessaires.

6. Composition des éléments de coopération

L'automatisation de la coopération externe va nécessiter, mais va aussi produire des éléments de coopération (événement, messages ou données) d'un niveau sémantique élevé permettant justement de se substituer à cette coopération externe.

C'est ainsi que nous utilisons le concept d'événements composés et de règles détectrices issus de nos travaux antérieurs [Tawbi 95] et décrits dans l'introduction. Nous étendons ce concept à tous les éléments de coopération afin d'agir aussi au niveau des informations persistantes ou non. Nous pourrons ainsi réaliser des événements, des messages et des données composés à l'aide de règles détectrices utilisant d'autres événements, messages et données (ces éléments de coopération pouvant éventuellement eux même être issus d'une composition) [Roose 00a]

6.1 Événements composés

L'utilisation telle quelle des événements directement disponibles est un peu pauvre. Elle ne permet pas de réagir à des situation complexes constituées par exemple d'une succession ordonnée d'événements, ce qui permettrait d'avoir un contenu sémantique plus important. Ceci autoriserait à ne réagir que dans certains cas augmentant ainsi l'efficacité de la communication. Il est donc nécessaire de prévoir un mécanisme permettant de détecter un schéma d'occurrences bien précis, et non plus l'occurrence d'un seul événement. Ce schéma peut être une suite d'occurrences d'éléments de coopération distincts, une répétition dans un laps de temps précis, etc. Bref, toute situation envisageable à partir du moment où elle est algorithmiquement exprimable.

Nous retrouvons ce cas de composition dans l'événement « *double-clic* », qui est l'unique cas de composition d'événement dans le domaine de la programmation événementielle. Bien qu'il soit possible en utilisant une multitude de variables d'état de gérer la composition d'événements, tout programmeur qui s'y est essayé connaît la complexité de la chose.

Voyons un autre exemple simple d'événement composé :

Exemple . *Considérons un module gérant un sas de sécurité. Lorsqu'un capteur détecte une trace de fumée (événement levé par un détecteur de fumée) et que les différentes portes étanches sont fermées (émission d'un message par le détecteur de fermeture), il faut signaler que toutes les mesures de sécurité ont été prises (événement appelé « événement composé » résultant de la composition des informations précédentes).*

Dans notre souci de mise en œuvre de la coopération, nous allons parfois nous trouver confrontés au problème suivant : un événement est nécessaire, mais il n'est produit par aucune entité du système considéré. C'est le cas dans l'exemple précédent où l'événement signalant que toutes les mesures de

sécurité ont été prises est « fabriqué » par composition d'informations produites dans le système. Il n'est signalé par aucun des modules mais doit néanmoins être produit puisqu'il est nécessaire. Il faut donc le fabriquer et ce ne peut être fait qu'à partir des autres informations identifiées dans le système.

Cette possibilité de composer algorithmiquement des éléments de coopération pour constituer de nouveaux événements offre deux avantages particulièrement intéressants :

- Augmentation de la sémantique des événements. Un événement complexe est la composition d'un ensemble d'informations. Il permet de signaler des informations de plus haut niveau.
- Une bonne utilisation des événements composés permet de ne signaler que les situations pertinentes. Cette manière de procéder évite d'alerter les modules de manière intempestive (et parfois sans raison), optimise la circulation des événements et par extension, optimise le fonctionnement des modules. On obtient ainsi une meilleure réactivité de l'application.

6.2 Messages composés

Les avantages liés aux événements composés se retrouvent intégralement avec les messages composés. Mais du fait qu'ils contiennent des informations, leur composition produit des informations encore plus riches.

La composition des messages peut être réalisée à partir d'autres messages, mais aussi à partir d'événements (y compris à partir d'événements composés) et de données puisées dans le système d'information de l'application. Ainsi, en utilisant la puissance de la composition de messages, il est possible de :

- Réaliser une synthèse de plusieurs messages. Ceci évite aux modules de recevoir des informations redondantes et donc évite des traitements multiples.
- Compléter le contenu d'un message. Un ou plusieurs messages sont produits mais ne contiennent pas une information pourtant disponible dans le SI. Le rôle de l'opérateur externe serait alors de consulter la BD pour compléter l'information. En utilisant la composition de messages, il est possible d'aller récupérer cette information et d'automatiquement l'inclure dans le message résultant.
- Traduire un message produit par un ou plusieurs modules mais non compréhensible par d'autres.
- Différer la diffusion d'un message jusqu'à l'occurrence d'un événement le rendant pertinent.

6.3 Données composées

La composition de données s'apparente à la composition de messages, la persistance de l'information nouvellement créée en plus. Des informations partielles disponibles ça et là sous forme de messages, d'événements ou même de données pourront être synthétisées en une données du SI. C'est en général un travail qui était confié à un opérateur de saisie dans l'ancienne application. Prenons un exemple pour illustrer cela :

Exemple . *Un bon de livraison (BL) est composé de différentes informations ayant transité sous forme de messages à divers moments entre différents services et/ou personnes. En récupérant à sa source chacune de ces informations et en les complétant de données puisées dans le système d'information, il est possible de créer, lors de l'occurrence d'un événement indiquant la disponibilité des produits, une nouvelle donnée contenant l'ensemble des champs composant le BL.*

L'exemple précédent nous montre la création d'une donnée, le bon de livraison, qui peut utiliser à la fois, des événements, des messages et des données déjà existantes. De manière plus générale, la création de données par composition des différents éléments de coopération permet de :

- Réaliser la synthèse de plusieurs informations en une autre. Il s'agit là en particulier d'informations constituées par un groupe de travail et donc correspondant à une vision plus organisationelle qu'opératoire.
- Compléter des informations contenues au sein de la base de données au fur et à mesure de l'évolution de l'application et non plus à la fin comme c'est le cas lorsque l'on fait appel à un opérateur de saisie.
- Stocker des messages composés (traçabilité),

6.4 Intérêt de la composition

Les éléments de coopération composés ont une sémantique plus élevée. Ils résultent d'une synthèse exprimée à l'aide d'un algorithme appelé « algorithme de composition ».

Exemple . *Chaque colis transitant sur le quai d'un transporteur est renseigné (poids, volume, destination, ...). A chaque saisie sur un poste de travail, ces informations sont transmises une à une aux services commercial et comptable sous forme de messages. En utilisant une composition de messages, il est possible d'intercepter ces informations pour ne générer qu'un seul et unique message contenant l'ensemble des informations caractéristiques d'un colis.*

Dans l'exemple précédent nous n'avons pas amélioré la sémantique des messages mais nous avons optimisé l'échange de ces informations puisqu'un seul message par colis est transmis au lieu d'un pour chaque information saisie.

Exemple . *Si les tarifs sont stockés dans une base de données, on peut imaginer que l'algorithme de composition consulte cette base pour ajouter le prix du colis au message. Ce travail ne sera plus à faire par l'un des opérateurs des services concernés.*

Dans cet exemple, nous obtenons des messages à valeur ajoutée, ils ont été enrichis sémantiquement. Le message résultant n'est plus une reformulation des informations présentes en entrée mais il est enrichi de nouvelles informations puisées dans le SI.

La couche coopérative est constituée de règles détectrices s'exécutant en parallèle et captant les éléments de coopération. Elles produisent de nouveaux éléments de coopération, sémantiquement plus riches, permettant d'optimiser la circulation des informations, d'en enrichir certaines ou bien de produire des informations synthétiques (ou nouvelles) à partir de certains éléments de coopération. Elles permettent en fait de produire les éléments de coopération nécessaires au bon fonctionnement de l'application et non directement disponibles à la sortie des modules.

Bien entendu les aspects réseau sont omniprésents dans les moyens de communication. Ainsi, les événements, messages et données transitent si nécessaire au travers du réseau de manière transparente. Ils sont entièrement gérés par l'architecture coopérative qui s'occupe de les acheminer aux divers groupes et modules [Roose 97b].

Un des résultats de la méthode est de permettre de dériver les règles nécessaires à la création de tels éléments de coopération. Notre démarche va aboutir à une description de la coopération par un langage de spécification permettant, dans un premier temps, de réaliser les dérivations vers les règles mais aussi autorisant une vérification formelle de la possibilité de création des éléments de coopération. Une fois dérivées, ces règles seront intégrées à la plate-forme coopérative.

7. Avantages de cette approche

La mise en place de la coopération par une couche active présente nombre d'avantages. Le premier d'entre eux, en terme de génie logiciel, est la réutilisation. En effet, il n'est pas nécessaire de modifier profondément les modules déjà existants ni le système d'information de l'application. La couche coopérative tire ses informations de l'observation des modules en fonctionnement et de la capture des éléments de coopération qu'ils produisent. Ceci peut être fait depuis l'extérieur du module sans en modifier son comportement.

En outre, elle prend en charge, à partir des spécifications qui lui sont fournies et qui sont obtenues au terme du processus d'analyse que nous allons décrire maintenant, l'intégralité de l'organisation coopérative : interprétation ou constitution puis diffusion des informations ainsi que gestion des

groupes. Cette organisation coopérative devient totalement automatique tout en restant détachée des modules opératoires eux mêmes. Ceci permet une réorganisation rapide de l'application sans perturbation du fonctionnement (flexibilité).

La couche ainsi rajoutée est indépendante de la répartition géographique des ressources dans le sens où elle est elle même répartie sur l'ensemble de ces sites (règles réalisées par des tâches). Ainsi elle prend totalement en charge l'aspect réparti de l'application.

Comme nous l'avons vu dans la première partie, les différents travaux présentés permettent de mesurer la variété des solutions et la difficulté d'appréhender des applications de manière principalement événementielle. Les problèmes soulevés par le modèle ADELE/TEMPO [Belkhatir 94] font apparaître la nécessité mais surtout la difficulté de la modélisation dans le domaine des applications coopératives. Ainsi, *« devant un domaine d'application aussi vaste, il est difficile de trouver un modèle formel qui modélise les différents types de collecticiels* » [Kanawati 97], ce qui nous pousse pour l'instant, devant la variété des champs d'application, à nous limiter à la ré-ingéniérie d'applications pour lesquels les besoins de coopération se font sentir.

8. Organisation de la coopération

Nous ne présenterons pas ici en détail les techniques mises en œuvre afin de gérer les différents moyens de communication. Nous allons néanmoins nous attacher à décrire les grands principes qui permettront de comprendre la philosophie et les avantages de notre approche (la description précise sera faite dans la partie concernant l'architecture de la plate forme pour la coopération).

8.1 Gestion des groupes dynamiques

Forts des travaux précédents réalisés au sein de notre laboratoire, nous avons imaginé faire reposer la gestion des groupes de travail sur des règles ÉCA (Événement-Condition-Action), en nous appuyant entre autre sur les travaux de [Pandey 93] [Yu 95] [Boyer 96].

La gestion des groupes dynamiques, c'est à dire l'entrée ou la sortie des modules des groupes seront à la charge de ces règles ÉCA. Elles seront déclenchées par un événement significatif de l'évolution de l'application. Elles auront pour rôle de vérifier la faisabilité de l'entrée ou de la sortie du modules qu'elles ont en charge dans le groupe et de la mise à jour de la base de donnée contenant la constitution de les groupes de travail. Leur algorithme, et en particulier la condition qu'elles évaluent, permet d'assurer la cohérence des groupes et, au delà, de l'application elle-même.

8.2 Gestion des éléments de coopération

Il est, bien entendu, tout d'abord nécessaire d'assurer la circulation et la diffusion des éléments de coopération sur le réseau. Puis, il est surtout indispensable de savoir fabriquer, par composition, les éléments de coopération non directement disponibles. Il ne s'agit pas seulement de savoir composer entre eux des éléments d'un même type mais de pouvoir constituer un élément de coopération à partir de tous ceux disponibles dans l'application.

Dans ce but, nous allons étendre le principe de règles détectrices d'ADACTIF [Tawbi 95] à la composition d'éléments de coopération quelconques. Une telle règle se verra transmettre des éléments dont elle a besoin et évaluera son algorithme de composition pour produire l'élément de coopération résultant.

8.2.1 Intégration des événements

Les événement circuleront sur le réseau, dès leur arrivée, ils seront immédiatement pris en compte par la plate-forme et transmis aux entités concernées (modules, règles, …). Notre objectif est d'arriver à mettre en place une coopération explicite entre différents modules. Ce sera le rôle des règles détectrices qui, prises dans ce contexte, peuvent être vues comme ayant un rôle de supervision.

Ainsi, une règle détectrice sera caractérisée par un ensemble d'éléments de coopération devant lui être transmis et, un événement (appelé *Événement Composé*) qui sera le résultat de la composition.

8.2.2 Intégration des messages

La circulation des messages utilise la technique du *message passing* [Tagg 97] que l'on peut assimiler au mail. On envoie une information à un ou plusieurs groupes/modules. Les destinataires des messages pourront dès lors consulter le message lorsqu'ils le désireront, éventuellement en extraire des informations et réagir si besoin est.

Auparavant, nous avons utilisé des règles détectrices d'événements. Nous allons utiliser le même principe pour les messages permettant, à partir de plusieurs éléments de coopération de produire un seul message.

8.2.3 Intégration des données

Un autre type de communication très largement répandu est réalisé par le partage de données. Des entités créent, modifient, suppriment des données dans une BD que d'autres peuvent consulter, mettre à jour, etc.

Comme nous l'avons vu avec les événements et les messages, nous pourrons réaliser à l'aide de règles détectrices, la composition d'éléments de coopération afin de créer des données complexes ou bien afin d'éviter une mise à jour trop fréquente de la BD.

Ainsi, au delà de la stricte manipulation de données persistantes, nous pouvons utiliser les messages et les événements afin de synthétiser des informations qui, une fois complétées, seront stockées dans la BD.

Afin d'éprouver ELKAR, nous avons décidé de nous appuyer sur un cas réel permettant d'illustrer chacun des points que nous décrirons par la suite. Nous allons maintenant brièvement décrire le fonctionnement de l'entreprise qui nous a servi de support.

9. Présentation de l'exemple : Une entreprise de messagerie et de transport

L'exemple développé pour illustrer la méthode est basé sur une entreprise de messagerie. La messagerie n'est pas du transport de marchandises classiques. Elle manipule de très nombreux colis de taille modeste généralement inférieure à 1 m^3. Ces colis doivent être traités le plus vite possible, autant administrativement (avec une remontée d'information quasi immédiate) qu'au niveau physique. Tout colis immobilisé sur le quai ou tout camion insuffisamment rempli pour sa tournée est une perte d'argent. Tout retard, toute livraison erronée est une perte d'image de marque de l'entreprise et donc de clients actuels ou potentiels.

La messagerie a un impératif : la vitesse d'acheminement. Tout client (particulier ou entreprise) s'adressant à une telle structure veut l'assurance du délai le plus court. Tout retard ou toute marchandise livrée non conforme est refusée par le client qui se verra indemnisé par l'entreprise (c'est la gestion des litiges, un département hautement critique dans ce type d'entreprise).

Ainsi, pour les envois nationaux, les délais généralement constatés sont de 24 à 48h, pour les colis départementaux, ils sont de 24h. La concurrence faisant rage dans ce domaine, tout en gardant des coûts de livraison au moins identiques, sinon à la baisse, les entreprises de messagerie ont engagé des réformes afin de réduire les délais de livraison. Pour réaliser cela, elles ont engagé de nombreux frais en tapis roulants, appareillages de saisie (pistolets de lecture de codes à barres), appareils de mesures laser, réorganisation des quais, des équipes et des méthodes de travail, …

Ce type d'entreprise connaît un autre problème important : l'obligation de gérer la flexibilité. Il est très fréquent que de gros clients (conserveurs, entreprises de l'habillement, …) aient soudainement de fortes demandes de réception/livraison. Ce genre de problème est actuellement géré dans l'urgence et un peu…*comme on peut !* Si l'entreprise veut continuer à garder ces clients importants, elle doit mettre

en œuvre des techniques permettant de gérer au mieux les besoins urgents en établissant les priorités nécessaires et en jouant sur la flexibilité des équipes en temps réel.

La vitesse d'exécution (due aux volumes en constante augmentation) s'accroissant, les entreprises se sont rendu compte d'un autre problème : celui de la circulation d'information. Ils sont fréquemment ralentis par des problèmes d'informations n'arrivant pas à temps, erronées, incomplètes ou non transmises. Ils sont de même pénalisés par des problèmes de réactivité et de remontée d'information ; ce dernier point étant actuellement jugé crucial par les dirigeants de ces entreprises.

Il nous a semblé que nos objectifs étaient compatibles avec les carences mises en évidence ci-dessus et nous avons imaginé éprouver notre méthode (et ainsi voir ses points forts mais aussi ses points faibles, et ses limites) dans le cadre d'une entreprise de messagerie.

Après avoir pris contact avec l'agence *Messageries de l'Adour* – filiale du groupe *Laussuy Messagerie*, nous avons étudié leurs méthodes de travail et mis en évidence leurs points faibles, qui sont ceux présentés dans le paragraphe précédent. Nous avons ensuite éprouvé notre méthode sur l'agence Bayonnaise et nous avons réalisé un simulateur nous permettant de vérifier les concepts développés, mais aussi d'éprouver l'architecture coopérative dans son intégralité avec un cas réel en toile de fond.

9.1 Les colis

Un colis peut être amené par le client lui même à l'entreprise. Pour cela, il le dépose en spécifiant la destination. Une facture lui sera alors adressée. Dès son arrivée, le paquet sera renseigné (récupération de l'ensemble des informations nécessaires comme le poids, la destination, le volume, …) et sera déposé sur l'un des tapis d'acheminement.

Chaque colis est étiqueté avec un identificateur unique qui l'accompagnera jusqu'à la livraison chez le client. Cet identificateur permettra un suivi du colis en temps réel (à la manière d'*UPS - United Parcel Service - http://www.ups.com -* qui réalise un suivi en temps réel de ses colis par Internet).

9.2 Le cheminement des colis

Selon sa destination (régionale ou nationale), un colis sera aiguillé vers des tapis d'acheminement qui le dirigeront vers les zones de stockage correspondantes. Dans ces zones, les paquets sont regroupés par palettes européennes de dimensions fixes. Des opérateurs prennent alors le relais et réalisent le plan de chargement afin de remplir au mieux un camion (par exemple, pour un semi-remorque, un plan de chargement équivaut à 33 palettes européennes). La gestion informatique permet de connaître en temps réel l'état des stocks et l'état de chaque colis (à quelle étape il en est).

CHAPITRE 3

LA METHODE

Rappelons qu'il ne s'agit pas pour nous de concevoir ex nihilo des applications coopératives. Nous proposons au contraire de définir des outils de modélisation et de mise en œuvre permettant de rendre coopératives des applications existantes. Aussi ne nous intéressons nous qu'aux aspects de constitution et d'évolution des groupes de travail, ainsi qu'à la gestion des éléments de coopération (communication) entre les modules et les groupes sans intervenir sur les fonctionnalités de ces modules ni sur le système d'information. Nous supposons en particulier que les modules opératoires existent et que le système d'information (SI) de l'application est déjà défini.

La solution que nous proposons consiste à mettre en place une couche coopérative prenant en charge de façon automatique la distribution des éléments de coopération ainsi que la constitution et la gestion dynamique des groupes de travail. Cette couche devra par conséquent être capable d'identifier, d'étiqueter et d'acheminer les informations (événements, messages et données) ainsi que de déterminer les conditions requises pour qu'un module puisse/doivent intégrer ou quitter un groupe de travail. Pour cela, la méthode proposée permettra d'identifier ces éléments clés et de les utiliser dans la mise en œuvre de la couche coopérative à l'aide de règles ÉCA - Événement-Condition-Action [Dayal 88] ou détectrices [Tawbi 95].

La difficulté de notre travail réside dans le fait que nous travaillons dans le domaine de la ré-ingénierie. Ainsi, de nombreuses contraintes inhérentes à ce domaine qui seront exprimées tout au long de la méthode ne nous permettront pas toujours de réaliser ce que nous souhaiterions ou ce qui serait le plus efficace.

Afin d'illustrer au mieux notre méthode, nous avons décidé de nous appuyer sur un exemple concret. Pour cela, nous avons pris contact avec une entreprise de transport (*Groupe LAUSSUY*) réalisant de la « *messagerie* » et avons appliqué notre méthode afin de tenter de faire coopérer les modules existants. L'objectif était, tout en tenant compte d'un certain nombre de contraintes, d'améliorer la flexibilité des parties critiques (gestion performante des quais de stockage), la circulation de l'information (état des stocks en temps réel) et l'amélioration des états quotidiens (statistiques, chiffre d'affaires quotidien, stocks, …).

1. Présentation intuitive de la méthode

L'automatisation des coopérations extérieures, l'amélioration (quantitative mais aussi qualitative) des communications et les possibilités de flexibilité vont contribuer à l'amélioration de l'application. Dans la mesure où nous fondons la coopération sur la communication, la mise en évidence des informations produites par cette coopération et la mise en œuvre de groupes de travail constitués de modules, ces éléments trouveront leur place dans chacune des étapes de la méthode que nous allons décrire.

Étape 1

Le premier niveau d'abstraction que nous considérons est celui des groupes de travail. Nous y identifions les différents groupes susceptibles d'être présents lors de l'exécution de l'application en réalisant un découpage organisationnel. L'identification de ces groupes de travail est réalisée en respectant nombre de contraintes prépondérantes (dans certains cas, il faudra tenir compte de contraintes géographiques, dans d'autres, de contraintes horaires ou de méthodes de travail, …). Les groupes de travail correspondent souvent au découpage structurel déjà en vigueur dans l'entreprise lorsqu'il existe. Chaque groupe de travail représente une entité à part entière, c'est à dire qu'il remplit un rôle précis, nécessite un certain nombre d'informations et en produit d'autres. En résumé, un groupe de travail a un sens et une fonction au niveau de l'organisation.

L'objectif de la méthode étant de mettre en œuvre la coopération par le biais d'une communication efficace, nous allons, pour chaque groupe de travail, identifier les éléments de coopération : événements, messages et données échangés. Nous ferons une distinction entre les informations qu'ils nécessitent pour fonctionner et celles qu'ils produisent.

Étape 2

Le deuxième niveau d'abstraction est le niveau module. Nous y faisons l'inventaire de l'ensemble des modules présents dans le système et identifions les groupes auxquels ils peuvent appartenir. Nous utilisons pour cela notre connaissance du rôle joué par chaque module. Nous regardons ensuite les conditions de son entrée ou de sa sortie pour chacun de ces groupes.

Nous allons ensuite répertorier les événements, les messages et les données produits par chacun de ces modules ainsi que ceux nécessaires à leur bon fonctionnement à l'instar de ce qui a été fait à la première étape pour les groupes.

Étape 3

Lorsqu'un module a été identifié dans plusieurs groupes de travail, en plus des conditions d'entrée/sortie de groupe, il faudra s'intéresser aux contraintes d'appartenance à ce groupe, c'est à dire, regarder s'il existe des incompatibilités d'appartenance simultanée d'un module à différents groupes ou de différents modules du même groupe. Des règles de gestion de la dynamique (qui seront des règles ÉCA) seront utilisées pour gérer ces questions et réaliser l'entrée/sortie des modules dans et hors des groupes de travail dynamiques. Chaque module se verra ainsi systématiquement associé à une règle de gestion de sa présence dans chacun des groupes auxquels il peut appartenir.

Étape 4

Les éléments de coopération et les événements liés à la gestion de la dynamique des groupes ayant été identifiés, répertoriés et détaillés dans un dictionnaire établi au cours des étapes précédentes, nous tenterons de les relier entre eux. Ce travail de liaison va se réaliser aux deux niveaux précédemment cités (niveau groupes et niveau modules) mais aussi pour les règles gérant la dynamique des groupes. Ainsi, nous ferons apparaître, pour certains éléments, un lien direct entre le producteur et le consommateur de l'information.

En pratique, même un tel lien direct supposera parfois une mise en forme de l'information afin de la rendre compatible avec son destinataire. Des règles détectrices de la plate-forme coopérative prendront en charge, lorsqu'il est nécessaire, ce rôle de traduction qui permettra aux entités réceptrices de recevoir les informations dans le format et avec le contenu désiré.

Au terme de ce travail, certains éléments (événements, messages et données) nécessaires à la coopération ne seront pas encore reliés. S'ils s'agit d'entrées, cela signifie qu'ils ne sont produits par aucune entité du système considéré, à l'inverse, s'ils s'agit de sortie, c'est que pour l'instant ils ne sont pas utilisés dans la mise en coopération de l'application.

Nous utiliserons un mécanisme de composition (règle détectrices) afin de créer ces éléments manquants et d'établir ainsi les derniers liens permettant la mise en œuvre complète de la coopération.

Notre approche de la méthode ressemble jusqu'ici à ce qui est fait dans Olan (cf. chapitre 1), c'est à dire que le premier travail est la séparation entre la description des modules (composants dans Olan) et la description des interactions entre eux. Elle y ajoute l'expression de la structure et la dynamique des groupes de travail.

Étape 5

La définition, la mise en correspondance et la création des éléments de coopération ont été décrits au fur et à mesure des étapes précédentes à l'aide d'un langage de spécification. Avant d'envisager l'implémentation de la coopération ainsi définie, nous devons nous assurer de la validité du résultat obtenu. Pour cela, nous nous appuierons sur ce langage de spécification pour réaliser une vérification formelle de la complétude des éléments de coopération. Nous validons ainsi les étapes précédentes en nous assurant que les éléments de coopération à produire peuvent effectivement l'être.

Étape 6

Nous pouvons maintenant envisager la dérivation en règles nous permettant d'aller jusqu'à l'implémentation. Nous allons pour cela nous appuyer sur un langage de spécification qui permettra de réaliser cette dérivation en règles ÉCA et détectrices.

Chaque étape de la méthode enrichira la description de la coopération dans ce langage afin d'arriver, lors de cette dernière étape, à une spécification suffisamment précise pour directement définir les règles. Celles-ci seront ensuite incluse dans la plate-forme coopérative afin de pouvoir récupérer, acheminer et produire les éléments de coopération, mais aussi gérer l'appartenance des modules aux groupes dynamiques.

La plate-forme doit permettre de faire circuler les éléments de coopération, pour cela, il faut que le langage de spécification puisse nous donner un certain nombre d'indications concernant chacun des éléments de coopération. Ces informations nous permettront de pleinement qualifier un élément de coopération, c'est à dire de connaître pour chacun d'eux son type, sa source et sa ou ses destinations. On y trouvera en particulier :

- Le groupe ou la règle qui émet l'information,
- Le(s) groupe(s) de ou le(s) modules ou le(s) règles destinataires de l'information,
- Le type d'information : événement, message ou donnée,
- Le nom de l'information.

2. La méthode en détail

Dans cette partie, nous allons nous attacher à décrire les différentes étapes de la méthode. Nous allons aussi décrire le langage de spécification correspondant en expliquant comment nous allons l'utiliser afin de réaliser les vérifications nécessaires et comment nous l'exploitons afin de parvenir à la dérivations en règles [Roose 00d].

Notre méthode permet de revenir en arrière afin d'affiner cette décomposition (ajouter/supprimer des groupes, modifier la gestion de la dynamique des groupes). De manière plus générale nous verrons qu'il est possible de revenir sur chacune des étapes de la méthode afin de peaufiner l'analyse en particulier lorsque l'on ne parvient pas à mettre en place la coopération souhaitée par manque d'éléments de coopération ou parce que l'organisation choisie ne permet pas de synthétiser les informations désirées.

2.1 Étape 1 : groupes

Dans un premier temps, nous devons observer le fonctionnement actuel de l'entreprise ainsi que son organisation. Nous devons ensuite nous pencher sur son infrastructure et plus particulièrement sur la manière dont circulent les différentes informations nécessaires aux applications informatiques.

De ce premier travail, nous allons pouvoir extraire une organisation décomposable en groupes de travail susceptibles d'intervenir dans l'application. Il existe ensuite deux niveaux de circulation des éléments de coopération : les informations relatives aux groupes de travail - informations intra-groupes - et les informations circulant entre les groupes de travail eux mêmes : les informations inter-groupes. Nous travaillerons donc sur ces deux niveaux en démarrant du niveau le plus élevé, celui des groupes pour ensuite descendre dans le détail.

2.1.1 Identification des groupes

L'étape de décomposition en groupes de travail est primordiale puisque c'est elle qui décrit l'organisation de la coopération.

Les groupes de travail sont donc les éléments structurants du système. Afin d'identifier ceux présents dans l'application, nous devons en avoir une vision organisationnelle.

On ne se lance pas dans la ré-ingénierie d'une application sans une réflexion préalable. Nous n'avons pas l'ambition de pouvoir appliquer notre méthode dans tous les cas de figure, certains vont s'y prêter plus ou moins selon que l'entreprise possède déjà ou non une organisation séparée en départements, équipes, unités de fabrication, …

Le premier travail de l'analyste, avant même d'arriver à cette étape, sera d'étudier la faisabilité de la chose, c'est à dire, d'étudier l'organisation actuelle de l'entreprise et de considérer les découpages organisationnels. Après, et uniquement après que ce travail d'étude de l'existant ait été réalisé, l'analyste aura une idée de ce qui est faisable ou pas. Il peut donc entamer le début de la méthode : la décomposition en groupes de travail et leur identification.

Dans l'exemple choisi, l'identification des groupes est calquée sur la décomposition en services de l'entreprise. On pourrait aussi imaginer une décomposition hiérarchique, c'est à dire avoir des groupes de travail selon les responsabilités. Enfin, il est tout à fait envisageable de réaliser une décomposition mixte, combinant les deux types de coopération. En fait, tout dépend de ce que l'on désire, seul l'analyste, en fonction de l'existant, peut élaborer cette décomposition.

La dynamique de composition des groupes induit qu'un groupe de travail ne sera pas forcément composé de manière identique du début à la fin de son existence. Il est même fort probable, et là est l'intérêt de la dynamique des groupes de travail, que sa composition évolue de manière conséquente au cours du temps.

Exemple . Il est possible de constituer un (ou plusieurs) groupes pour la gestion de problèmes particuliers. Leur existence est alors directement liée à l'apparition de ces problèmes dans l'entreprise (pannes, sous-production, litiges, etc.). Lorsque rien de particulier n'est à signaler, ces groupes sont vides, mais peuvent voir leur composition modifiée à tout instant si les conditions nécessaires à leur inclusion sont réunies.

Afin de mieux comprendre notre démarche, nous allons illustrer chaque étape par des schémas que nous allons appliquer à notre exemple d'entreprise de messagerie. Nous allons démarrer en illustrant cette première étape de décomposition en groupes de travail.

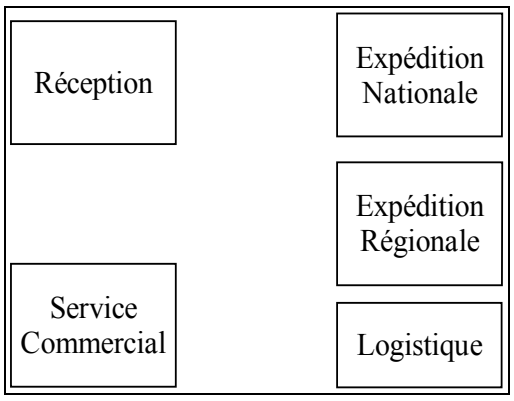


Figure 25 : Identification des groupes de travail

Nous mettons ici en évidence cinq groupes de travail : *Réception*, *Expédition Nationale (EN)*, *Expédition Régionale (ER)*, *Logistique (L)* et *Service Commercial (SC)*. Ces groupes correspondent aux séparations organisationnelles constatées dans cette d'entreprise après une étude « classique » de l'existant.

- *Réception*

Ce groupe de travail prend en charge les colis dès leur arrivée sur le quai de réception. C'est ici qu'ils sont renseignés (*renseigner* signifie obtenir leur destination, poids, volume, délai de livraison, client, etc.). Ce groupe de travail concentre l'ensemble des entités permettant d'obtenir les informations nécessaires à l'acheminement, à la tarification ainsi qu'au stockage des colis en transit.

- *Expédition Nationale et Expédition Régionale*

Selon leur destination, les colis sont ensuite dirigés vers les zones de stockage et d'expédition correspondantes (*Expédition Nationale* ou *Régionale)* où ils seront mis sur palettes européennes par des agents de quai et envoyés aux destinataires au cours de tournées nationales ou régionales. Si nous avons choisi de séparer l'expédition nationale de l'expédition régionale, c'est parce que les colis ne sont pas traités de manière identique, comptabilité et gestion des délais y sont différents. C'est en fait l'organisation du traitement qui est différente. Par exemple, les expéditions nationales sont acheminées par tractions nocturnes, alors que les expéditions régionales sont acheminées dès qu'une tournée est complète, c'est à dire dès qu'il y a de quoi remplir un camion où que le délai maximum de livraison est atteint.

- *Service Commercial*

Parallèlement à l'acheminement des colis, le groupe *Service Commercial* est tenu au courant de l'évolution de manière à éditer les bordereaux de livraison et les factures correspondantes. C'est ici aussi que se gèrent les tarifs, que les bascules comptables sont faites entre colis en traitement et colis traités. Enfin, c'est ici aussi que seront traités les litiges (colis refusés par un client pour raison de retard ou de casse, colis perdus ou volés, destinataires non identifiés, …).

- *Logistique*

Ce groupe s'occupe de la maintenance et de la supervision du parc de camions. Il constate aussi les éventuelles pénuries en signalant le type et le nombre de camions manquants (le SC pourra ainsi réagir et éventuellement faire appel à des correspondants ou sous-traitants afin d'assurer momentanément l'acheminement). C'est ici aussi que sont contrôlés les kilométrages des chauffeurs, des camions et les horaires de chacun des chauffeurs. C'est ici aussi que sont gérés tous les problèmes liés à l'étape de camionnage : les contraventions, les chauffeurs malades ou ayant atteint leur quota d'heures quotidiennes ou hebdomadaires, les camions en panne, …

Cette étape a défini une décomposition organisationnelle de l'entreprise sur laquelle se calquera l'application coopérative. Nous allons maintenant nous intéresser aux éléments de coopération

nécessaires au fonctionnement de chacun de ces groupes. Nous allons aussi nous intéresser à ceux qu'ils produisent au cours de leur exécution.

2.1.2 Éléments de coopération

Comme nous l'avons déjà mentionné, pour rendre les applications coopératives, nous utilisons trois moyens de communication : les événements, les messages et les données, chacun avec ses propriétés propres. Ainsi, pour chacun des groupes, nous devons connaître :

- les événements susceptibles d'être présents en entrée et en sortie du groupe, c'est à dire ceux dont le groupe a besoin pour fonctionner correctement et ceux qu'il est susceptible de lever,
- les messages qu'il peut recevoir, ainsi que ceux qu'il peut produire,
- les données à récupérer au sein du système d'information, mais aussi celles crées ou mises à jour dans le SI de l'application.

Illustrons cela par un exemple simple :

Exemple . *Le service de Réception est informé à chaque arrivée de colis (déposé par un client ou récupéré au cours d'une tournée). Ce colis est ensuite pesé, mesuré, étiqueté (destinataire, date et lieu de livraison, ...). Le service de Réception aura donc en entrée l'information correspondant à l'arrivée d'un colis, et produira en sortie les renseignements associés.*

Mais identifier des éléments de coopération n'est pas suffisant. Chacun sait qu'une énumération n'est rien sans une explication complète, précise et claire. Pour chacun des éléments de coopération, il est impératif de connaître sa sémantique, ce pourquoi il est identifié, c'est à dire qu'elle est sa fonction dans l'application. Nous devons par conséquent réaliser un dictionnaire appelé *dictionnaire sémantique* qui contiendra les indications sur lesquelles nous pourrons nous appuyer au cours des étapes suivantes.

Pour chacun de ces éléments de coopération (événements, messages, et données), il est impératif de réaliser une entrée dans un dictionnaire. Elle comportera son nom, son type ainsi qu'une description aussi détaillée que possible. Cette description sémantique portera sur son contenu (sauf dans le cas des événements bien évidemment), sur les conditions d'occurrence, sur les différentes manières d'interpréter cette information, son rôle, etc.

Le dictionnaire contient en somme toutes les informations qui ne peuvent être représentées graphiquement. Si les schémas que nous établissons sont une représentation organisationnelle et/ou

fonctionnelle, le dictionnaire est une représentation sémantique, ce qui en fait le complément idéal. Le concepteur n'aura de cesse de le consulter.

Une entrée dans ce dictionnaire pourrait être :

Exemple . *E_{CNP} (événement). Cet événement est levé pour chaque colis à destination nationale traité par le groupe Réception. Il permettra de prévenir le groupe des Expéditions Nationales. Cet événement ne peut être levé que lorsque l'ensemble des renseignements nécessaires ont été récupérés (date d'expédition, client, destination, ...) de manière à ce que le colis puisse être traité au plus vite sur le quai.*

Une information est souvent dépendante du contexte dans lequel elle est crée ou reçue. Une même information ayant un sens particulier dans le contexte où elle a été produite peut en avoir un autre une fois placée dans un autre contexte. Même si son contenu est identique, sa sémantique peut être différente et d'ailleurs l'information est généralement identifiée sous un autre nom. Ainsi, un même élément de coopération identifié dans un groupe de travail pourra signifier autre chose pour un autre groupe et y avoir un autre nom. On se basera sur le dictionnaire sémantique pour réaliser les correspondances.

Nous allons réaliser trois schémas, un par type d'élément de coopération. Nous représentons les groupes de travail par des boîtes rectangulaires identifiées par un nom. En entrée de chacune d'elles (coté gauche) sont représentées les informations entrantes, en sortie, ce sont les informations produites susceptibles d'être réutilisées par d'autres groupes de travail.

Pour chaque groupe, nous identifions de manière exhaustive toutes les informations qu'ils nécessitent ainsi que celles qu'ils produisent. Ainsi, au terme de cette étape, nous disposerons d'une représentation graphique pour chacun des trois types d'éléments de coopération.

Enfin, chaque élément de coopération donnera lieu à une entrée dans le dictionnaire. Ainsi, non seulement nous connaîtrons le groupe émetteur ou destinataire de chaque information mais nous connaîtrons également son sens et les critères à remplir pour que cette information soit valable, pour qu'elle puisse être produite ou prise en compte, etc.

Nous allons maintenant nous employer à mettre en évidence les éléments de coopération nécessaires ainsi que ceux produits par chacun de ces groupes.

2.1.2.1 Les événements

Nous allons tout d'abord sur notre exemple, représenter les événements. Afin de ne pas surcharger les dessins, nous ne représenterons que les groupes étant la source ou la destination événement. Ainsi, dans la représentation suivante, n'apparaît plus le groupe *Logistique* car il ne produit aucun événement et n'est pas non plus la cible d'au moins un événement.

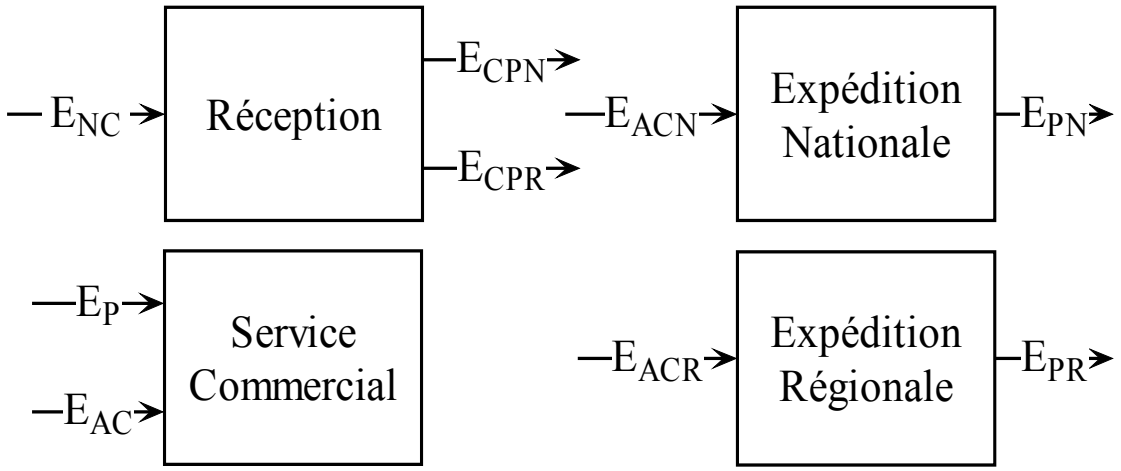


Figure 26 : Niveau Groupes - Événements nécessaires/produits

Ce schéma présente, pour chaque groupe de travail, les événements auxquels il peut être sensible et ceux qu'il peut lever.

- **E_{CNP} (événement)** : Événement qui signale l'arrivée d'un nouveau colis à traiter.
- **E_{CNP}, E_{CRP} (événement)** : Événements produits par le groupe *Réception* indiquant qu'un colis à destination nationale (E_{CNP}) ou régionale (E_{CRP}) est prêt (un colis prêt veut dire qu'il a été renseigné c'est à dire que l'expéditeur, le destinataire, le lieu de destination, son poids, sa taille, etc. sont connus). Cet événement est signalé par l'opérateur lorsque toutes les informations du colis sont disponibles.
- **E_{ACN}, E_{ACR} (événement)** : Un colis destiné à la zone nationale (N) ou régionale (R) est signalé. Cela correspond au passage d'un colis sous le portillon d'un des tapis roulants d'acheminement.
- **E_{AC} (événement)** : Un ensemble de colis est prêt. Le SC doit aller chercher les informations les concernant et remplir leur bordereau d'expédition avec les factures correspondantes. En réalité, si le nombre de colis maximum n'est pas atteint durant une certaine période (variable selon l'activité de l'entreprise) les factures sont tout de même réalisée, c'est à dire que cet événement est tout de même produit.
- **E_P (événement)** : Une palette est pleine, il faut préparer les bordereaux nécessaires. La gestion locale des colis composant la palette est terminée, à partir de ce moment là, ils sont considérés comme en livraison.
- **$E_{PR,PN}$ (événement) :** Lorsqu'un service d'expédition a composé une palette, il le signale. Cette palette quitte le quai pour être chargée dans un camion.

Dès cette première étape, nous pouvons obtenir une première description en nous appuyant sur le langage de spécification. Au fur et à mesure de l'avancée dans l'analyse, nous enrichirons cette description de manière à avoir toutes les informations nécessaires (origine et destination précises). D'ores et déjà, nous connaissons le type de l'élément de coopération, son nom ainsi que son groupe d'origine.

Langage de spécification :

- **DéfinirÉvénement** E_{NC} **De Groupe** *Réception*
- **DéfinirÉvénement** E_{CNP} **De Groupe** *Réception*
- **DéfinirÉvénement** E_{CRP} **De Groupe** *Réception*
- **DéfinirÉvénement** E_{PN} **De Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{PR} **De Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{ACN} **Pour Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{ACR} **Pour Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{AC} **Pour Groupe** *Service Commercial*
- **DéfinirÉvénement** E_{P} **Pour Groupe** *Service Commercial*

Nous allons maintenant réaliser le même travail pour les messages.

2.1.2.2 Les messages

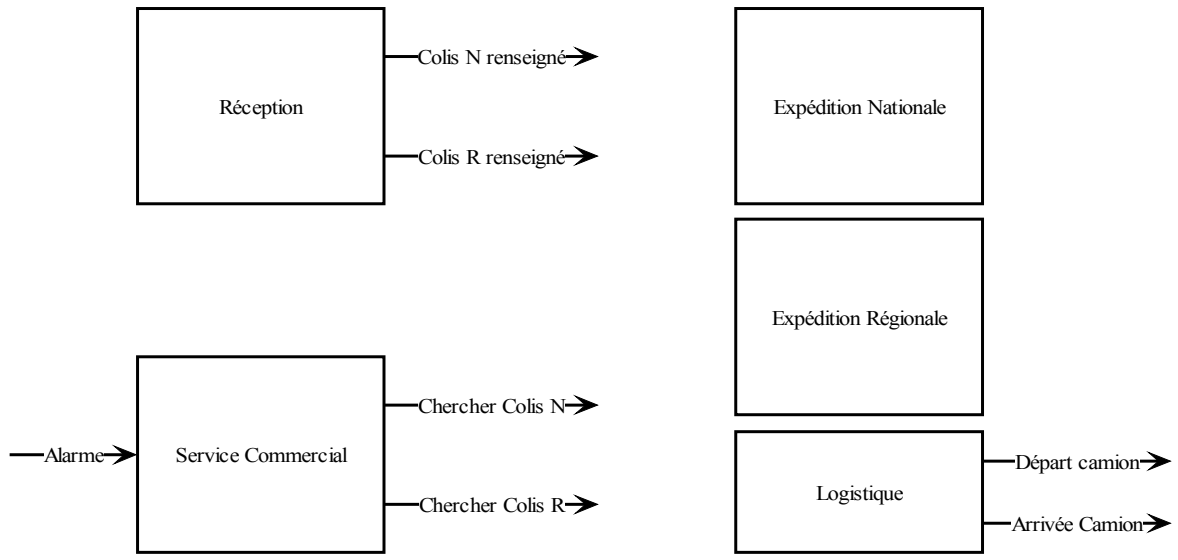


Figure 27 : Niveau Groupes - Messages nécessaires/produits

Contrairement aux événements qui n'ont aucun contenu, les messages eux en possèdent un qu'il est utile de préciser. Ainsi, pour chacun d'eux, les informations contenues dans le message doivent être décrites. Ces précisions permettront éventuellement de retrouver à l'intérieur même d'un message une information non identifiée en tant que telle auparavant mais néanmoins nécessaire.

- **Alarme (message)** : Il n'y a plus de camion disponible (ou le nombre de camions disponibles devient insuffisant). Il faut alors le notifier au SC afin d'éventuellement faire appel à des correspondants si nécessaire (le service commercial évaluera les besoins en fonction des tournées et des prévisions de retour). Ce message contient les informations sur le type et le nombre de camions devant critique.
- **Départ camion (message)**: signale le départ d'un camion pour une tournée. Ce message contient les informations sur le camion, le type de tournée, l'heure de départ ainsi que le chauffeur responsable.
- **Arrivée camion (message)**: signale le retour de tournée d'un camion. Ce message contient les mêmes informations que le message précédent (Départ camion) mis à part que l'heure de départ est remplacée par l'heure d'arrivée.
- **Colis N/R renseigné (message)** : un nouveau colis a été renseigné, ce message contient ses caractéristiques (volume, poids, destination, …).
- **Chercher Colis N/R (message)** : Un colis national ou régional est à aller chercher au cours d'une tournée. Ce message contient les informations nécessaires aux services d'expédition (où aller chercher le colis, à qui s'adresser, estimation du poids, du volume). A son retour, ce colis prendra le cycle classique d'un colis apporté à l'agence pour être acheminé vers la zone correspondante.

De manière analogue à celle utilisée pour les événements, nous pouvons d'ores et déjà réaliser une première description des messages à l'aide du langage de spécification.

- **DéfinirMessage** *Colis N Renseigné* **De Groupe** *Réception*
- **DéfinirMessage** *Colis R Renseigné* **De Groupe** *Réception*
- **DéfinirMessage** *Chercher Colis N* **De Groupe** *Service Commercial*
- **DéfinirMessage** *Chercher Colis R* **De Groupe** *Service Commercial*
- **DéfinirMessage** *Alarme* **Pour Groupe** *Service Commercial*
- **DéfinirMessage** *Départ Camion* **De Groupe** *Logistique*
- **DéfinirMessage** *Arrivée Camion* **De Groupe** *Logistique*

2.1.2.3 Les données

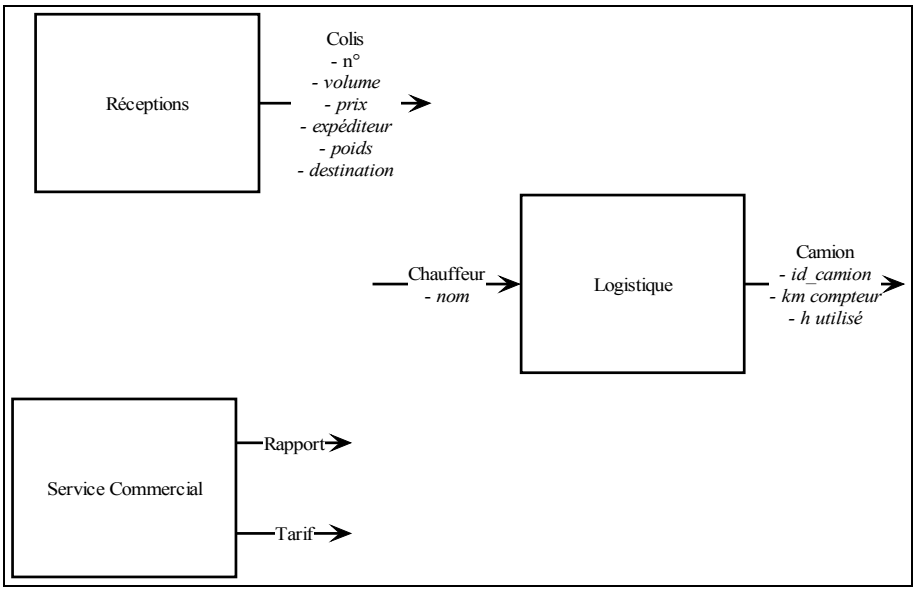


Figure 28 : Niveau Groupes - Données nécessaires/produites

- **Tarif (donnée)** : Les tarifs sont des informations produites en fonction de l'activité du moment, des offres ponctuelles et éventuellement des clients, mais aussi bien évidemment, en fonction du volume, du poids, de la destination, de la nature même du colis (les colis dangereux sont par exemples sur-taxés),…
- **Camion (donnée)** : Lorsqu'un chargement part, un chauffeur lui est associé (champs *nom* de la table Camion). Pour chaque type de tournée, il y a une durée de maximale de travail autorisée ainsi qu'un tarif horaire particulier. De plus, cette information permet un calcul du taux d'utilisation et de remplissage de ce camion (ce qui permet une meilleure gestion du parc de camions).
- **Colis (donnée)** : le groupe réception produit un ensemble d'informations (ensemble de champs) de la table *Colis* (volume, prix, expéditeur, poids, destination). Le champs *n°*, automatiquement, est nécessaire au groupe réception afin de réaliser l'étiquetage.
- **Rapport (donnée)** : Donne un rapport d'activité de la société de manière quotidienne, hebdomadaire et mensuelle.

Langage de spécification :

- **DéfinirDonnée** *nom* **Dans Table** *Chauffeur* **Pour Groupe** *Logistique*
- **DéfinirDonnée** *n°* **Dans Table** *Colis* **De Groupe** *Réception*
- **DéfinirDonnée** *volume* **Dans Table** *Colis* **De Groupe** *Réception*
- **DéfinirDonnée** *prix* **Dans Table** *Colis* **De Groupe** *Réception*
- **DéfinirDonnée** *expéditeur* **Dans Table** *Colis* **De Groupe** *Réception*
- **DéfinirDonnée** *poids* **Dans Table** *Colis* **De Groupe** *Réception*
- **DéfinirDonnée** *destination* **Dans Table** *Colis* **De Groupe** *Réception*
- **DéfinirDonnée** *id_camion* **Dans Table** *Camion* **De Groupe** *Logistique*
- **DéfinirDonnée** *km_compteur* **Dans Table** *Camion* **De Groupe** *Logistique*
- **DéfinirDonnée** *h_utilisé* **Dans Table** *Camion* **De Groupe** *Logistique*

- **DéfinirDonnée Table** *Rapport* **De Groupe** *Service Commercial*
- **DéfinirDonnée Table** *Tarif* **De Groupe** *Service Commercial*

Ce que nous avons fait dans cette première étape est un inventaire des différentes informations susceptibles d'être soit nécessaires, soit produites par au moins un des groupes de travail. Or, ces groupes sont des entités entièrement virtuelles. Les éléments de coopération mis en évidence ici ne sont actuellement pas réellement produits et/ou reçus par les groupes de travail, bien qu'ils apparaissent ici comme tels. Nous devons, pour aller plus loin, entrer dans le détail de ces groupes, c'est à dire que nous devons connaître les modules entrant dans leur composition mais aussi les informations liées à chacun de ces modules et qui permettront de faire le lien avec les éléments de coopération du groupe.

Nous connaissons maintenant l'ensemble des groupes composant l'application. Pour chacun d'eux, nous avons réalisé l'inventaire des divers éléments de coopération qu'ils sont susceptibles de produire ou bien qu'ils nécessitent. Enfin, pour chaque information mise en évidence, nous avons réalisé une description la plus précise possible de manière à en connaître la sémantique. Ce dernier point nous sera particulièrement utile ultérieurement.

2.2 Étape 2 : modules

À cours de cette étape, nous entrons dans le détail et regardons les différents composants de chaque groupe. Mais avant de pouvoir réaliser ce travail, deux choses sont nécessaires :

- Réaliser un inventaire complet des modules tant logiciels que matériels.
- Pour chacun des modules identifiés, en faire la description la plus complète possible.

En fait, ces deux étapes ne sont qu'une étude de l'existant plus précise que celle qui consistait à mettre en évidence les groupes de travail. La différence est, que cette fois-ci, nous nous basons sur des entités concrètes déjà présentes dans l'application existante - les modules - et non plus sur des entités virtuelles - les groupes de travail.

Une fois cet inventaire réalisé, nous disposons d'un état des lieu des ressources disponibles. Nous pouvons dès lors prendre les modules un à un et les placer dans les différents groupes de travail auxquels ils sont susceptibles de participer.

Nous ne nous préoccupons pas ici de la gestion de la dynamique des groupes et gérons les modules dits « *partagés* » comme des modules « classiques ». La modélisation de cette dynamique se fera à une étape ultérieure.

Tout comme nous l'avons réalisé pour les groupes de travail, nous mettons ici aussi en évidence les divers éléments entrant dans la coopération. Ainsi, pour chaque module, nous faisons l'inventaire des trois types d'éléments de coopération.

2.2.1 Identification des modules

En premier lieu, il convient de recenser les modules présents dans le système. Ils peuvent être de nature aussi bien logicielle (gestion des stocks, interfaces, supervision,…) que matérielle (cellules de production, automates programmables, …).

Lorsque l'on s'intéresse à de la ré-ingénierie d'applications, ce recensement constitue une description des moyens disponibles. C'est une contrainte forte de l'application et ce travail mérite une attention toute particulière. Une fois ce recensement terminé, nous devons organiser les modules par centre d'intérêt, c'est à dire les placer un à un dans les groupes de travail, sachant qu'un même module peut être placé dans différents groupes de travail.

Revenons par exemple sur le groupe Réception qui est constitué de 4 modules :

- BV_1 et BV_2 sont des automates chargés de peser et de mesurer les colis avec une capacité maximale de traitement de 60 colis à la minute.
- *S* est un poste de saisie sur lequel un opérateur enregistre les colis avant passage par BV_1 ou BV_2. Il permet aussi la sortie d'étiquettes (format code à barre) indiquant la référence du colis ainsi que sa destination et le client concerné.
- *PX* permet de calculer le prix en fonction du poids, du volume, de la nature du colis, du client et de la destination. C'est un poste de travail (module logiciel) qui en fonction des informations précédentes permet de rechercher dans une BD le prix correspondant.

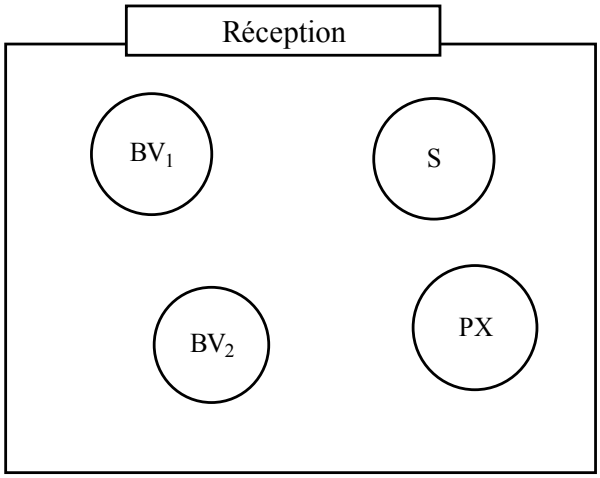


Figure 29 : Modules du groupe de travail *Réception*

2.2.2 Appartenance potentielle des modules aux groupes de travail

Les groupes ayant une structure dynamique, les modules répertoriés n'y seront pas tous présents au même moment. Nous allons décrire, pour chaque groupe de travail, les modules pouvant en faire partie à un moment donné de l'exécution de l'application. Ainsi, dans le groupe réception, il n'est nécessaire que les deux modules BV1 et BV_2 travaillent ensemble que si le nombre de colis en attente est trop important. Nous allons permettre que l'un ou l'autre de ces deux modules puisse quitter le groupe réception quand l'activité y devient trop faible. De la même façon, un module matériel particulièrement onéreux peut entrer dans la composition de plusieurs groupes et sera partagé par chacun d'eux. Ce partage peut être réalisé de manière exclusive (le module ne peut être dans plusieurs groupes au même instant), ou non exclusive. Dans ce cas, l'un des objectif de la coopération sera justement d'organiser ce partage en tenant compte des contraintes de l'application mais aussi de chacun des modules. Nous montrons ici trois modules IHM_1, IHM_2, IHM_3 pouvant être intégrés aux groupes *Expédition Nationale* et *Expédition Régionale*.

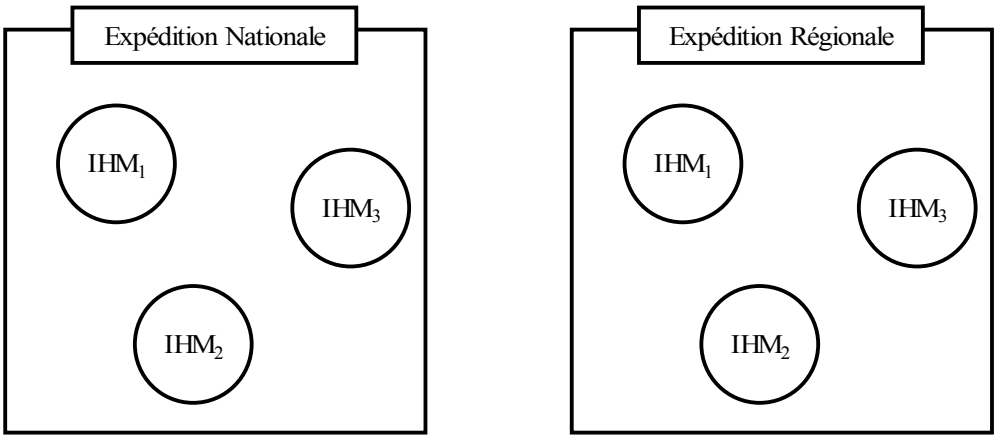


Figure 30 : Appartenance potentielle des modules aux groupes *EN* et *ER*

Ces trois modules correspondent aux postes de travail des opérateurs chargés de préparer un plan de chargement. Ils permettent d'avoir une vue en temps réel de l'état des zones de stockage. Lorsqu'un colis est sélectionné pour être envoyé, il doit être immédiatement enlevé des autres interfaces de manière à ne pas être choisi plusieurs fois (on évite ainsi aux manutentionnaires de chercher un colis qui n'est plus présent sur le quai, ces problèmes sont une source importante de perte de temps dans ce type de manipulations).

Ces interfaces pouvant aussi bien réaliser leur travail pour le groupe *Expédition Nationale* que pour le groupe *Expédition Régionale*, il faudra gérer cette dynamique (nous le verrons plus loin dans l'analyse) de manière à ce qu'elles n'affichent que les colis concernant leur groupe d'appartenance.

Afin d'illustrer nos propos, nous avons volontairement pris un exemple simple avec peu de modules. Montrer ici plus de modules n'aurait pas grand intérêt et compliquerait la compréhension. Néanmoins, nous pouvons illustrer un problème concernant la gestion de la dynamique, puisque les modules identifiés sont susceptibles de changer de groupe.

L'ensemble des colis arrivait auparavant sur toutes les interfaces, à charge pour l'opérateur de ne sélectionner que ceux concernant son affectation actuelle (nationale ou régionale). Nous avons expliqué que la mise en coopération devait pouvoir supprimer la coopération externe, c'est son but premier, mais nous avons aussi souligné qu'elle pouvait améliorer le fonctionnement de l'application. C'est le cas ici, car avec une gestion par groupes de travail, le traitement des colis sera optimisé car en assignant chaque interface à un groupe de travail, elle ne recevra que les informations correspondant à son groupe. La gestion des groupes dynamiques permettra de réaliser des changements de groupe lorsque le besoin s'en fera sentir.

Cette étape nous permet de connaître pour chaque groupe de travail l'ensemble des modules susceptibles d'y être inclus, et réciproquement, pour chaque module l'ensemble des groupes auxquels il peut appartenir.

Maintenant que nous avons le détail de la composition des groupes de travail, nous allons nous intéresser aux éléments de coopération des modules de la même manière que nous l'avons fait à l'étape 1.

2.2.3 Éléments de coopération

Nous allons prendre un à un les modules et là aussi mettre en évidence les éléments de coopération qu'ils produisent ainsi que ceux dont ils ont besoin.

2.2.3.1 Les événements

Nous illustrons la méthode en montrant, les événements que peuvent lever les modules décrits précédemment :

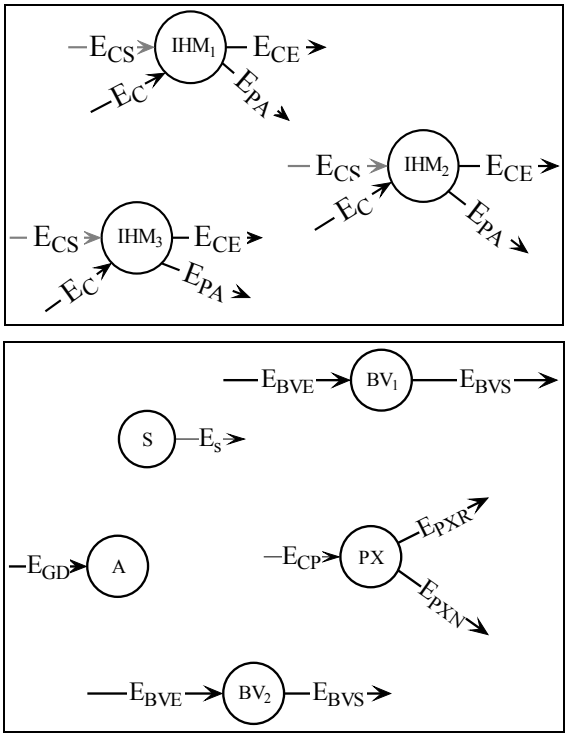


Figure 31 : Niveau Modules - Événements nécessaires/produits

Sans détailler tous les événements mis en évidence, voici ce que des entrées dans le dictionnaire pourraient être :

- ***E_{PA}* (événement)** : Cet événement est levé à chaque fois qu'une interface a terminé la composition d'une palette. La palette terminée peut être chargée. L'interface peut donc passer à une autre palette,.
- ***E_{CS}* (événement)** : Cet événement est nécessaire afin que chaque interface soit mise au courant lorsqu'un colis a été sélectionné par l'un des autres postes. Cette entrée n'existait pas, nous créons une entrée « artificielle » afin de pouvoir maintenir les interfaces à jour en n'affichant que les colis non encore sélectionnés.
- ***E_{CE}* (événement)** : (colis enlevé) Chaque fois qu'un manutentionnaire sélectionne un colis pour être chargé ce colis doit être enlevé de la liste des colis en stock de manière à ne pas être choisi plusieurs fois (ce qui obligerais les manutentionnaires à le chercher…et donc à perdre un temps trop important). L'interface génère donc un événement à chaque colis sélectionné.
- ***E_C* (événement)** : Signale l'arrivée d'un nouveau colis à prendre en compte.
- ***E_{SR} (événement)*** : Signale la fin d'une saisie sur la console
- **E_{BVE} (événement)** : Ces événements sont nécessaires afin de savoir quand un colis arrive sur le tapis de mesure.
- **E_{CP} (événement)** : Nécessaire pour prévenir qu'il y a un tarif à calculer.
- **E_{BVS} (événement)** : Cet événement est produit à la fin des mesures.

- **E_{GD} (événement)à** : Commande le basculement de l'automate d'aiguillage.
- **E_{PXN}, E_{PXR} (événements)** : Signalent qu'un colis à destination nationale ou régionale vient d'être complètement renseigné.

Remarque : Certains éléments de coopération, comme l'événement E_{CS}, sont en fait des entrées indirectes ajoutées aux modules. Leur occurrence provoquera la mise à jour de l'interface correspondante, alors que d'autres comme l'événement E_C sont des entrées directement attendues par le module. Nous faisons la différence entre entrée « d'origine » (flèches noires) et entrée « artificielle » (flèches grises) – cf. Figure 31 - permettant d'améliorer la comportement global de l'application.

Tout comme à l'étape 1 nous pouvons indiquer l'émetteur de chaque événement. L'étape précédente nous permet d'ajouter une précision supplémentaire concernant le groupe dans lequel peut-être situé le module.

- **DéfinirÉvénement** E_{PA} **De Module** IHM_1 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{PA} **De Module** IHM_2 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{PA} **De Module** IHM_3 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{CE} **De Module** IHM_1 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{CE} **De Module** IHM_2 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{CE} **De Module** IHM_3 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{CS} **Pour Module** IHM_1 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{CS} **Pour Module** IHM_2 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{CS} **Pour Module** IHM_3 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_C **Pour Module** IHM_1 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_C **Pour Module** IHM_2 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_C **Pour Module** IHM_3 **Dans Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{PA} **De Module** IHM_1 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{PA} **De Module** IHM_2 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{PA} **De Module** IHM_3 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{CE} **De Module** IHM_1 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{CE} **De Module** IHM_2 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{CE} **De Module** IHM_3 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{CS} **Pour Module** IHM_1 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{CS} **Pour Module** IHM_2 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{CS} **Pour Module** IHM_3 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_C **Pour Module** IHM_1 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_C **Pour Module** IHM_2 **Dans Groupe** *Expédition Régionale*

- **DéfinirÉvénement** E_C **Pour Module** IHM_3 **Dans Groupe** *Expédition Régionale*
- **DéfinirÉvénement** E_{SR} **De Module** S **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{GD} **Pour Module** A **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{BVE} **Pour Module** BV_1 **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{BVE} **Pour Module** BV_2 **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{BVS} **De Module** BV_1 **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{BVS} **De Module** BV_2 **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{CP} **Pour Module** PX **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{PXN} **De Module** PX **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{PXR} **De Module** PX **Dans Groupe** *Réception*

Même si nous avons fait la distinction au niveau de la représentation graphique par un changement de couleur, nous ne faisons en réalité aucune différence entre une entrée « originale » et une entrée indirecte. En effet, ce qui nous importe ici c'est de pouvoir dire à la plate-forme coopérative quels sont les éléments de coopération présents, quelles sont les entités qui les produisent et quelles sont celles qui en ont besoin.

2.2.3.2 Les messages

Nous avons mis en évidence un certain nombre d'événements. Nous allons maintenant voir ce qu'il en est des messages pour quelques uns des modules précédents :

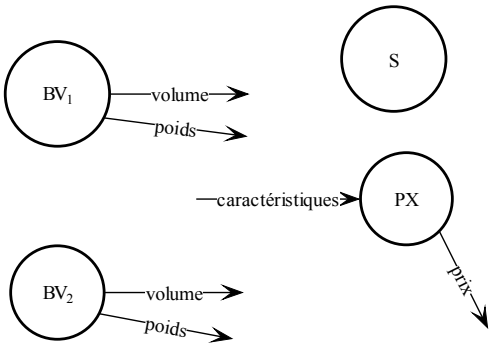


Figure 32 : Niveau Modules - Messages nécessaires/produits

- **Volume (message)** : cette information est produite par les cellules BV_1 ou BV_2 qui mesurent le volume des colis passant sur le tapis roulant d'acheminement.
- **Poids (message)** : comme précédemment, ce sont les mêmes cellules qui réalisent ce travail, et donnent le poids des colis passant sur le tapis.
- **Caractéristiques (message)** : Les caractéristiques sont formées du volume, du poids, mais aussi de la destination du colis, des délais imposés ainsi que du client (certains bénéficient de remises selon leur contrats).

- **Prix (message)** : Ce message contient le prix du colis, calculé en fonction de ses caractéristiques (poids, volume, nature, client, destination.

Langage de spécification :

- **DéfinirMessage** *Volume* **De Module** BV_1 **Dans Groupe** *Réception*
- **DéfinirMessage** *Volume* **De Module** BV_2 **Dans Groupe** *Réception*
- **DéfinirMessage** *Poids* **De Module** BV_1 **Dans Groupe** *Réception*
- **DéfinirMessage** *Poids* **De Module** BV_2 **Dans Groupe** *Réception*
- **DéfinirMessage** *Caractéristiques* **Pour Module** *PX* **Dans Groupe** *Réception*
- **DéfinirMessage** *Prix* **De Module** *PX* **Dans Groupe** *Réception*

2.2.3.3 Les données

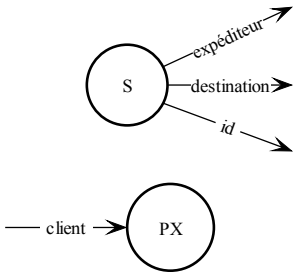


Figure 33 : Niveau Modules - Données nécessaires/produites

- **n° (donnée)** : le numéro de colis permet d'étiqueter les colis qui rentrent.
- **expéditeur (donnée)** : identifie l'expéditeur du colis.
- **destination (donnée)** : identifie le destinataire du colis.
- **client (donnée)** : identifiant du client permettant de réaliser un rabais si celui-ci correspond à un client privilégié.

Remarque : c'est lors de la saisie sur la console que les champs et destination sont insérés dans la BD.

Langage de spécification :

- **DéfinirDonnée** *expéditeur* **Dans Table** *Colis* **De Module** *S* **Dans Groupe** *Réception*
- **DéfinirDonnée** *destination* **Dans Table** *Colis* **De Module** *S* **Dans Groupe** *Réception*
- **DéfinirDonnée** *n°* **Dans Table** *Colis* **De Module** *S* **Dans Groupe** *Réception*
- **DéfinirDonnée** *client* **Dans Table** *Colis* **Pour Module** *PX* **Dans Groupe** *Réception*

Les étapes 1 et 2 permettent d'appréhender les éléments de la coopération en montrant de quoi a besoin chaque groupe et chaque module, ainsi que ce que chacun d'eux produit. Nous avons crée un dictionnaire permettant de connaître en permanence la sémantique de chacun des éléments de coopération que nous avons mis en évidence.

Nous sommes par contre, pour le moment, incapables de dire pour quels destinataires (groupe et/ou module) les événements, messages et données ont été produits, de même que nous ne savons pas toujours d'où ils proviennent. En fait, nous avons mis en évidence les éléments de la coopération mais pas encore la coopération elle même.

2.3 Étape 3 : gestion de la dynamique

À partir du moment où nous ne touchons pas à l'implémentation des modules, leur entrée ou sortie de groupe ne peut se faire à leur propre initiative. Il est nécessaire de le faire pour eux.

L'étape suivante va consister à mettre en évidence les éléments nécessaires à l'évaluation par la plate-forme coopérative des conditions d'E/S de groupe.

Nous avons mentionné lors de la présentation intuitive de la méthode qu'à chaque module, nous associions systématiquement une règle de gestion de la dynamique pour chaque groupe de travail auquel il peut appartenir. Cette règle est en réalité une règle ÉCA.

Un module pouvant appartenir à plusieurs groupes de travail sera doté d'autant de règles de gestion de la dynamique que de groupes auxquels il peut appartenir. Chacune d'elles étant chargée de vérifier les conditions d'entrée/sortie du module dans le groupe dont elle a la charge.

À l'étape 2, nous avons mis en particulier en évidence les modules $IHM_{1..3}$, pouvant être présents dans les deux groupes *Expédition Nationale* et *Régionale*.

Il va s'agir dans cette étape de définir les conditions de leur appartenance à ces groupes. Dans notre cas, la principale condition pour qu'un module de type $IHM_{1..3}$ puisse sortir d'un groupe de travail est qu'il ne soit pas actuellement en train de composer une palette. En effet, il ne paraît pas raisonnable de brutalement sortir une IHM d'un groupe de travail alors qu'un opérateur est en train d'y préparer une palette. Une des conditions à respecter sera donc l'occurrence de l'événement E_{PA}.

De plus, dans ce cas précis, une IHM ne peut appartenir qu'à un seul groupe à la fois puisque l'objectif de cette appartenance est justement d'éviter de traiter à la fois des colis à destination nationale et régionale. Il est à remarquer que ceci constitue un cas particulier et qu'en général rien n'empêche qu'un module soit simultanément dans plusieurs groupes.

Mais, on peut vouloir affiner ce changement de groupe en tenant compte de l'activité du quai c'est à dire des colis en transit dans chacune des zones (E_{ACN} et E_{ACR}) ainsi que des palettes terminées pour chaque type d'expédition (E_{PN} et E_{PR}).

La condition exprimée ici est valable pour l'ensemble des modules $IHM_{1..3}$ car ceux-ci ne sont que des instances du même module. Dans le cas contraire, c'est à dire dans le cas où les modules seraient différents, il est bien évident que chacun d'eux posséderait sa propre condition. Tout comme nous avons une condition par module, nous avons aussi des critères différents selon le groupe concerné. L'entrée/sortie d'un module d'un groupe sera aussi propre à ce dernier. Ainsi pour le groupe « *Expédition Nationale* », une autre condition à prendre en compte pour la sortie d'une interface de ce groupe est que certains clients appelés « clients privilégiés » envoient des colis en grand nombre. Lorsque l'un d'eux arrive, il y a fort à parier que de très nombreux autres arrivent aussi. Ces colis ont, de plus, un caractère urgent et doivent être expédiés au plus vite. Dans l'entreprise Laussuy, ces clients sont par exemple des fabricants de vêtements de mode qui ont des collections à faire parvenir à l'ensemble des enseignes de la marque. Ainsi, avant qu'une IHM située dans le groupe « *Expédition Nationale* » ne puisse être sortie de ce groupe pour passer dans le groupe d'expédition régionale, il est nécessaire de vérifier qu'un colis d'un client privilégié ne soit pas arrivé dans le groupe réception. C'est ici la donnée *client* de la table *Colis* qui est ici concernée.

À l'image du dictionnaire sémantique réalisé au cours des deux premières étapes, nous allons réaliser un dictionnaire des règles qui contiendra pour chacune d'elles le nom de la règle, ses constituants et son rôle.

Nous allons également maintenir un dictionnaire de gestion de la dynamique. Nous y mettrons l'ensemble des informations/conditions relatives aux entrées/sorties dans un groupe, et ce pour chacun des modules concernés. Nous aurons ainsi sous la main, l'ensemble des informations concernant la dynamique des groupes.

Puisque chacun des modules de type $IHM_{1..3}$ peut appartenir à deux groupes, nous aurons donc deux règles pour chacun (une par groupe). Ainsi, pour le module IHM_1, nous aurons :

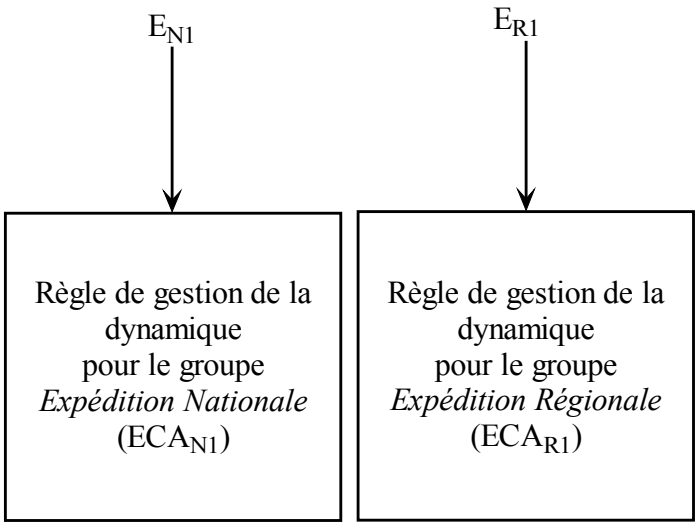


Figure 34 : Règles de gestion de la dynamique pour le module IHM_1

L'entrée correspondant à ECA_{N1} dans le dictionnaire des règles pouvant être la suivante :

- ECA_{N1} : règle de gestion de la dynamique associée au module IHM_1 pour le groupe *Expédition Nationale*. Elle est déclenchée par l'événement E_{N1}. Elle doit tenir compte :
  - de l'action actuelle de l'IHM,
  - du nombre de colis en attente,
  - du nombre de palettes expédiées,
  - de l'apparition de clients privilégiés,
  - du fait qu'au moins une IHM doit rester dans le groupe.

Étant de type ÉCA, ces règles sont sensibles à un événement survenant lorsque les conditions nécessaires à l'évaluation des conditions d'E/S du module correspondant peuvent être calculées (E_{N1} et E_{R1}). Nous ajouterons également ces deux nouveaux événements au dictionnaire sémantique que nous avons créé aux deux premières étapes. Ainsi, son contenu sera à jour et contiendra une description de l'ensemble des éléments de coopération nécessaires ou produits.

Concernant le langage de spécification, nous ajoutons :

- **DéfinirÉvénement** E_{N1} **Pour Règle** ECA_{N1}
- **DéfinirÉvénement** E_{R1} **Pour Règle** ECA_{R1}

Cet exemple permet de voir comment est gérée la dynamique des groupes. Nous voyons qu'elle se réalise sur l'occurrence d'un événement qui, lorsque la condition est vérifiée, permet de faire entrer ou sortir le module d'un groupe.

Nous retrouvons là les trois principes des règles ÉCA. L'événement É, la condition (aussi complexe que nécessaire) pouvant par exemple faire au système d'information de l'application, et enfin, l'action réalisant l'entrée ou la sortie du module.

2.4 Étape 4 : mise en correspondance des éléments de coopération et création des éléments manquants

Les étapes précédentes ont consisté à mettre en évidence les groupes de travail constituant l'application, et d'y placer les modules appropriés. Elles ont aussi permis de faire l'inventaire des différents éléments de coopération produits ainsi que ceux nécessaires aux groupes et modules. Elles ont enfin montré sur quelles occurrences d'événements les E/S des modules pouvant appartenir à des groupes dynamiques peuvent être réalisées.

Nous allons maintenant entrer au cœur de la méthode qui consiste en deux mots à mettre en place la coopération.

La circulation de l'information était auparavant réalisée par des opérateurs externes, c'est ce que l'on appelait la coopération externe. Notre objectif étant d'automatiser ce travail, nous allons mettre en évidence les liens entre les informations ; la plate-forme coopérative se chargera de leur acheminement.

Nous allons pour cela nous appuyer sur le dictionnaire sémantique et le dictionnaire de règles établi lors des trois premières étapes. C'est grâce à lui qu'on pourra s'apercevoir qu'une information identifiée en sortie correspond à une autre marquée comme une entrée. Lorsque ce cas se présentera, nous relierons graphiquement les deux éléments de coopération, et exprimerons ceci au travers du langage de spécification. Ainsi, lors de l'implémentation, la plate-forme coopérative se chargera, lorsque cette information sera produite, de la capturer afin de la communiquer aux entités destinataires (groupes et/ou modules).

2.4.1 Mise en correspondance

Nous allons agir comme au préalable en travaillant sur deux niveaux. Nous allons tout d'abord établir toutes les jonctions entre les éléments correspondant à la coopération intergroupes puis nous nous intéresserons à la coopération intra-groupe et enfin aux événements de gestion de la dynamique.

2.4.1.1 Mise en correspondance inter-groupe

2.4.1.1.1 Jonction des événements

Afin de réaliser cette opération, nous nous appuyons sur le dictionnaires et les schémas établis lors des étapes précédentes.

Dans le groupe *Réception*, nous avons l'événement E_{CNP} *« colis à destination nationale prêt»* et dans *Expédition Nationale*, l'événement E_{ACN} *« colis à destination nationale en zone de stockage »*. Bien

que ces deux événements ne portent pas le même nom, nous percevons bien qu'il correspondent à la même chose. Lorsque E_{CNP} est émis, le colis a été renseigné (destination, poids, volume, prix …). Il peut être ajouté dans la liste des colis en zone de stockage national, et donc mis à jour sur l'écran des IHM correspondants, c'est à dire sur l'ensemble des interfaces présentes dans le groupe de travail Expédition Nationale. Bien qu'ayant une sémantique et un nom différent, on se rend compte que ces deux événements constituent un point de synchronisation.

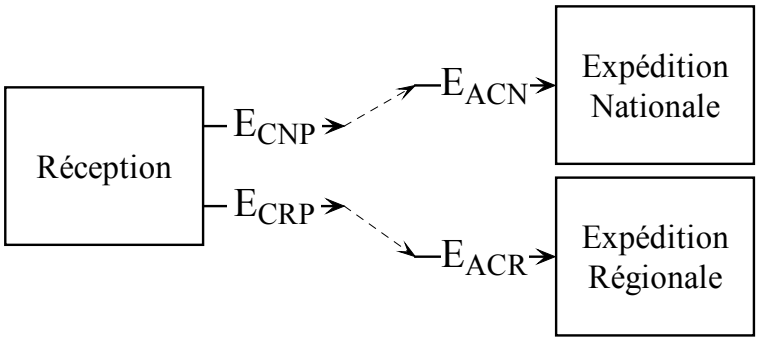


Figure 35 : Jonctions des événements intergroupes

Nous pouvons d'ores et déjà commencer à affiner les spécifications obtenues lors des étapes 1 et 2.

Nous avions les quatre entrées suivantes :

- **DéfinirÉvénement** E_{CNP} **De Groupe** *Réception*
- **DéfinirÉvénement** E_{CRP} **De Groupe** *Réception*
- **DéfinirÉvénement** E_{ACN} **Pour Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{ACR} **Pour Groupe** *Expédition Régionale*

Nous connaissons maintenant pour ces informations quelle est leur source et quels sont les destinataires. Aussi nous pouvons affiner notre spécification ainsi :

- **DéfinirÉvénement** E_{ACN} **Pour Groupe** *Expédition Nationale* **De Groupe** *Réception* **as** E_{CNP}
- **DéfinirÉvénement** E_{ACR} **Pour Groupe** *Expédition Nationale* **De Groupe** *Réception* **as** E_{CRP}

Le mot clé **as** permet la définition de synonymes. Ainsi, l'événement E_{ACN} est considéré comme produit lorsque l'événement E_{CNP} le sera. Au niveau communication, ces deux événements sont identiques. Nous pourrions harmoniser l'ensemble en ne gardant qu'un seul nom (celui de l'émetteur ou du destinataire par exemple) mais il nous semble plus intéressant de garder les deux avec chacun leur entrée dans le dictionnaire sémantique. Ils ont ainsi chacun leur description propre avec une sémantique pouvant être différente même s'ils sont reliés.

2.4.1.1.2 Jonction des messages

Contrairement aux événements, les messages ont un contenu et donc une structure. Ainsi, les messages émis par un quelconque module vont répondre à une certaine syntaxe. Si un autre module a besoin de

ce message, il ne peut le récupérer tel quel s'il n'a pas été prévu pour cela. Le problème des messages est donc qu'ils doivent pouvoir être compréhensibles par le destinataire, c'est à dire que l'entité réceptrice puisse récupérer dans le corps du message tout ou partie des informations. Or, ce n'est possible que si nous avons affaire à des machines et/ou logiciels ayant le même protocole de communication, ce qui ne sera généralement pas vrai dans une application initialement non coopérative pour laquelle les échanges d'informations n'ont été ni prévus ni harmonisés.

Ainsi, un message émis par le premier module, bien que contenant toutes les informations nécessaires au deuxième, devra être récupéré puis remis en forme (ou traduit, ou complété, …) pour devenir compréhensible par ce dernier.

Pour résoudre ce problème, deux possibilités s'offrent à nous :

- L'émetteur et/ou le récepteur sont modifiés afin de se mettre d'accord sur un protocole de communication,
- Une procédure externe est établie, récupérant l'information émise par l'émetteur, elle la transforme afin qu'elle soit comprise par le destinataire puis la lui communique.

Si l'on veut pouvoir utiliser la première solution, il nous faut modifier l'implémentation des modules, ce qui ne peut se faire dans notre cas.

Il nous reste la seconde solution : la plate-forme coopérative détectera l'émission du message, l'interceptera puis le modifiera de manière à le rendre compréhensible par le ou les destinataires. Elle le transmet ensuite aux entités concernées. Ce travail sera réalisé par une règle détectrice dont l'entrée est le message initial et l'information composée résultante le message mis en forme.

2.4.1.1.3 Jonction des données

Même si le cas ne se présente pas dans notre exemple, nous devons nous intéresser à la jonction entre informations persistantes. Nous rencontrerons ces jonctions au niveau intra-groupe à l'étape suivante.

2.4.1.2 Mise en correspondance intra-groupe

Cette démarche, ayant déjà été décrite en détail précédemment, nous nous contenterons ici de mettre en évidence les points clés de cette étape de l'analyse.

Nous allons tout d'abord établir toutes les jonctions entre les éléments correspondant à la coopération inter-groupes. Afin de réaliser cette opération, nous nous appuyons une fois de plus sur le dictionnaire établi lors des deux étapes précédentes.

Par exemple, l'événement E_{ACN} correspondant à l'arrivée d'un colis national doit parvenir à l'ensemble des modules de ce groupe, d'où la jonction entre E_{ACN} et E_C de chaque IHM du groupe *Expédition Nationale*.

Afin de mieux illustrer les jonctions d'événements entre modules, nous allons appliquer la méthode sur le groupe de travail Expédition Nationale :

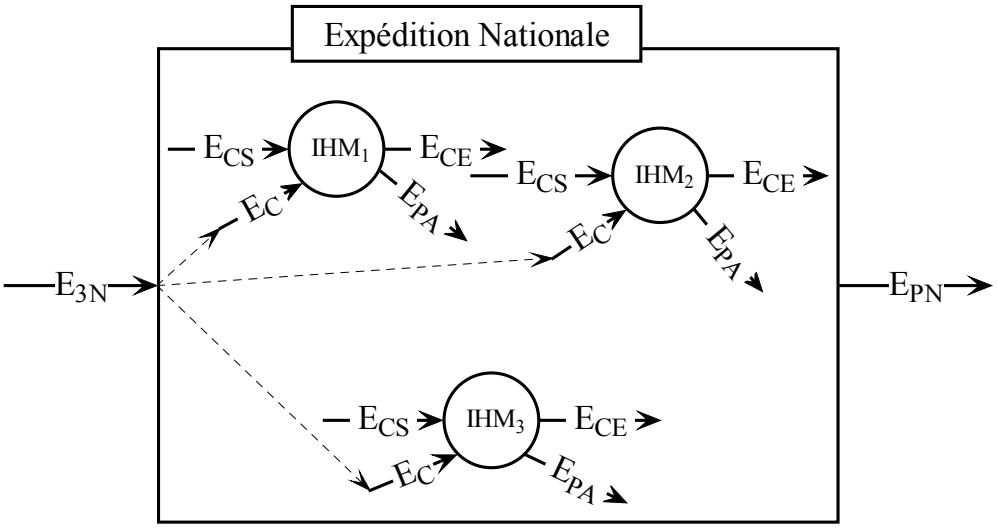


Figure 36 : jonction d'événements intra-groupe

Lorsqu'un nouveau colis à destination nationale est arrivé (E_{ACN}), il doit être signalé à l'ensemble des interfaces présentes dans le groupe d'expédition nationale.

Concernant les données *client* et *expédition*, lorsque la console de saisie enregistre un colis, elle y associe un expéditeur. Cette information sera utilisée par le module PX pour éventuellement réajuster le prix si ce client fait partie des client privilégiés.

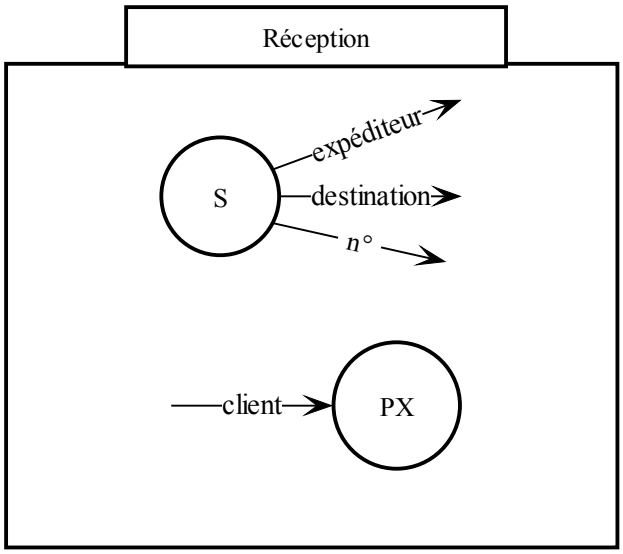


Figure 37 : jonction de données intra-groupe

2.4.1.3 Mise en correspondance des événements de gestion de la dynamique

Lors de l'étape 3 nous avons défini des règles ÉCA ayant en charge la gestion de la dynamique des groupe. Chaque module se voit ainsi affecté d'une règle par groupe auquel il peut appartenir. Ces règles doivent être déclenchées par un événement significatif de l'évolution de l'application. Ainsi, dans le groupe Réception, les deux modules BV_1 et BV_2 chargés de la mesure des colis peuvent se voir l'un ou l'autre sortis de ce groupe lorsque la quantité de colis à traiter ne justifie pas l'utilisation des deux à la fois. L'événement déclenchant l'évaluation des conditions d'entrée ou de sortie de ces modules de groupe est bien évidemment celui qui permet de mesurer l'activité de ce groupe en terme de colis à traiter, c'est à dire l'événement E_{NC} qui signale chaque arrivée de colis.

Nous pouvons donc relier directement cet événement aux règles associées aux modules BV_1 et BV_2.

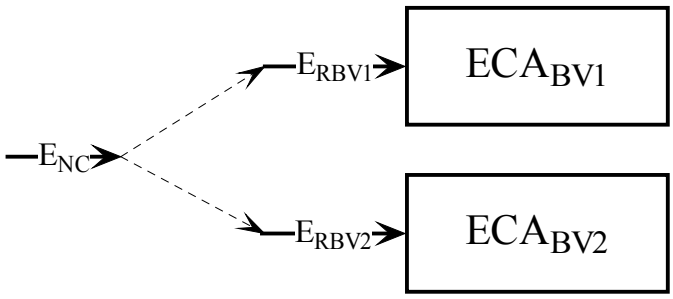


Figure 38 : gestion des événements aux règles de gestion de la dynamique

Langage de spécification :

- **DéfinirÉvénement** E_C **Pour Module** IHM_1 **Dans Groupe** *Expédition Nationale* **as** E_{ACN}

- **DéfinirÉvénement** E_C **Pour Module** IHM_2 **Dans Groupe** *Expédition Nationale* **as** E_{ACN}
- **DéfinirÉvénement** E_C **Pour Module** IHM_3 **Dans Groupe** *Expédition Nationale* **as** E_{ACN}
- **DéfinirDonnée** *id* **Dans Table** *Colis* **Pour Module** PX **Dans Groupe** *Réception* **as** *client*
- **DéfinirÉvénement** E_{GBV1} **Pour Règle** ECA_{BV1} **as** E_{CN}
- **DéfinirÉvénement** E_{GBV2} **Pour Règle** ECA_{BV2} **as** E_{CN}

Tous les éléments de coopération ne peuvent malheureusement pas être obtenus ainsi par de simples liaisons directes. En effet, dès qu'une information doit être obtenue par composition de plusieurs autres (réalisé auparavant par la coopération externe), cette méthode n'est plus suffisante.

Il est nécessaire d'accomplir un traitement supplémentaire : réaliser la synthèse d'information. Nous entrons là dans le domaine de la production d'éléments de coopération, ce que va s'attacher à décrire la prochaine partie.

2.4.2 Création des éléments de coopération manquants

A l'étape précédente, nous avons relié entre eux les éléments de coopération lorsque ce lien était direct, c'est à dire lorsqu'il qu'il ne nécessitait aucune autre modification qu'une simple mise en forme, traduction, ...

Mais les jonctions directes d'informations ont leurs limites. Cette technique sous entend que chaque information nécessaire est forcément produite de manière complète par une entité quelconque de l'application. Or, ce cas a peu de chances de se produire dans la réalité. Si certaines informations, généralement simples, vont en effet correspondre parfaitement, il est par contre peu probable qu'une information nécessaire à partir du moment où elle commence à être un peu complexe (composée, synthèse de plusieurs autres) trouve son pendant produit par une quelconque entité du système.

Les éléments de coopération directement disponibles ne permettent pas de réagir à des situations complexes constituées, par exemple, d'une succession ordonnée d'événements. Ceci permettrait d'obtenir un contenu sémantique plus riche ou bien permettrait de ne réagir que dans certains cas bien précis augmentant ainsi l'efficacité. Il est donc nécessaire de prévoir un mécanisme permettant de détecter un schéma d'occurrence bien précis. Ce schéma peut être une suite d'événements distincts, le contenu d'un message, une valeur d'une donnée du SI, une répétition dans un laps de temps, etc. Bref, toute situation envisageable à partir du moment où elle est algorithmiquement exprimable.

Nous allons rencontrer des situations où une information produite n'est pas réutilisable telle quelle, parce que trop canonique par exemple, ou bien que des informations nécessaires ne sont jamais produites. Nous pouvons illustrer ces cas par E_{CS} *et* E_{AC} – (cf. Figure 26 et Figure 31) par exemple qui

sont des éléments de coopération pour lesquels il n'est pas possible de trouver une correspondance directe avec un élément identifié.

En nous référant au dictionnaire, nous allons tenter de trouver comment ces éléments de coopération peuvent être obtenus par composition d'autres.

Il peut arriver que malgré la composition des informations, il soit impossible de produire une information nécessaire. De deux choses l'une :

1. Lors des étapes précédentes, nous avons oublié de mettre en évidence un élément de coopération,
2. Il est impossible de produire cette information soit parce qu'elle n'est pas productible de manière algorithmique, soit parce qu'il est impossible de l'identifier au sein de l'application.

Dans le premier cas, il est tout à fait envisageable de revenir sur les étapes précédentes afin d'ajouter le ou les éléments de coopération qui ont été omis. Il suffira de modifier les graphiques, d'ajouter les entrées correspondantes dans le dictionnaire sémantique et de mettre à jour le langage de spécification.

Le deuxième cas est plus délicat et ne pourra pas toujours être résolu du fait de la spécificité de notre approche, la ré-ingénierie d'applications ne permet pas d'avoir la souplesse que l'on a lorsque l'on crée de toutes pièces une application ou que l'on peut intervenir de manière importante sur le contenu (le code) de celle-ci.

Toutefois, il se peut que la non faisabilité vienne aussi du fait que l'information nécessaire identifiée par l'analyse soit une information théorique trop précise et irréalisable en pratique. Néanmoins, une reprise partielle de l'analyse devrait pouvoir nous permettre d'identifier (ou de construire) une nouvelle information peut être moins précise mais suffisante.

Si nous pouvons être aussi affirmatifs sur le fait que nous arriverons à produire les informations manquantes c'est qu'il faut garder à l'esprit que nous intervenons sur des applications déjà existantes et donc que les informations identifiées sont déjà produites d'une façon d'une autre par une quelconque entité de l'application.

Nous avons proposé, lors de travaux antérieurs [Tawbi 96], un mécanisme dérivé des règles ÉCA appelé *règles détectrices*. A la différence des règles ÉCA, l'action des règles détectrices consiste à lever des événements (appelé *événements composés*) après composition. Contrairement aux autres

mécanismes développés dans le domaine des BDA où les compositions ne peuvent être réalisées qu'à partir d'opérateurs booléens (et de contraintes temporelles pour certains [Dayal 88]), le principe de règle détectrice programmable permet de réaliser des compositions aussi complexes que nécessaire, plus en rapport avec une activité coopérative où les paramètres sont nombreux et de plusieurs ordres (temporels, valeurs, seuils, état de l'activité …). Lorsque les événements détectés valident l'algorithme, la règle lève un événement composé qui peut éventuellement être diffusé sur le réseau [Roose 97a] [Roose 97b].

Nous proposons maintenant d'étendre ce principe et le généraliser à la composition des autres éléments de coopération. Ceci nous permet de constituer des événements et des messages composés ainsi que des données synthétiques. Ainsi, nous disposerons de règles détectrices qui, au lieu de réaliser leur composition à partir d'événements seulement, pourront recevoir et produire tout type d'élément de coopération.

Nous allons maintenant nous intéresser à la constitution des éléments de coopération manquants permettant de faire coopérer les groupes et modules. Une informations manquante est aisément identifiable graphiquement. Elle correspond à une entrée d'un groupe, d'un module ou d'une règle pour laquelle aucune jonction n'a pu être réalisée à l'étape précédente. Cela veut donc dire que l'information nécessaire n'est pas encore présente. Il faut donc la fabriquer. C'est à ce niveau qu'interviennent les règles détectrices. Elle vont, à partir des informations générées par des groupes, des modules, ou même à partir d'autres règles détectrices, fabriquer les informations encore manquantes.

2.4.2.1 Création des événements composés

Le schéma correspondant va nous permettre de représenter graphiquement les éléments traités par l'algorithme de composition contenu dans chacune des règles détectrices ainsi que l'information qu'elles produisent. Afin de toujours avoir une référence sur ce que l'on fait, à partir de quoi et comment on produit un élément composé, nous tiendrons à jour un dictionnaire des règles détectrices.

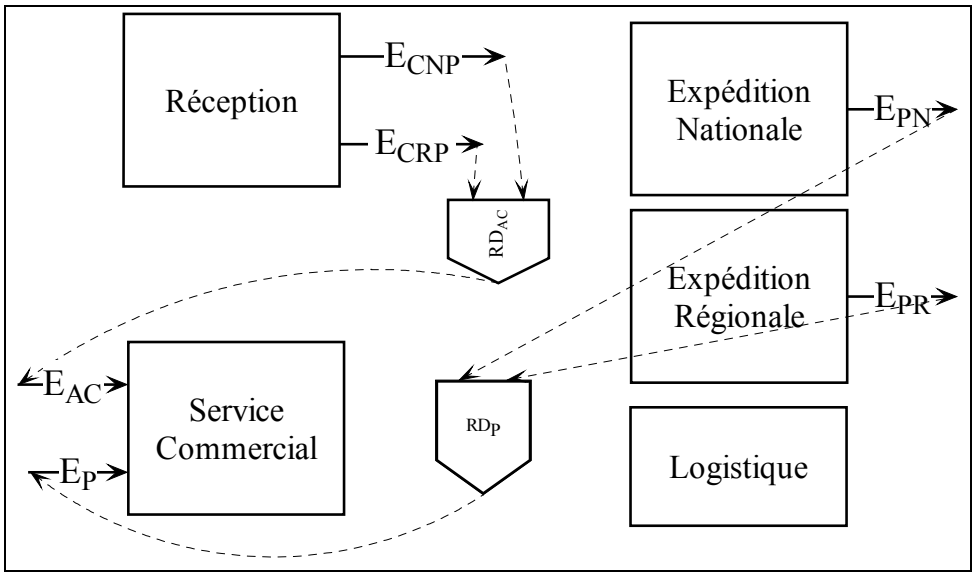


Figure 39 : Utilisation d'une RD pour générer les éléments manquants

Exemple d'entrée dans le dictionnaire des règles :

- **RD_{AC}** : Cette règle permet de détecter la présence d'un lot de colis renseignés, qu'ils soient à destination nationale ou régionale.

L'exemple suivant va illustrer la création d'événements composés. Prenons par exemple, E_{AC} (en entrée du groupe *Service Commercial*) et qui n'a pas encore été produit (pas relié). Nous allons examiner la manière de le créer. Cet événement sert à déclencher l'impression des factures pour les clients. Afin d'optimiser le traitement de ces factures, cet événement ne sera pas produit à chaque nouveau colis mais lorsqu'un nombre suffisant de colis à destination nationale ou régionale est à facturer (ou lorsque le délai entre deux facturations est trop important). Cet événement pourra donc être constitué par une règle détectrice RD_{AC} à partir des événements E_{CNP} et E_{CRP} produits par le groupe *Réception*.

- **RD_P** : Cette règle permet de signaler chaque palette prête à partir. Elle réalise en fait le *ou* des entités E_{PN} et E_{PR}.

En nous appuyant sur le dictionnaire sémantique nous avons vu que l'événement E_P arrive lorsqu'une palette a été remplie. Afin de ne pas surcharger le service commercial, nous pouvons améliorer ce traitement. Toujours à l'aide du dictionnaire sémantique, nous pouvons savoir que l'événement E_P peut être fabriqué en réalisant une composition entre les événements E_{PN} et E_{PR}. Ainsi, la règle RD_P recevra tous les événements E_{PN} et E_{PR}. Dans le cas présent, la composition consistera à réaliser un *ou*.

Bien entendu, chacune de ces règles fera l'objet d'une entrée dans le dictionnaire des règles. On y trouvera son nom, les éléments de coopération nécessaires (avec leur type) ainsi que leur rôle dans

l'évaluation de la condition. Enfin, nous décrirons aussi l'information produite par cette règle de manière à ce qu'elle puisse être reprise plus tard par d'autres entités de l'application.

On a donc les définitions suivantes :

- **DéfinirÉvénement** E_{CNP} **De Groupe** *Réception* **Pour Règle** RD_{AC}
- **DéfinirÉvénement** E_{CRP} **De Groupe** *Réception* **Pour Règle** RD_{AC}
- **DéfinirÉvénement** E_{PN} **De Groupe** *Expédition Nationale* **Pour Règle** RD_P
- **DéfinirÉvénement** E_{PR} **De Groupe** *Expédition Régionale* **Pour Règle** RD_P
- **DéfinirÉvénement** E_{AC} **De Règle** RD_{AC} **Pour Groupe** *Service Commercial*
- **DéfinirÉvénement** *Ep* **De Règle** RD_P **Pour Groupe** *Service Commercial*

Il nous reste à répondre à une question : comment gérer les colis qu'il faut aller chercher chez le client ? Bien que ces colis ne soient pas passés par le groupe réception et n'aient pas subi le traitement habituel, il faut prévoir leur place dans les camions en partance. Pour ce faire, nous pouvons nous contenter de signaler leur présence future au groupe Expédition Nationale ou Expédition Régionale qui pourront ainsi réserver leur place dans les camions. En ré-évaluant les jonctions et les créations déjà réalisées, nous nous rendons compte qu'il suffit que leur présence soit composée avec celles des colis présents sur le quai. Ainsi, nous allons modifier la Figure 35 correspondant à des jonctions directes d'événements inter-groupes afin que les événements E_{ACN} et E_{ACR} reflètent réellement les colis à gérer.

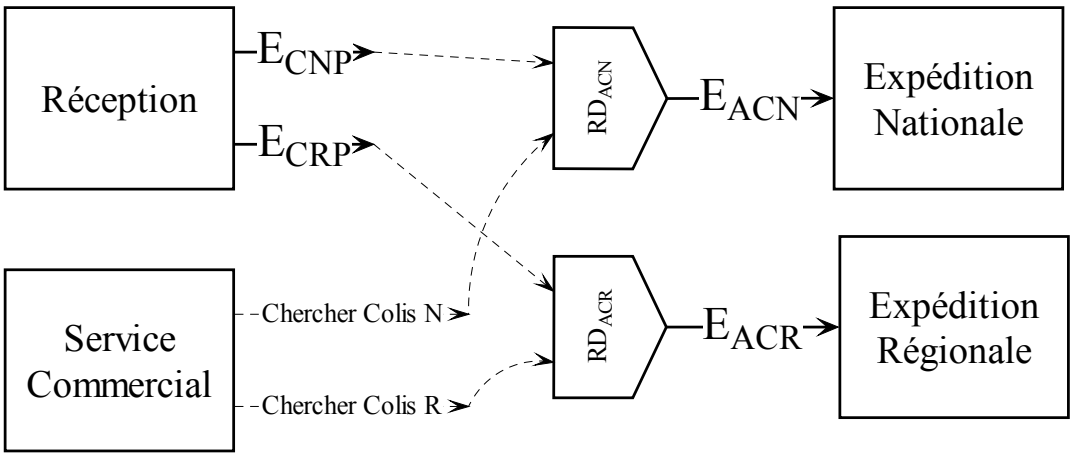


Figure 40 : Mise à jour des jonctions

Nous allons donc modifier les spécifications suivantes…

- **DéfinirÉvénement** E_{ACN} **Pour Groupe** *Expédition Nationale* **De Groupe** *Réception* **as** E_{CNP}
- **DéfinirÉvénement** E_{ACR} **Pour Groupe** *Expédition Régionale* **De Groupe** *Réception* **as** E_{CRN}

… afin qu'elles prennent en compte cette considération.

- **DéfinirÉvénement** E_{CNP} **Pour Règle** RD_{ACN}
- **DéfinirÉvénement** E_{CRP} **Pour Règle** RD_{ACR}
- **DéfinirMessage** *Chercher Colis N* **Pour Règle** RD_{ACN}
- **DéfinirMessage** *Chercher Colis R* **Pour Règle** RD_{ACR}
- **DéfinirÉvénement** E_{ACN} **De Règle** RD_{ACN} **Pour Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_{ACR} **De Règle** RD_{ACR} **Pour Groupe** *Expédition Régionale*

Nous voyons qu'il est ainsi possible de revenir sur la méthode afin d'affiner certains résultats obtenus. Nous voyons aussi que, grâce à la composition, il est possible de générer des informations possédant une sémantique plus élevé, plus à mêmes d'exprimer la coopération dans l'application. Cet exemple illustre également la composition d'éléments de coopération de différents types. En effet, même si pour des raisons de clarté nous n'avons pas fait apparaître sur les schémas l'ensemble des éléments de coopération réellement présents, ce cas est très souvent rencontré. Ainsi il n'est pas rare de rencontrer une composition de messages qui, selon les cas, peut donner lieu à l'introduction d'une synthèse dans un SGBD ou bien provoquer la levée d'un événement. N'oublions pas que nous évoluons dans un environnement de modules à la fois logiciels et matériels et que nous nous situons dans la ré-ingénierie.

Voici par exemple le résultat que l'on obtient sur la groupe Réception :

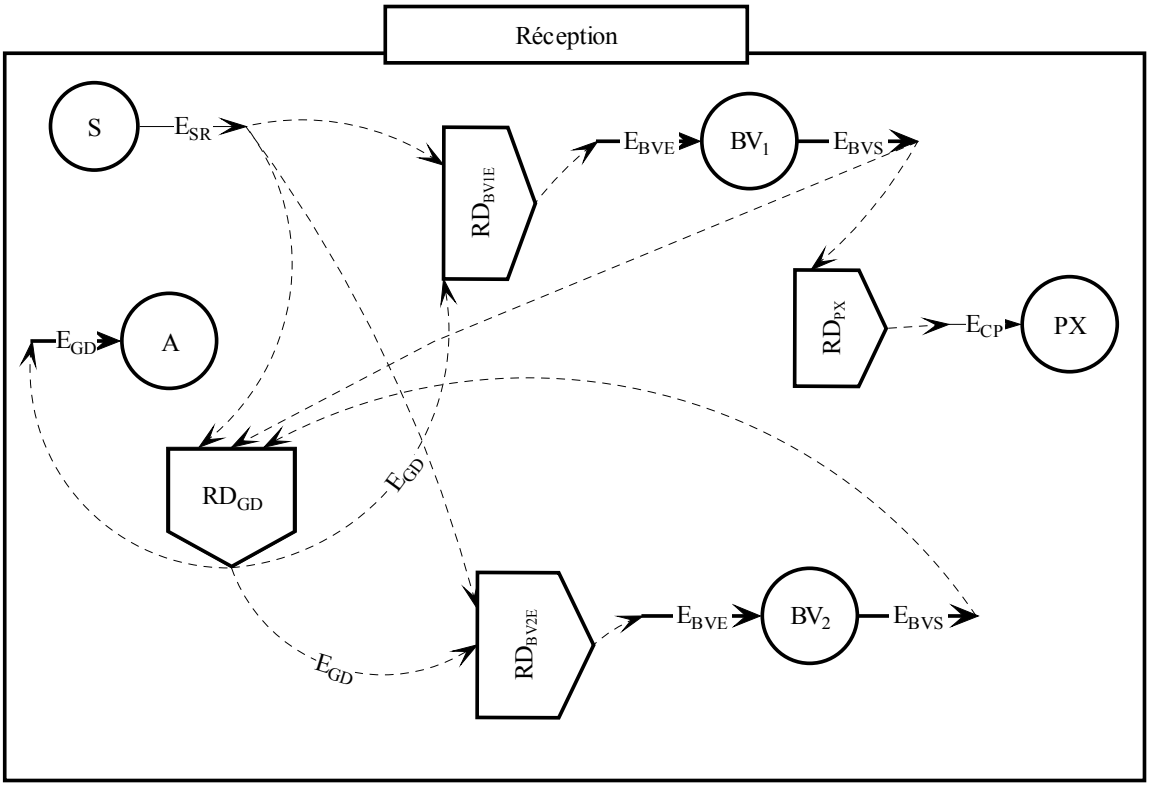


Figure 41 : Événements manquants créés dans le groupe Réception

- **RD_{GD} :** Cette règle permet de piloter l'aiguillage. Afin de savoir où en sont les cellules BV_1 et BV_2, elle a besoin de recevoir les événement respectif signalant la fin de leur traitement. Lorsqu'une saisie est réalisée (E_{SR}), selon l'état de BV_1 et BV_2 (obtenu par E_{BVS}) elle provoquera un changement d'aiguillage.
- **RD_{BV1E}, RD_{BV2E}:** Ces deux règles sont en relation directe avec RD_{GD}. Elles permettent de savoir lorsqu'un changements d'aiguillage et donc de savoir laquelle des cellules BV_1 ou BV_2 va recevoir le colis.
- **RD_{PX} :** Enfin, RD_{PX} est mis au courant lorsqu'une $BV_{1,2}$ a terminé son travail, permettant de signaler à PX qu'il y a un prix à calculer.

Langage de spécifications :

- **DéfinirÉvénement** E_{SR} **De Module** *S* **Dans Groupe** *Réception* **Pour Règle** RD_{BV2E}
- **DéfinirÉvénement** E_{SR} **De Module** *S* **Dans Groupe** *Réception* **Pour Règle** RD_{BV1E}
- **DéfinirÉvénement** E_{SR} **De Module** *S* **Dans Groupe** *Réception* **Pour Règle** RD_{GD}
- **DéfinirÉvénement** E_{BV1S} **De Module** BV_1 **Dans Groupe** *Réception* **Pour Règle** RD_{GD}
- **DéfinirÉvénement** E_{BVS} **De Module** BV_1 **Dans Groupe** *Réception* **Pour Règle** RD_{PX}
- **DéfinirÉvénement** E_{BVS} **De Module** BV_2 **Dans Groupe** *Réception* **Pour Règle** RD_{GD}
- **DéfinirÉvénement** E_{BVS} **De Module** BV_2 **Dans Groupe** *Réception* **Pour Règle** RD_{PX}
- **DéfinirÉvénement** E_{BVE} **De Règle** RD_{BV2E} **Pour Module** BV_2 **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{BVE} **De Règle** RD_{BV1E} **Pour Module** BV_1 **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{GD} **De Règle** RD_{GD} **Pour Module** *A* **Dans Groupe** *Réception*

- **DéfinirÉvénement** E_{CP} **De Règle** RD_{PX} **Pour Module** PX **Dans Groupe** *Réception*
- **DéfinirÉvénement** E_{GD} **De Règle** RD_{GD} **Pour Règle** RD_{BV2E}
- **DéfinirÉvénement** E_{GD} **De Règle** RD_{GD} **Pour Règle** RD_{BV1E}

La création des événements composés s'intéresse à la fois aux événements identifiés lors des deux premières étapes (identification des éléments de coopération inter et intra-groupes), mais aussi lors de la troisième étape dont le but est l'identification des informations nécessaires pour gérer les groupes dynamiques. Puisque ce sont des règles ÉCA qui sont chargées de ce travail, elles nécessiteront forcément un événement en entrée, événement qu'il sera éventuellement nécessaire de fabriquer.

Pour les messages et les données composés, seules les deux premières étapes seront nécessaires, il est bien entendu impossible d'avoir de nouveaux messages et données identifiés à l'étape 3.

À l'étape 3, nous avons mentionné l'utilisation d'une règle ($ÉCA_{N1}$) afin de gérer les entrées/sorties du groupe *Expédition Nationale* du module IHM_1. Or, jusqu'à présent, cet événement n'a nullement été identifié. Il faut donc le créer. Là encore, nous nous appuyons sur la puissance des règles détectrices. Il faut pour cela mettre en évidence les conditions de cette dynamique. La condition est basée sur l'état de la BD (état des stocks) ainsi que sur celui de l'application (une interface ne peut pas sortir d'un groupe si elle est la dernière à en faire partie). Lorsque $ÉCA_{N1}$ recevra l'événement E_{N1} elle évaluera s'il est opportun de faire entrer/sortir le module IHM_1 du ou des groupes de travail dans lesquels il est.

Un module du type de l'IHM_1 ne peut sortir d'un groupe qu'une fois la palette en cours de remplissage terminée. En effet, il ne paraît pas raisonnable de brutalement sortir une IHM d'un groupe de travail alors qu'un opérateur est en train de préparer une palette. Cette sortie ne pourra donc se faire qu'après l'occurrence de l'événement E_{PA} « *chargement de palette terminée* » de ce module. Il est aussi nécessaire de prendre en compte les colis en attente dans chacune des zones (E_{ACN} et E_{ACR}) ainsi que les palettes terminées pour chaque type d'expédition (E_{PN} et E_{PR}). Ainsi, en fonction du nombre de palettes complétées et de colis renseignés, il sera possible de choisir s'il faut qu'une des interfaces change de groupe. Enfin, nous avons parlé de clients privilégiés. Ainsi, en plus des informations précédentes, il est nécessaire de récupérer la donnée *expéditeur* produite lors de chaque passage par la console de saisie. Enfin, il appartiendra à la règle $ÉCA_{N1}$ de respecter la contrainte qui est que chaque groupe doit obligatoirement comporter au moins une interface.

Nous n'allons pas le montrer ici, mais il revient à inclure les lignes précédentes en ajoutant le nom de la règle concernée (RD_{G1}) ainsi que l'élément de coopération produit E_{N1} (cf schéma suivant).

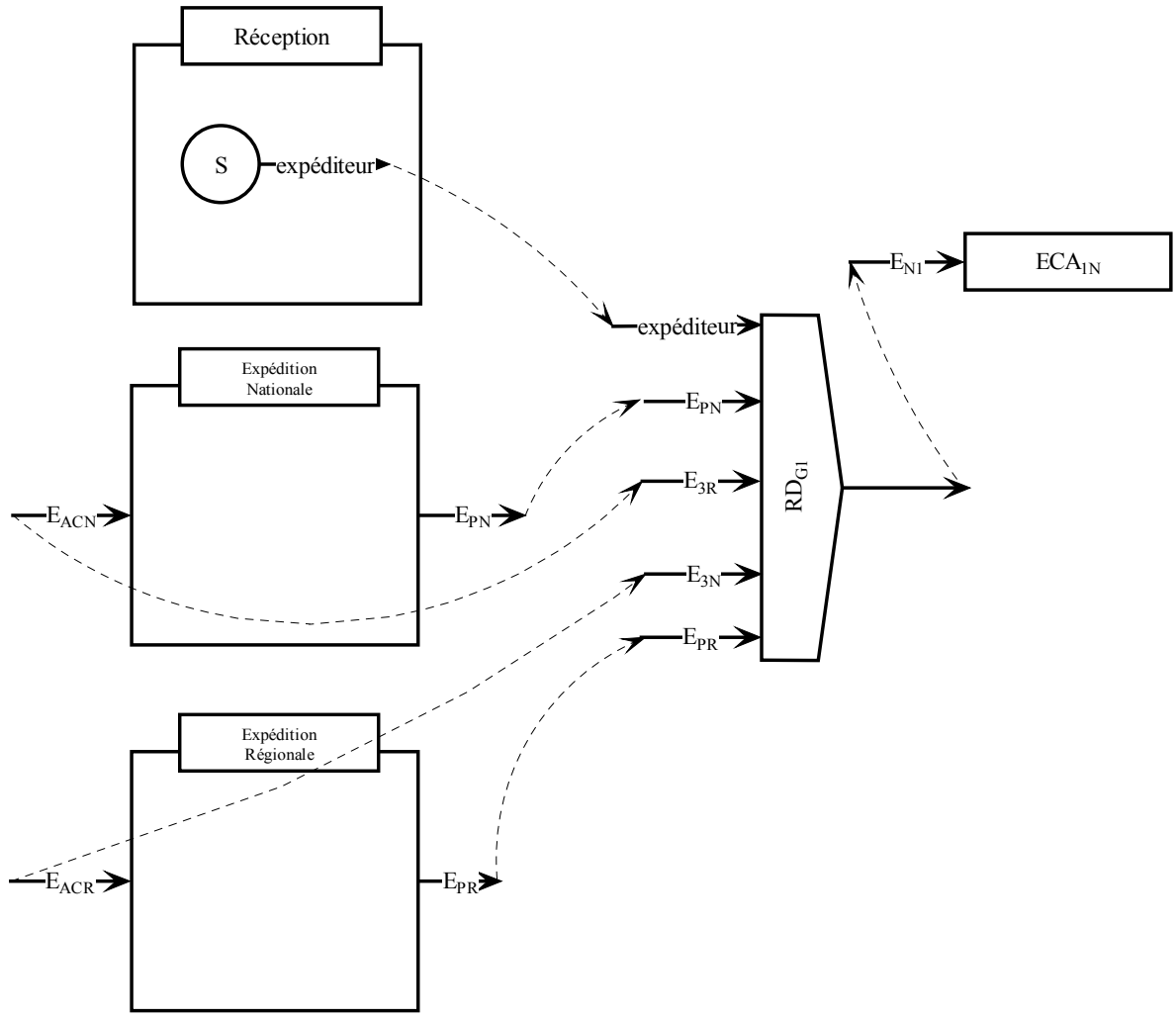


Figure 42 : Création d'un événement de gestion de la dynamique

Nous pouvons voir que les spécifications continuent de se préciser, et pour certaine d'entre elles comme E_{PA}, l'ensemble des informations nécessaires à la dérivation sont déjà présente cet événement est maintenant pleinement qualifié.

- **DéfinirÉvénement** E_{ACN} **Pour Règle** R_{G1}
- **DéfinirÉvénement** E_{ACR} **Pour Règle** R_{G1}
- **DéfinirÉvénement** E_{PN} **De Groupe** *Expédition Nationale* **Pour Règle** RD_{G1}
- **DéfinirÉvénement** E_{PR} **De Groupe** *Expédition Régionale* **Pour Règle** RD_{G1}
- **DéfinirÉvénement** E_{N1} **De Règle** RD_{G1} **Pour Règle** $ÉCA_{N1}$
- **DéfinirDonnée** *expéditeur* **De Table** *Client* **Pour Règle** RD_{G1}

2.4.2.2 Création des messages composés

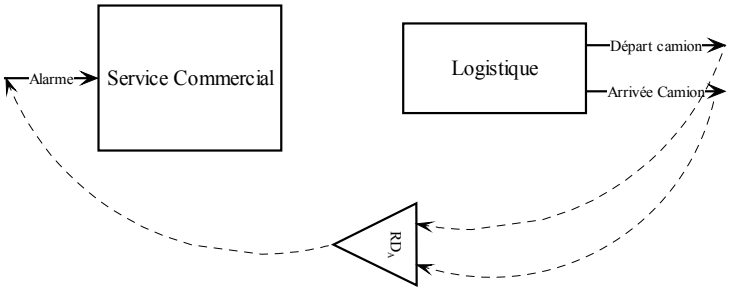


Figure 43 : Création des messages intergroupes manquants

Dictionnaire des règles détectrices

- **RD_A :** En fonction des départs et des arrivées des camions ainsi que de leur type de tournée, elle notifiera une alarme lorsque le nombre de camions disponible deviendra trop faible.

Dans le schéma ci-dessus, nous n'avons pas illustré le cas d'une règle détectrice dont le rôle serait uniquement de traduire un message pour le rendre compréhensible par un destinataire, néanmoins cela reste une possibilité qu'il faut garder à l'esprit.

Langage de spécification

- **DéfinirMessage** *Alarme* **De Règle** RD_A **Pour Groupe** *Service Commercial*
- **DéfinirMessage** *Départ Camion* **De Groupe** *Logistique* **Pour Règle** RD_A
- **DéfinirMessage** *Arrivée Camion* **De Groupe** *Logistique* **Pour Règle** RD_A

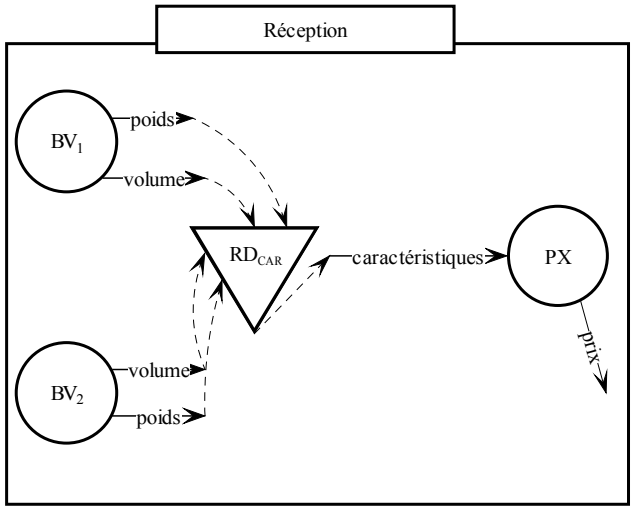


Figure 44 : Messages crées pour le groupe Réception

- **R_{CAR}** : Afin de calculer le prix d'un colis, le module PX récupère les informations (poids, volume) dans le message identifié ici par *caractéristiques*. $BV_{1,2}$ produisent ces informations mais dans deux messages séparés. Il faut donc réaliser la synthèse des deux messages et ce sans mélanger le poids mesuré par BV_1 avec la volume donné par BV_2. C'est ici qu'intervient la règle RD_{CAR} qui réalise cette synthèse et se charge de la mettre au format reconnu par PX.

Langage de spécification

- **DéfinirMessage** *volume* **De Module** *BV_1* **Dans Groupe** *Réception* **Pour Règle** *RD_{CAR}*
- **DéfinirMessage** *volume* **De Module** *BV_2* **Dans Groupe** *Réception* **Pour Règle** *RD_{CAR}*
- **DéfinirMessage** *poids* **De Module** *BV_1* **Dans Groupe** *Réception* **Pour Règle** *RD_{CAR}*
- **DéfinirMessage** *poids* **De Module** *BV_2* **Dans Groupe** *Réception* **Pour Règle** *RD_{CAR}*
- **DéfinirMessage** *caractéristique* **De Règle** *RD_{CAR}* **Pour Module** *PX* **Dans Groupe** *Réception*
- **DéfinirMessage** *prix* **De Module** *PX*

2.4.2.3 Création des données composées

Enfin, nous avions expliqué à la première étape la nécessité de produire l'ensemble des données nécessaires à chaque nouveau colis renseigné, mais nous n'avions jusqu'à présent, pas montré comment ces données étaient créées.

Avant la mise en place de la coopération, un opérateur saisissait toutes ces informations. L'étude que nous venons de mener permet de constater que ces informations pouvaient être collectées auprès des différents modules.

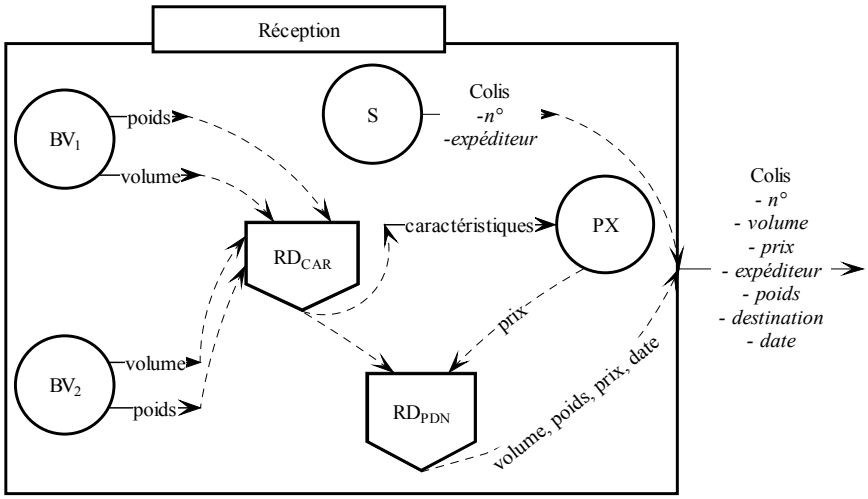


Figure 45 : Création des données manquantes

- **RD_{PDN}** : Cette règle (*RD_{PDN}*) permet de récupérer ces informations et mettre à jour la base de données. Pour ce faire, elle complète les données produites par *S* (*S* est le poste sur lequel, à chaque colis apporté par un

client, un opérateur saisit les informations nécessaires, c'est à dire la *destination* et l'*expéditeur*) à l'aide des messages crées par RD_{CAR} et émis par *PX*. RD_{PDN} ajoute donc les données *volume, prix, poids et date*, récupérées dans les messages *caractéristiques* et *prix*.

Langage de spécification :

- **DéfinirDonnée** *volume* **Dans Table** *Colis* **De Règle** RD_{PDN}
- **DéfinirDonnée** *prix* **Dans Table** *Colis* **De Règle** RD_{PDN}
- **DéfinirDonnée** *expéditeur* **Dans Table** *Colis* **De Module** *S* **Dans Groupe** *Réception*
- **DéfinirDonnée** *poids* **Dans Table** *Colis* **De Règle** RD_{PDN}
- **DéfinirDonnée** *destination* **Dans Table** *Colis* **De Module** *S* **Dans Groupe** *Réception*
- **DéfinirDonnée** *n°* **Dans Table** *Colis* **De Module** *S* **Dans Groupe** *Réception*
- **DéfinirDonnée** *date* **Dans Table** *Colis* **De Règle** RD_{PDN}
- **DéfinirMessage** *volume* **De Module** BV_1 **Dans Groupe** *Réception* **Pour Règle** RD_{CAR}
- **DéfinirMessage** *volume* **De Module** BV_2 **Dans Groupe** *Réception* **Pour Règle** RD_{CAR}
- **DéfinirMessage** *poids* **De Module** BV_1 **Dans Groupe** *Réception* **Pour Règle** RD_{CAR}
- **DéfinirMessage** *poids* **De Module** BV_2 **Dans Groupe** *Réception* **Pour Règle** RD_{CAR}
- **DéfinirMessage** *caracteristiques* **De Règle** RD_{CAR} **Pour Module** *PX* **Dans Groupe** *Réception*
- **DéfinirMessage** *caracteristiques* **De Règle** RD_{CAR} **Pour Règle** RD_{PDN}
- **DéfinirMessage** *prix* **De Module** *PX* **Dans Groupe** *Réception* **Pour Règle** RD_{PDN}

Remarque : Nous aurions tout aussi bien pu aussi utiliser la composition des événements si nécessaire.

Jusque là, nous avons crée les groupes de travail, étudié leur composition et mis en œuvre la coopération entre les modules composant les groupes mais aussi entre les groupes eux mêmes. Nous connaissons maintenant l'ensemble des éléments de coopération présents dans l'application. Nous connaissons les entités émettrices et réceptrices (Groupes, Modules, Règles).

Avant de dériver les règles de la plate-forme, nous devons nous assurer de la cohérence en vérifiant que chaque information peut bien être produite.

2.5 Étape 5 : Vérification formelle

Cette partie plus formelle va nous permettre de nous assurer de la cohérence de notre approche. Cette étape consiste à vérifier que tous les éléments de coopération identifiés dans l'application peuvent être effectivement produits. Dans les cas où il viendrait à en manquer, nous les identifierons et nous proposerons une solution afin de résoudre ce problème.

Nous partons pour cela de l'hypothèse simple qui est qu'à la base de toute production d'information doit se trouver un module. Une fois que l'élément est produit, il permettra la génération d'autres éléments pour arriver à l'information ultime (élément terminal), celle à partir de laquelle aucune autre n'est produite.

A partir du langage de spécification, il est possible de créer un graphe de dépendance entre les différents éléments de coopération. Reprenons une partie des résultats obtenus précédemment pour illustrer cela :

- **DéfinirÉvénement** E_{SR} **De Module** *S* **Dans Groupe** *Réception* **Pour Règle** RD_{GD}
- **DéfinirÉvénement** E_{BVS} **De Module** BV_1 **Dans Groupe** *Réception* **Pour Règle** RD_{GD}
- **DéfinirÉvénement** E_{GD} **De Règle** RD_{GD} **Pour Règle** $RDBV_{2E}$

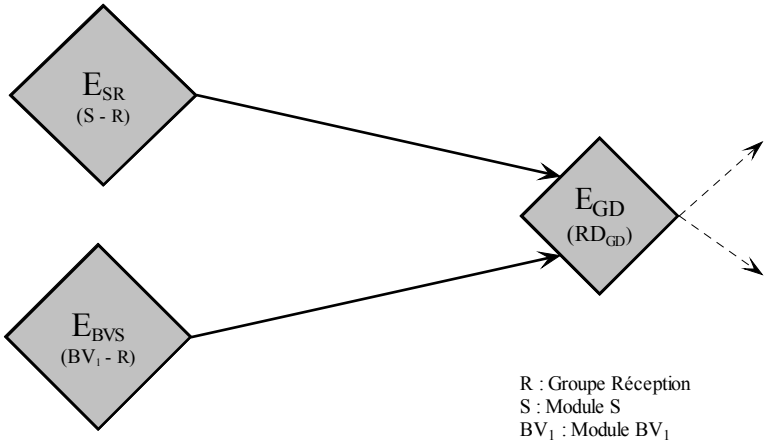


Figure 46 : Création du graphe de dépendance

On indique par ce graphe que la production de l'événement E_{GD} dépend de l'occurrence des événements E_{SR} du module S du groupe Réception et E_{BVS} du module BV_1 de ce même groupe.

2.5.1 Identification et propriétés des relations

Ce travail permettra de hiérarchiser les éléments de coopération. Nous touchons là directement au domaine des ensembles ordonnés permettant la comparaison de deux ou plusieurs objets.

Dans les paragraphes qui suivent, nous allons définir les relations d'ordre établies entre les éléments de coopération identifiés au cours de l'analyse.

En utilisant le langage de spécification, nous avons une relation de succession (que nous appellerons η_1) qui s'établit. Si l'on reprend l'exemple ci-dessus, nous voyons que les deux événements E_{SR} *et* E_{BVS}

entrent dans la composition d'une règle produisant l'événement E_{GD}. Ainsi, les deux éléments de coopération E_{SR} *et* E_{BVS} sont précédents à E_{GD}, et inversement, E_{GD} est leur successeur.

À partir du langage de spécification, nous obtenons le graphe orienté G (E, A, ņ1) des éléments de coopération. De manière plus formelle cela s'exprime ainsi :

- *E est un ensemble fini des sommets de G – l'ensemble des éléments de coopération identifiés.*
- *A est un ensemble d'arêtes – les liens de dépendances entre éléments de coopération.*
- *ņ1 une relation sur E de graphe G permettant à chaque arête d'associer une destination.*

En appliquant cette définition à notre cas de figure, nous avons :

L'ensemble des sommets E est composé des éléments de coopération (événements, messages, données) mis en évidence au cours de l'analyse. Ils sont reliés entre eux par des arêtes appartenant à A. Chaque arête représente un lien entre deux informations. Par exemple, si pour avoir E_j, il faut que E_i soit produit, E_i et E_j seront reliés.

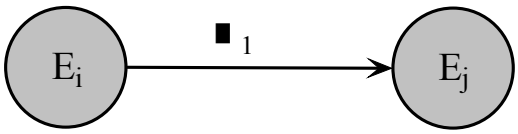


Figure 47 : Liens entre éléments de coopération

Mais, avant d'aller plus loin, il est nécessaire d'étudier cette relation ņ1 afin de connaître ses caractéristiques. Nous allons auparavant faire un rappel de quelques propriétés sur les relations binaires [Arnold 1993] :

Une relation est dite :

- Réflexive, si

  Še gE, e ņ e

- Irréflexive, si

  Še, e' gE, e ņ e' Q e ã: e'

- Symétrique, si

  Še, e' gE, e ņ e' Q e' ņ e

- Antisymétrique, si

$\forall e, e' \in E,\ e \eta e' \text{ et } e' \eta e \Rightarrow e' = e$

- Asymétrique, si

  $\forall e, e' \in E,\ e \eta e' \Rightarrow \neg(e' \eta e)$

- Transitive, si

  $\forall e, e', e'' \in E,\quad e \eta e' \text{ et } e' \eta e'' \Rightarrow e \eta e''$

La relation η_1

η_1, est une relation appelée relation de précédence qui pour deux sommets (éléments de coopération) donnés (E_i et E_j), permet de dire si l'un des deux succède à l'autre, c'est-à-dire, si une information est produite à partir de l'autre.

Exemple, si E_j ne peut se produire que lorsque E_i l'est, alors : $E_j \eta_1 E_i$, c'est à dire que *i* suit *j*. Illustrons ceci par un exemple plus conséquent : soient E_1,E_2,E_3, E_4, E_5, E_6, E_7 six sommets tels que $E_1, E_2, E_3, E_4, E_5, E_6, E_7 \in E$ et reliés par le graphe suivant :

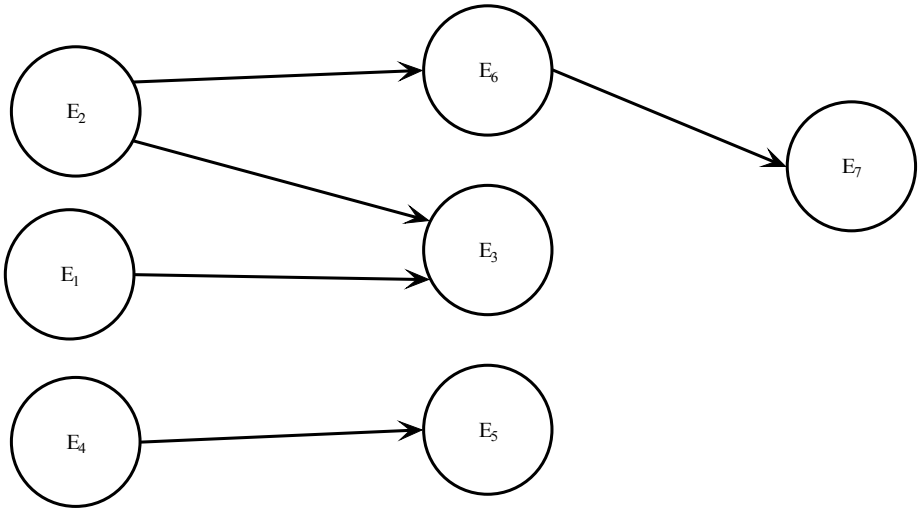


Figure 48: Relation η_1

Associée à la relation η_1, nous avons une fonction appelée *suiv( )*, qui à chaque sommet $s \in E$, associe l'ensemble des sommets *s'* suivant tels que : $s \eta_1 s'$.

Ainsi en considérant E_2, la fonction *suiv ( )* associe E_6 et E_3 à ce sommet, puisqu'ils ne peuvent être produits que lorsque E_2 le sera.

$$suiv(E_2) = \{E_3, E_6\}$$
$$suiv(E_1) = \{E_3\}$$

Nous définissons une opération plus générale qui pour un ensemble de sommets donnés fournit l'ensemble de leurs successeurs. Cette fonction est appelée *Suiv ( )*.

Définition :

Soit S · E, Suiv (S) = { x ϵ E / ∃y ϵ S et y ɳ$_1$x}

On remarque que :

$$\text{Suiv (S)} = \bigcup_{y \in S} \text{suiv(y)}$$

Nous allons maintenant tenter de caractériser la relation ɳ$_1$ (*suivant*) afin de connaître ses propriétés :

- Un élément s ϵ E ne peut être en relation avec lui même (s ɳ$_1$ s). En effet, cela voudrait dire que le suivant de cet élément est lui même. La relation ɳ$_1$ est a-réflexive (x ɳ$_1$ x est faux).
- Si l'on a s, s' ϵ E, s ɳ$_1$ s', alors nous n'avons pas s' ɳ$_1$ s. En effet, un élément suivant d'un autre ne peut avoir ce même autre élément comme suivant. La relation ɳ$_1$ est a-symétrique (si xɳ$_1$y, alors on a pas yɳ$_1$x).
- Considérons trois éléments s, s', s'' ϵ E. Si s est un suivant de s' alors nous avons s ɳ$_1$ s'. De même si s' est un suivant de s'', alors s' ɳ$_1$ s''. Si un élément est le suivant d'un second qui est lui même le suivant d'un troisième, alors le premier est aussi suivant du troisième. Nous avons ici la notion de transitivité. La relation ɳ$_1$ est transitive (x ɳ$_1$ y et y ɳ$_1$ z alors x ɳ$_1$ z).

Définition

Une relation ɳ est un ordre large si elle est réflexive, antisymétrique (ou asymétrique) et transitive. ɳ est une relation d'ordre strict si au lieu d'être réflexive, elle est irréflexive.

Si l'on considère la relation ɳ$_1$ (*suivant*), elle est a-symétrique, a-réflexive et transitive. C'est une relation d'**ordre strict**, nous la noterons > .

Nous pouvons aussi définir une relation (inverse de *suiv*) appelée *préc* qui à chaque sommet *s* ϵ *E* associe l'ensemble des sommets *s'* précédents de *s* tel que : *s'* η *s*

En reprenant l'exemple de la Figure 48, nous avons la fonction *précédent* qui associe E_2 et E_1 à E_3 puisque ce sommet ne peut être produit qu'après les occurrences de E_1 et E_2. La fonction *préc( )* retourne l'ensemble des précédents d'un élément. De manière analogue à la fonction *Suiv ( )*, nous définissons la fonction *Préc ( )* qui permet de mettre en évidence l'ensemble des éléments de coopération nécessaires à la production d'autres.

Sue le graphe de la Figure 48, on a :

- suiv(e_1) = {e_3 } ; suiv(e_2) = {e_3,e_6 } ; Suiv(e_5) = ?
- préc(e_6) = {e_2} ; préc (e_3) = {e_2,e_1} ; préc (e_4) = ?

Pour deux éléments s et s' ϵ E, il n'est pas certain que l'on ait s η_1 s' ou que s' η_1 s. En reprenant la Figure 48, considérons les éléments e_1 et e_4, nous voyons qu'ils n'ont aucune relation entre eux.

Ainsi, ∀e, e' ϵ E, nous n'avons pas forcément e η_1 e' ou e' η_1 e ainsi η_1 ne définit pas un ordre total mais partiel.

Si nous avons défini la relation η_1 c'est avec l'objectif de vérifier à la fin de l'analyse que tous les éléments de coopération identifiés peuvent effectivement être produits, c'est à dire qu'à partir d'un ensemble d'éléments de coopération de base, il est possible de produire tous les autres.

Du fait que nous ayons un ensemble ordonné (partiellement ordonné plus précisément), nous pouvons définir des parties de cet ensemble. Chacune d'elles interviendra dans la justification de la méthode.

2.5.2 Complétude des éléments de coopération

Ces relations vont nous permettre de vérifier la complétude des éléments de coopération définis lors des étapes précédentes. En effet, le problème qui se pose est celui de savoir si notre étude nous permet effectivement de disposer de tous les éléments de coopération nécessaires.

Pour cela, nous allons établir le graphe complet de dépendances pour tous les éléments de coopération de l'application. Toutefois, nous avons volontairement conservé des synonymes reliés par la relation « as » afin de ne pas perdre les noms et la sémantique associée.

La première étape consiste à établir le graphe de dépendances défini par la relation η_1 (suivant) et à s'assurer qu'il ne comporte pas de circuit. On pourra pour cela utiliser un algorithme classique de parcours de graphe en profondeur ou des méthodes plus évoluées comme l'algorithme de Warshall [Sedgewick 91]

Le nombre d'éléments de coopération identifiés étant fini, nous savons maintenant que nous travaillons sur un ensemble partiellement ordonné fini.

Une propriété des ensemble finis ordonnés est qu'ils admettent au moins un élément maximal et au moins un élément minimal.

Définition :

Soit E' une partie d'un ensemble ordonné (E, >). Un élément x de E' est dit maximal dans E' si $\nexists$ y $\in$ E', y $\neq$ x / y > x. Respectivement, un élément x de E' est dit minimal dans E' si $\nexists$ y $\in$ E', y $\neq$ x / x > y

On notera :

- Max(E') l'ensemble des maximaux de E'
- Min(E') l'ensemble des minimaux de E'.

Dans notre cadre d'utilisation, ces deux ensembles ont une signification particulière. Ainsi, l'ensemble des éléments de coopération Min(E) représente les informations « minimales », c'est à dire celles qui ne sont produites à partir d'aucune autre information de E (c'est l'ensemble des sommets de degré entrant égal à 0) et donc qui ne sont pas produites par une quelconque composition. Ce sont les informations de base à partir desquelles il doit être possible de produire toutes les autres. Nous devons nous assurer qu'il s'agit d'éléments de coopération directement disponibles dans l'application. Pour cela nous pouvons identifier ces éléments à partir du langage de spécification grâce à la clause « **De Module** » :

DéfinirÉlémentdeCooperation EC **De Module** M Dans Groupe G …

Voici l'algorithme d'identification des éléments de coopération disponibles :

```
Pour chaque déclaration d'élément de coopération {
        S'il possède la clause De Module M {
          Récupérer son nom
          Ajouter son nom à la liste des éléments minimaux
        }
}
```

algorithme 1

Il est ainsi aisé à partir du langage de spécification d'isoler l'ensemble des éléments de coopération produits à partir de modules puis de vérifier qu'il est égal à l'ensemble des éléments minimaux - Min(E).

Ceci permet de s'assurer que les racines du graphe sont bien des éléments de coopération disponibles. Toutefois, cela ne garantit pas que toues les autres éléments de coopération puissent être crées. Pour cela, nous allons nous intéresser aux éléments maximaux.

Max(E) représente l'ensemble des informations dites « terminales », c'est à dire celles qui ne servent à la production d'aucune autre (c'est l'ensemble des sommets de degré sortant égal à 0).

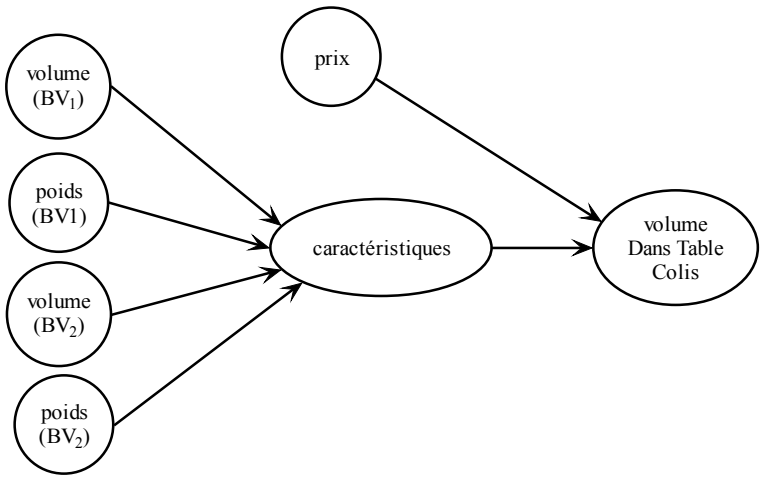
Au niveau du langage de spécification, ces éléments correspondent à des information produites identifiées (*DéfinirÉlémentdeCoopération*) au sein de l'application, mais qui ne sont pas reprises, c'est à dire qui n'ont pas de destination et donc n'apparaissent pas dans la clause « *Pour* » (*Pour Règle, Pour Groupe, Pour Module*). Voici l'algorithme d'identification des éléments maximaux :

```
Pour chaque déclaration d'élément de coopération {
        Pour chaque déclaration d'élément de coopération {
          Est-ce que cet élément est repris après le mot clé Pour ?
          Si oui, on passe à l'élément suivant,
          Si non et qu'on a parcouru toutes la liste des éléments de coopération, ajouter
          l'élément à la liste des éléments de coopération terminaux
        }
}
```

algorithme 2

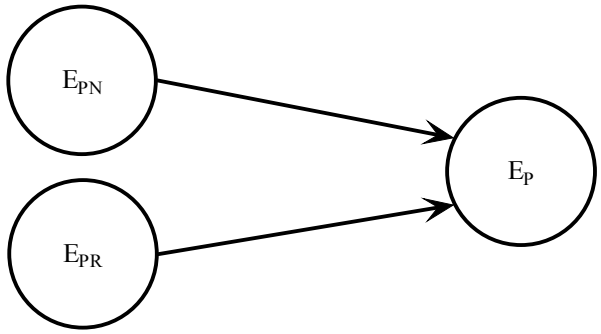
Notre graphe n'étant pas connexe, il nous devons identifier l'ensemble de ses composantes connexes. Puis, en partant des sommets maximums de chacune des composantes connexes du graphe et en établissant la fermeture transitive correspondante, nous devrions passer par l'ensemble des sommets du graphe (vérification par coloration des sommets par exemple). La fermeture transitive se terminera forcément sur un élément minimum. En effet, dans le cas où un sommet n'est pas parcouru par la fermeture transitive, cela signifierait que ce sommet n'est pas lui même un maximum (on démarre la fermeture transitive à partir de tous les sommets maximums) et qu'il appartient à un chemin dont le but n'est pas non plus un maximum, ce qui n'est pas possible sur un graphe sans circuits.

Nous allons partir du langage de spécification pour réaliser les vérifications nécessaires. Prenons l'exemple de la donnée *volume* de la table Colis. Nous avons le graphe suivant (cf. Figure 45) :



La donnée *volume* de la table *Colis* est un élément maximum tandis que les messages prix, poids, volume sont issus des modules PX, BV_1 et BV_2 du groupe réception et sont donc minimaux. Cette donnée pourra donc bien être constitués par les règles RD_{CAR} et RD_{PDN}.

Si nous nous intéressons maintenant à l'événement E_P en entrée du groupe Service Commercial (cf. Figure 39) nous avons le graphe suivant :



Toutefois, ni E_{PN} ni E_{PR} ne sont minimals (ils sont issus de groupes et non de modules). Ces informations étant nécessaires (elles entrent en plus dans la composition de plusieurs règles), il va falloir revenir sur certaines étapes de l'analyse afin de mettre en évidence leur moyen de production. Deux possibilités s'offrent à nous : l'information est produite dans l'application (identifiée aux étapes 1 ou 2) et n'avait pas été reliée correctement (étape 4) ou bien, l'information n'existe pas, il faut alors la produire (étape 4).

Si nous prenons l'événement E_{PR} correspondant à la constitution d'une palette, nous allons nous rendre compte que manifestement il est nécessaire de le créer puisqu'aucun module ni dans le groupe d'expédition régionale, ni d'ailleurs dans les autres groupes, ne produit d'événement compatible. En fait, les trois modules d'interface peuvent le produire lorsqu'elles sont dans ce groupe. Il va donc falloir établir une règle permettant de produire E_{PR} en fonction des trois événements E_{PA} *(Palette Composée)* des trois interfaces de saisie $IHM_{1..3}$. Cette règle est relativement simple puisqu'elle se contente de recevoir chacun des E_{PA} des interfaces lorsqu'elles seront dans le groupe puis de produire l'événement E_{PR} par un opérateur *ou*. Nous ne détaillerons pas l'intérieur de la règle ce qui aurait peu d'intérêt.

La vérification de la complétude nous a donc permis de mettre en évidence un oubli dans les étapes précédentes. Il faudra donc revenir en arrière pour traiter le composition de E_{PR} et affiner le langage de spécification en remplaçant la ligne suivante :

- **DéfinirÉvénement** E_{PR} **De Groupe** *Expédition Régionale* **Pour Règle** RD_{G1}

Par :

- **DéfinirÉvénement** E_{PA} **De Module** IHM_1 **Dans Groupe** *Expédition Régionale* **Pour Règle** RD_{PR}
- **DéfinirÉvénement** E_{PA} **De Module** IHM_2 **Dans Groupe** *Expédition Régionale* **Pour Règle** RD_{PR}
- **DéfinirÉvénement** E_{PA} **De Module** IHM_3 **Dans Groupe** *Expédition Régionale* **Pour Règle** RD_{PR}
- **DéfinirÉvénement** E_{PR} **De Règle** RD_{PR} **Pour Règle** RD_{G1}

On remarquera que la création de l'événement E_{PN} se fait de la même manière, la seule différence est que la règle de composition ne prend en compte les événements E_{PA} que lorsque l'IHM correspondante est dans le groupe Expédition Nationale.

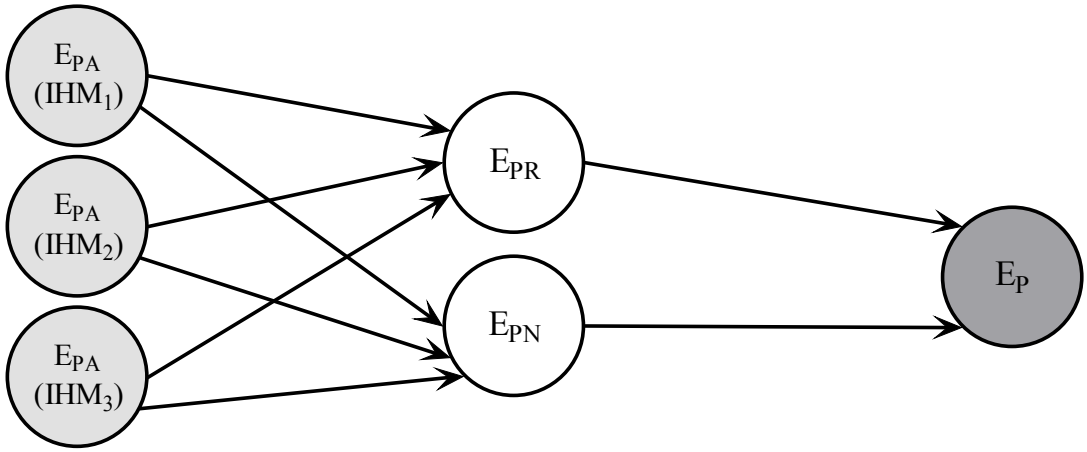


Figure 49 : Développement du graphe

Après ces modifications réalisées, nous avons une informations E_P (terminale) réalisable puisque tout ses prédécesseurs sont des informations minimales.

Ce point de la méthode revêt une importance capitale. Nous avons mentionné à plusieurs reprises la possibilité de revenir sur les étapes de la méthode comme nous venons de le faire ici. Bien qu'il soit évident que revenir en arrière n'est pas la solution idéale, la meilleure étant de ne rien oublier (!), nous pensons toutefois qu'il est nécessaire de proposer ce moyen de correction et plus important encore, nous pensons que pouvoir mettre en évidence les erreurs possibles constitue un avantage indéniable.

2.5.3 Résumé de la vérification formelle

Le premier travail à réaliser est la construction, après suppression des synonymes, du graphe de dépendances entre les différents éléments de coopération. Ce graphe est orienté à l'aide de la relation ɳ1 permettant pour chaque information d'identifier son ou ses successeurs. Il est construit en partant

du langage de spécification qui nous permet d'identifier les différents éléments de coopération de l'application.

Une fois cette construction faite, un certain nombre de contrôles sont nécessaires :

- Le premier travail à réaliser est de vérifier que le graphe obtenu ne comporte pas de cycle. Cette vérification est nécessaire puisque dans le cas où cela se produirait, cela signifierait qu'une information à besoin d'elle même afin d'être produite. La relation η_1 ne serait plus une relation d'ordre strict. Dans le cas où un cycle serait découvert, il faut s'intéresser aux sommets concernés et remonter dans l'analyse afin de faire les modifications nécessaires. Afin de réaliser cette vérification, nous pourrons utiliser un parcours en profondeur (ou en largeur) du graphe.
- Nous vérifions sur le graphe que les minimaux (l'origine de chacune des chaînes) correspondent bien à la liste des éléments de coopération issus de modules établie à l'aide du langage de spécification (cf. algorithme 1 précédent). Dans le cas contraire, nous devons revenir les différentes étapes de l'analyse afin, soit d'ajouter une information non encore identifiée (étape 1 et 2), soit de remplacer une information non productible par une autre (les étapes concernées dépendent de la nature de l'information. Ce peut être aux étapes 1,2, 3 ou 4). Il est à noter que lorsque l'on revient sur une étape, il est nécessaire de répercuter les modifications sur l'ensemble des autres étapes
- Nous constituons ensuite la liste des éléments maximals grâce à l'algorithme 2 puis vérifions par fermeture transitive que nous passons bien par chaque sommet du graphe.

Erreurs possibles :

Si un cycle est détecté, nous allons revenir dans l'analyse à l'étape où les informations sont reliées, c'est à dire à l'étape 4. le cycle est mis en évidence. Nous allons ensuite étudier les autres éléments de coopération afin de regarder si l'un d'entre eux n'est pas plus approprié. Dans la négative, cela signifie qu'un élément de coopération nécessaire n'est pas identifié. De deux choses l'une, soit c'est un simple oubli lors de l'inventaire des informations (on intervient sur les étapes 1 et 2), soit cette information n'est pas présente, il faudra alors définir la composition permettant sa réalisation (étape 4).

Les problèmes identifiés dans le second cas sont d'une toute autre nature. Ce point permet de mettre en évidence des éléments de coopération produits pour lesquels un ou plusieurs de leurs antécédents (les éléments nécessaire à leurs production) ne sont, soit pas identifié soit, pas mis en relation.

2.6 Étape 6 : de la méthode à l'implémentation - dérivation en règles

La production d'éléments de coopération composés peut être réalisée à partir des différents éléments de coopération (événements, messages, données) mais aussi à partir d'autres éléments coopération composés.

Éléments de composition produits	*Rôle*
Événement	▪ Cette composition permet de produire des événements de sémantique élevée et donc plus à même d'être utilisés dans un cadre coopératif.
Message	▪ La composition permet d'obtenir des messages « à valeur ajoutée », c'est à dire des messages complétés, traduits, mis en forme.
Donnée	▪ Génération de donnée synthétiques de sémantique élevée. Elle réalise de manière automatique la synthèse de certaines informations disséminées jusqu'alors et dont le regroupement était réalisé via la coopération externe.

Tableau 7 : Récapitulatif des composition d'éléments de coopération

Comme nous l'avons expliqué en introduction à cette thèse, un des objectifs est de rendre la méthode opérationnelle. Ainsi, en exploitant le langage de spécification nous devons pourvoir être capables de dériver les règles ÉCA et les règles détectrices.

Dans le langage de spécification nous pouvons voir quelles sont les règles détectrices et quels sont les éléments de coopération, avec leur type, entrant dans leur composition. A chaque fois que nous rencontrons l'occurrence de *Pour Règle* nous savons que l'élément de coopération identifié est destiné à cette règle, de même que nous connaissons la règle auquel est destiné cet élément. Nous connaissons donc pour chaque règle l'ensemble des éléments de coopération qu'il est nécessaire de lui communiquer.

Le rôle d'une règle détectrice est de produire un élément de composition en en composant d'autres.

Squelette d'une règle détectrice :

```
DéfinirRègleDétectrice nomRègle {
        -- Algorithme de composition --
        produireÉlementComposé(E_ACN) ;
}
```

Reprenons un extrait du langage de spécification (cf. Figure 40). Nous pouvons y voir que la règle RD_{ACN} nécessite l'événement E_{CNP} ainsi que le message *Chercher Colis N* pour créer l'événement composé E_{ACN}.

- **DéfinirÉvénement** E_{CNP} **De Groupe** *Réception* **Pour Règle** RD_{ACN}
- **DéfinirMessage** *Chercher Colis N* **De Groupe** *Service Commercial* **Pour Règle** RD_{ACN}
- **DéfinirÉvénement** E_{ACN} **De Règle** RD_A **Pour Groupe** *Expédition Nationale*
- **DéfinirÉvénement** E_C **Pour Module** IHM_1 **Dans Groupe** *Expédition Nationale* **as** E_{ACN}

Lors de chaque occurrence d'un des deux éléments de coopération précédents, il est nécessaire de le transmettre à la règle RD_{ACN}, ce qui se réalise de la manière suivante :

```
envoyer Événement E_CNP De Groupe Réception À RD_ACN
envoyer Message Chercher Colis N De Groupe Service Commercial À RD_ACN
```

Ces deux instructions (ainsi que d'autres que nous verrons plus loin) ne font pas exactement partie du langage de spécification. Elles sont proposées au sein d'une bibliothèque permettant aux entités de l'application de signaler la production d'éléments de coopération à la plate-forme coopérative.

C'est dans la partie – *Algorithme de composition* – que la règle va implémenter l'algorithme permettant l'évaluation des différents éléments de coopération auxquels elle est réceptive. Dans l'exemple ci-dessus, en fonction du corps du message *Chercher Colis N* et de l'événement E_{CNP}, elle produira éventuellement l'événement E_{ACN} qui sera récupéré par la plate-forme coopérative afin d'être acheminé vers les modules concernés (en l'occurrence ici, le module IHM_1 à la condition qu'il soit dans le groupe Expédition Nationale).

Nous n'allons pas écrire entièrement l'algorithme de composition mais en donner un extrait afin d'avoir une idée plus précise sur la manière dont il peut s'écrire. Nous mettrons en gras l'ensemble des informations pouvant être automatiquement générées.

```
DéfinirRègleDétectrice RD_ACN {
        // -- Début de l'algorithme de composition --
        String messageColisChercher = new String ( ) ;
        String evt = new String( ) ;
        while ( true ) { // une règle détectrice s'exécute à l'infinie
        evt = acceptRendezVous ( ) ;
        if (estÉvénement(evt, " E_ACN ","Réception") ) {
                produireÉlementComposé(E_ACN) ;
                }
        if (estMessage(evt, "Chercher Colis N", "Service Commercial") {
                messageColisChercher = lireBAL(« Chercher Colis N ») ; // Bien que l'on intervienne pas
sur le contenu du message, il est nécessaire de le retirer de la BAL.
                produireÉlementComposé(E_ACN) ;
                }
        } // Fin de la bouclie infinie
    } // -- Fin de l'algorithme de composition --
}
```

Comme nous pouvons le voir sur l'exemple précédent, la dérivation reprend l'ensemble des informations fournies par l'analyse. L'utilisation du dictionnaire sémantique des éléments de coopération et du dictionnaire des règles doit permettre l'écriture de l'algorithme.

Nous pouvons être surpris du fait qu'un rendez-vous est pris avec chacun des éléments de coopération y compris avec ce qui est identifié comme un message. Nous n'allons pas expliquer ceci dans cette partie, nous y reviendrons dans le paragraphe 0 de manière précise en expliquant ce choix par la même occasion.

La mise en œuvre de la coopération ne repose pas uniquement sur les règles détectrices. Nous avons aussi vu la nécessité des règles ÉCA qu'il est aussi possible de dériver afin de gérer la dynamique des groupes. Ce cas est en fait plus simple puisqu'une règle ÉCA ne prend en compte qu'un seul événement en entrée. De plus, contrairement à la règle détectrice, elle ne produit pas d'élément de coopération mais réalise une action : entrée/sortie de groupe d'un module. Au niveau de l'architecture coopérative, la gestion est elle aussi différente puisque contrairement aux règles détectrices qui sont actives tant que l'application coopérative fonctionne, les règles ÉCA sont elles déclenchées uniquement lors de l'occurrence de l'événement.

Nous allons présenter le squelette d'une règle ECA

```
DéfinirRègle nomRègle { // nomRègle est déclenchée sur l'occurrence de l'événement … produit par …
        acceptRendezVous ( ) ;
        if (Condition) // On vérifie la condition
                action // Si la condition est validée, on réalise l'action
        }
```

Nul besoin de se perdre en explications. Nous retrouvons dans ce squelette les deux parties Condition-Action (CA) de la règle ÉCA, la première instruction *acceptRendezVous( )* permettant d'acquitter le rendez-vous avec le gestionnaire de règles [Tawbi 96].

Nous allons montrer par exemple la dérivation de la règle ECA_{N1} (cf. Figure 34) à partir du langage de spécification et du dictionnaire des règles. Pour cela, nous identifiions les lignes dans lesquelles elle référencées :

- DéfinirÉvénement E_{N1} De Règle RD_{G1} Pour Règle $ÉCA_1$

Nous n'en trouvons qu'une, ce qui est logique puisqu'une règle ÉCA n'admet qu'un unique événement en entrée : une ligne dans le langage de spécification.

Grâce au dictionnaire des règles nous pouvons établir l'algorithme suivant.

```
DéfinirRègle ÉCAN1 { // ÉCAN1 est déclenchée sur l'occurrence de l'événement EN1 produit par la règle détectrice RDG1
        acceptRendezVous ( ) ;
        if ( ! memberOfClientPrivilégié(client.id)  { // Il n'y a pas de colis pour un client privilégié à traiter
                if (dansGroupe("IHM1", "Expédition Nationale") {
                        sortieGroupe("IHM1", "Expédition Nationale") ;
                        entréeGroupe("IHM1", "Expédition Régionale") ;
                        }
                else {
                        sortieGroupe("IHM1", "Expédition Régionale") ;
                        entréeGroupe("IHM1", "Expédition Nationale") ;
                }
        }
}
```

Dans les règles ÉCA, le gros du travail est l'écriture de la partie Condition-Action (CA) qui ne peut être générée de manière automatique à partir du langage de spécification. Nous pouvons remarquer l'utilisation de trois primitives particulières n'appartenant pas en natif au langage Java. Leur rôle est ici de gérer l'appartenance d'un module à un groupe - *dansGroupe(nomModule, nomGroupe)* – son entrée – *entréeGroupe(nomModule, nomGroupe)* – ou sa sortie – *sortieGroupe(nomModule, nomGroupe)*. Ces trois primitives seront plus amplement décrites lors de la présentation de la plate-forme coopérative. Elles font partie d'une bibliothèque livrée avec la plate-forme afin d'interagir avec elle.

3. Bilan

La méthode proposée ici permet de résoudre un certain nombre de problèmes rencontrés par exemple dans la modélisation des systèmes flexibles de production répartis tels ceux identifié par [Martineau 96] et [Briand 95]. En effet, la dynamique de tels systèmes peut se modéliser à l'aide de groupes de travail composés de modules (cellules de production, logiciels de conduite et de supervision, …).

Il existe une demande forte des industriels concernant la modélisation de l'activité incorporant la notion de flexibilité. Les applications industrielles sont d'autant plus intéressantes qu'elles induisent naturellement les problèmes liés à la supervision et à la prise de décision en temps réel auxquels notre approche peut apporter des solutions.

Plus généralement, la méthode veut apporter des éléments de réponse dans le contexte plus large des applications coopératives distribuées pour lesquelles il convient d'intégrer les aspects liés à l'utilisation d'un réseau et les aspects coopératifs induits par de nouveaux modes de travail.

L'approche descendante proposée autorise les retours en arrière permettant de procéder par affinages successifs. En outre, tout au long du cycle de vie de l'application il est possible, sans remettre en question l'analyse déjà effectuée, d'ajouter des groupes de travail, des modules, ou bien de revenir sur des éléments non identifiés jusqu'alors mais néanmoins nécessaires, et plus généralement, sur tout le travail ayant été effectué jusqu'alors.

La méthode focalise de façon homogène lors de chaque étape sur la mise en évidence des événements, des messages et des données sur lesquels s'appuie la coopération. La solution proposée offre également l'avantage de l'homogénéité au niveau de la cible basée sur le seul mécanisme des règles détectrices et ÉCA.

En ayant défini un langage de spécification possédant une grammaire volontairement simple, nous pouvons réaliser les vérifications nécessaires. En définissant une grammaire algébrique adaptée, il serait très aisé de rendre toute cette étape entièrement automatique.

Le langage de spécification ne permet pas de corriger les problèmes rencontrés mais de les mettre évidence. Pour chacun d'eux, il sera nécessaire de revenir dans l'analyse au niveau du problème qui est aisément identifiable en fonction de l'information concernée.

Enfin, il est possible d'aboutir à une description par un langage de spécification de l'ensemble de la coopération. La dérivation en règles détectrices et ÉCA à partir du langage de spécification permet d'automatiser grandement les parties les plus répétitives.

Après chaque retour pour modification dans l'analyse, il faudra à nouveau réaliser les vérifications afin de s'assurer que la correction effectuée n'a pas généré d'autres problèmes comme la création d'un cycle par exemple.

L'architecture cible trouvera dans le langage de spécification la totalité des informations lui permettant de fonctionner et d'assurer la coopération des modules et la gestion des groupes de travail. La mise en place de règles réalisées par des processus parallèles avec lesquels sont pris des rendez-vous lors de l'occurrence des éléments déclenchants permet ensuite d'assurer le fonctionnement de cette couche active. Nous aurons ainsi une méthode permettant d'aller jusqu'à l'implémentation complète de l'application coopérative.

CHAPITRE 4

LA PLATE-FORME COOPERATIVE

1. Objectifs

Les différentes étapes de la méthode ont fait apparaître de nombreuses informations :

1. Nous avons premièrement réalisé l'inventaire des éléments de coopération de l'application (événements, messages, données),
2. Nous avons mis en relation ceux de ces éléments qui avaient un lien de coopération direct (lien de type producteur - consommateur),
3. Par composition d'éléments de coopération, nous avons créé ceux qui manquaient afin d'automatiser la coopération externe,
4. Nous avons mis en évidence la dynamique des groupes de travail et avons trouvé les conditions d'entrée/sortie des modules concernés.
5. Enfin, nous avons mis en œuvre une vérification formelle afin de vérifier la plausibilité d'occurrence des éléments de coopération et avons réalisé la dérivation afin de faciliter l'implémentation.

Toutefois, notre projet global va au delà de la simple proposition d'une méthode pour modéliser la coopération dans les applications distribuées. Nous gardons en permanence le souci de la rendre opérationnelle. Pour arriver à ce résultat, il nous faut pouvoir concrétiser l'ensemble des points énumérés ci-dessus. La spécificité de notre approche liée au domaine de la ré-ingénierie nous oblige à proposer des solutions permettant de toucher le moins possible, et idéalement pas du tout, à l'existant.

Il nous faut donc pour cela pouvoir :

1. Capturer et injecter les éléments de coopération mis en évidence, c'est à dire mettre en œuvre les mécanismes de détection/production d'événements, messages et données.
2. Gérer les différents éléments de coopération permettant leur circulation des entités émettrices aux réceptrices.
3. Agir sur les interfaces des modules afin de créer de nouvelles entrées,
4. Créer les informations manquantes par composition de celles existantes,
5. Mettre en œuvre un mécanisme permettant de gérer la dynamique des groupes.

Nous n'allons pas pour cela partir de zéro et allons nous appuyer sur une architecture déjà réalisée : ADACTIF. Nous partons de son modèle d'exécution qu'il est nécessaire d'adapter de manière à ce qu'il puisse prendre en compte l'ensemble des caractéristiques que nous avons énumérées précédemment. Il faut qu'il puisse gérer les différents éléments de coopération, la mise en réseau ainsi que les groupes de travail dynamiques.

1.1 Gestion des éléments de coopération

Les différents modules présents dans le système sont susceptibles de recevoir et de produire les trois types d'éléments de coopération (événements, messages et données). Lorsqu'un élément est produit au sein de l'application coopérative, il faut connaître les entités (modules, et/ou règles détectrices, et/ou règles ÉCA) auxquelles il doit être signalé, sachant que ces entités peuvent être éventuellement distantes. Nous puiserons dans le langage de spécification l'ensemble des renseignements nécessaires à la bonne gestion de chacun des éléments de coopération.

Ce rôle d'aiguillage sera alloué au Gestionnaire d'Éléments de Coopération ($G_{ÉC}$), extension du gestionnaire d'événements d'ADACTIF, qui sera tenu au courant de l'ensemble des éléments de coopération produits dans l'application coopérative et se chargera de les transmettre aux entités concernées.

Or, si l'on désire réaliser cette transmission, il faut être aussi capable de les récupérer là où il se produisent, c'est à dire à l'origine même de l'entité émettrice.

1.2 Interactions des modules avec leur environnement

Les modules logiciels et physiques peuvent produire et récupérer des informations par des moyens très variés :

- Par variables partagées,
- Par fichiers,
- Par bases de données par le biais d'un SGBD,
- Par connexion réseau (type client/serveur, bus logiciel, courtiers d'objets, …) avec un modem ou une carte réseau,
- Par des lignes physiques de communication,
- Par les ports physiques de la machine (ports parallèles, séries, USB, FireWire, …),
- Par actions sur l'interface, et donc de manière indirecte, par tout périphérique de type souris, clavier, tablette graphique, micro, … pour les entrées, et tout périphérique de type écran, haut-parleurs, cartes dédiées, … pour les sorties.

Notre problème est donc de capturer et de produire ces informations sans pour autant perturber le fonctionnement du module. Dans le cas de modules physiques (automates, cellules de productions, machines outils, …), le moyen le plus efficace consiste à « écouter » ou à « activer » certaines lignes. Il ne s'agit pas de le faire pour toutes les lignes, mais uniquement pour celles qui nous intéressent. En ce qui concerne les modules « logiques », nous devrons adapter nos moyens au coup par coup.

La récupération et la production d'informations est une partie délicate de la plate-forme coopérative :

- Il est inutile de capturer l'ensemble des événements et messages produits. La récupération ne concerne bien évidemment que ceux qui ont été mis en évidence au cours de l'analyse. Nous avons vu lors des étapes précédentes que chaque information nécessite une qualification, c'est à dire qu'elle doit être « étiquetée » avec une origine et une destination. Lorsqu'un module produit une information, celle-ci ne contient évidemment pas la qualification de son origine. Cette dernière ne peut être qu'ajoutée par la plate-forme coopérative.
- Les événements et messages significatifs devant être transmis aux modules doivent l'être sur le médium et selon le format attendu par celui-ci. Par exemple, l'arrivée d'un message sur un module pourra se traduire par son ajout dans le contenu d'un fichier afin que le module trouve l'information lorsqu'il ira lire ce fichier. Dans d'autres cas, il sera possible de produire une entrée pour un module en modifiant son interface en conséquence afin que cette information soit portée à la connaissance de l'opérateur. Ce pourra être par exemple l'affichage d'une boite de dialogue contenant tout ou partie d'un message d'alerte à l'écran.

Ce travail de récupération et de production d'éléments de coopération ainsi que ce travail d'action sur les interfaces des modules sera le rôle du Gestionnaire de Modules (GM) que nous décrirons plus en détail dans ce chapitre.

1.3 Création des éléments de coopération manquants

Au cours de la méthode, nous avons mis en évidence le besoin de créer des informations manquantes représentant synthèse, une traduction ou une mise en forme. Ces nouvelles informations peuvent aussi être utiles à la gestion de la dynamique des groupes.

Cette création d'éléments de coopération peut se faire à partir d'éléments produits par différents modules quels qu'ils soient et où qu'ils soient. Ainsi, chaque information créée doit elle aussi être identifiée afin de savoir s'il est nécessaire de l'acheminer sur un autre site et si oui, vers quelle entité.

Le problème de savoir où est située physiquement cette entité est d'une autre nature et relève de la partie implémentation pure, nous éviterons d'en parler ici.

Une fois de plus, nous allons réutiliser le langage de spécification. Il nous permet de connaître pour chacun des éléments de coopération composés, l'ensemble des informations nécessaires avec leur provenance logique. Une fois la composition réalisée, il nous permet également de connaître la ou les destinations de l'information nouvellement produite.

À chaque fois qu'un élément de coopération entrant dans une composition sera produit, il sera acheminé vers chacune des règles détectrices, une par élément de coopération à créer. Celles-ci exécuteront leur algorithme de composition respectif et, le cas échéant, produiront l'information résultante qui sera récupérée par la plate-forme coopérative pour être acheminée aux groupes et modules concernés, ou pour directement être incluse dans le SI.

Concernant la localisation physique des règles détectrices, sans entrer dans le détail de l'implémentation, nous avons fait le choix de les placer sur le site où doit être produit l'élément composé.

1.4 Circulation des éléments de coopération

L'étiquetage des éléments de coopération permet à la plate-forme de connaître l'émetteur. A partir de l'information munie de son origine, la plate-forme coopérative est capable d'identifier la ou les entités destinatrices afin de leur acheminer l'élément de coopération concerné.

Il faut pour cela que, lorsqu'un élément de coopération est produit, il soit récupéré par la plate-forme coopérative, qualifié puis envoyé aux groupes et modules concernés avec éventuellement une étape de mise en forme, de traduction, et/ou de complétion.

Nous devons garder à l'esprit que nous travaillons dans un environnement distribué et donc que certains éléments de coopération auront à être signalés et pris en compte sur des sites distants.

Les éléments de coopération produits par un module (ou une par règle détectrice) doivent parfois être communiqués à d'autres entités situées à l'autre bout du réseau. Pour réaliser cela, il est nécessaire de pouvoir les diffuser, tout comme il est nécessaire de pouvoir les recevoir au travers de ce même réseau. Se pose alors le problème de savoir où et à quels modules il faut acheminer un élément reçu. Ce problème, directement issu de l'apport de la distribution, est un problème que nous allons qualifier de « classique » dans ce type d'architecture où il est nécessaire de savoir où acheminer les informations. Si dans un contexte local et non coopératif le destinataire suffit, dès que l'information

nécessite d'être diffusée, l'émetteur doit être ajouté dans la qualification de l'élément coopératif. Cette obligation s'explique par l'introduction de la notion de groupes dynamiques. Une même information peut être produite pas des modules différents. Nous entendons par même information : même identificateur et même contenu s'il a lieu d'être. La qualification prend ici toute son importance puisqu'elle permet de différencier une information par son producteur ainsi que par son destinataire. Il est des cas où l'élément produit change de destinataire selon le groupe dans lequel se trouve le module producteur. Ceci explique cette nécessité de pleinement qualifier un élément de coopération.

À partir de la méthode et plus particulièrement à partir du langage de spécification, nous obtenons de précieuses informations. Nous connaissons son type, par quel module, situé dans quel groupe un élément de coopération a été produit, de même que nous connaissons avec précision le ou les destinataires - module(s), groupe(s) ou règle(s) – c'est la qualification des éléments de coopération qui sera automatiquement incluse dans leur contexte. Ainsi, chaque élément transitant sur le réseau sera « étiqueté » avec les informations correspondantes à sa ou ses destinations, ainsi sa provenance.

Il faut donc prévoir un dispositif dont le rôle sera de relier chacune des plate-forme coopératives présentes sur chacun des sites du réseau. Il devra être à l'écoute du réseau afin de recevoir les informations le concernant, mais devra aussi pourvoir envoyer des éléments sur ce même réseau lorsque ceux-ci nécessiteront d'être diffusés.

Lorsque ce dispositif reçoit un élément de coopération en provenance du réseau, il est nécessaire qu'il soit transmis au $G_{ÉC}$ afin que celui-ci puisse en tenir compte et qu'il puisse le transmettre aux entités concernées (modules, groupes, règles) [Roose 99c].

Pour réaliser ce travail de liaison avec l'« extérieur », nous ajoutons un gestionnaire spécifique, le Gestionnaire de Communication (GC) qui sera le lien entre chacune des plate-formes coopératives présentes.

Remarque : Les données mises en évidence dans la coopération sont exclues de ce mode de fonctionnement. En effet, il s'agit de données partagées dans le SI de sorte que les entités qui les manipulent y accèdent directement.

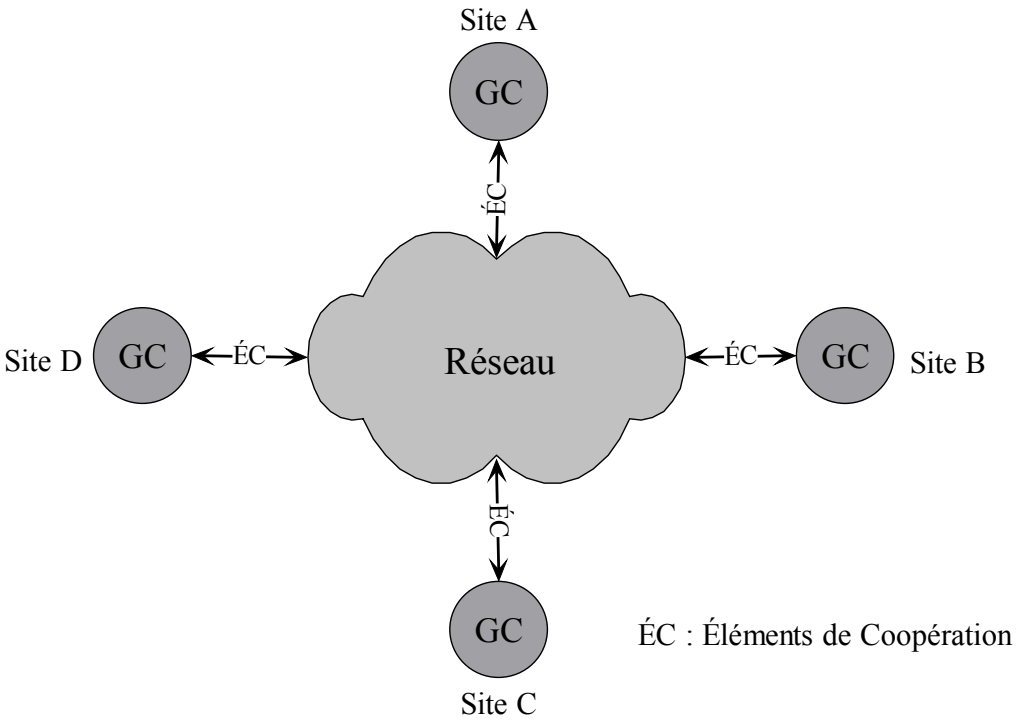


Figure 50 : Circulation des éléments de coopération via les GC de chaque site

L'incorporation d'un GC modifie sérieusement le modèle d'exécution d'ADACTIF. Nous avons maintenant deux niveaux de gestion des éléments de coopération. Ceux produits sur un site et ne nécessitant pas d'être envoyés sur un autre (informations locales au site) et ceux qui en plus d'avoir à éventuellement être traités localement, doivent être envoyés à un ou plusieurs autres site sur lesquels ils seront reçus par le GC local.

Ainsi, à chaque fois que le $G_{ÉC}$ détectera localement un élément de coopération, il ira regarder dans sa base s'il doit être ou non diffusé, si oui, il (avec son contexte qui est en fait sa qualification – origine, destination) sera communiqué au GC afin que celui-ci le transmette sur le réseau vers les autres GC concernés.

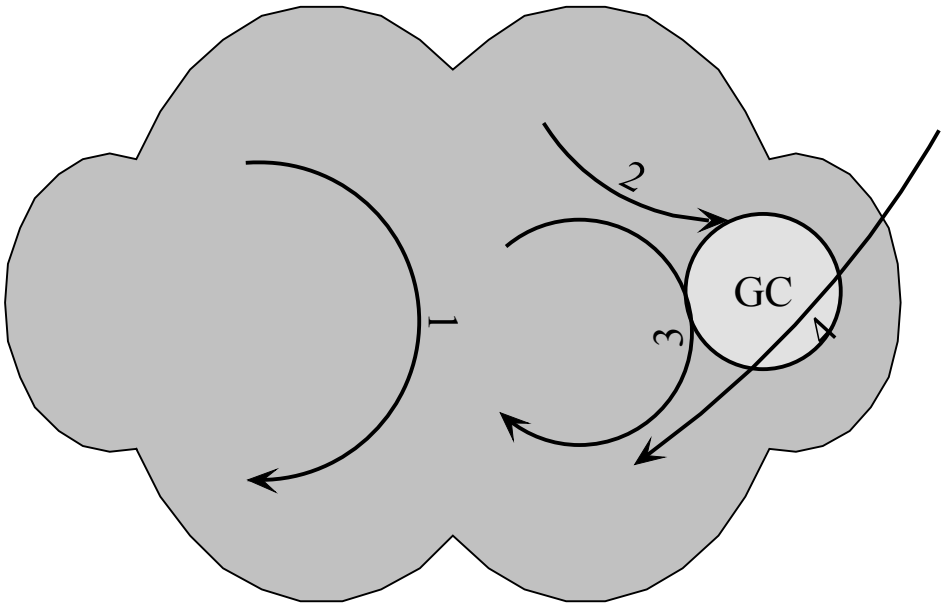


Figure 51 : Éléments de Coopération locaux/à diffuser

Nous pouvons rencontrer quatre cas de production d'éléments de coopération :

1. Pas de diffusion au travers du réseau, il reste entièrement local et donc ne sera pas transmis au GC.
2. L'élément ne nécessite pas une prise en compte locale mais doit être communiqué à d'autres entités distantes.
3. Ce cas illustre une information nécessitant une prise en compte à la fois locale, mais aussi distante.
4. Une information produite sur un site quelconque a été envoyée sur ce site afin d'informer une entité locale.

1.5 Gestion de l'état de l'application coopérative

Nous avons vu dans le chapitre précédent que la gestion des groupes dynamiques était dévolue aux règles ÉCA, une par module et en autant d'exemplaires que de groupes dynamiques auxquels peut appartenir ce module. Lorsqu'un rendez-vous est pris avec l'une d'elles suite à l'occurrence de son événement déclencheur, elle évaluera sa condition afin de savoir si le module qui lui est rattaché doit entrer ou sortir du groupe ou bien s'il faut ignorer cet événement.

La décision d'entrée/sortie de groupe tenir compte de la composition même des groupes de travail, de leur activité, mais aussi de l'état général de l'application coopérative. Il faut donc stocker dans une base de données les informations de gestion de groupes (composition) auxquelles pourront se référer

les règles de gestion de la dynamique. Chaque règle de gestion pourra consulter cette BD, et donc l'état global de l'application coopérative, et prendre ainsi sa décision concernant l'entrée ou la sortie du module en ayant connaissance de l'ensemble des informations nécessaires.

Nous avons précisé en introduction de la méthode qu'il est possible de destiner un élément de coopération à tout un groupe, cela veut dire que cette information doit être signalée à l'ensemble des modules composant le groupe identifié. La plate-forme coopérative doit donc aussi pouvoir connaître les modules constituant chacun des groupes afin de pouvoir transmettre l'information lorsqu'ils sont concernés et où qu'ils se trouvent physiquement.

Toutes les informations nécessaires à la gestion des groupes de travail dynamiques seront stockées dans une base de données centralisée (*BD-ÉTAT*) représentant l'état de l'application, c'est à dire :

- La liste des groupes de travail dynamiques,
- La liste des modules avec leur localisation physique,
- La liste des module intégrés à chaque groupe de travail.

2. Le modèle d'exécution d'ELKAR

En nous replaçant dans le contexte de la ré-ingénierie d'applications non coopératives, nous avions à notre disposition un ensemble de modules avec leurs entrées et sorties avec une coopération établie par l'extérieur (intervention humaine, programmes dédiés, …). En s'appuyant sur la méthode développée ainsi que sur le langage de spécification qui en découle, notre plate-forme coopérative doit pouvoir remplacer cette coopération extérieure tout en autorisant de surcroît une coopération plus performante en terme de rapidité mais aussi en terme de qualité.

Dans cette partie, nous nous appuyons sur les travaux réalisés dans l'équipe avant mon arrivée et qui portaient sur les Bases de Données Actives (BDA). Ces travaux étaient jusqu'alors menés dans un contexte que l'on peut qualifier de « traditionnel ». Nous entendons par là qu'ils étaient menés dans un environnement local. Le résultat a été ADACTIF (décrit au début de cette thèse) dont les principales caractéristiques sont une implémentation des règles ÉCA sous forme de tâches s'exécutant en parallèle avec prises de rendez-vous permettant d'assurer la synchronisation.

Remarque : une tâche est une unité s'exécutant en parallèle avec d'autres unités du système. Durant sa vie et à leur demande, elle rend des services aux autres unités du système.

La synchronisation entre le déclencheur et la tâche est réalisée grâce au mécanisme de rendez-vous de type ADA [ADA 83]. Ainsi, la demande d'un service offert par une tâche T est conclue d'une part par la sollicitation d'un rendez-vous avec T, et d'autre part par son acceptation par T. Dès que la tâche sollicitée accepte le rendez-vous, le service demandé est exécuté et les deux unités (appelante et appelée) poursuivent leur exécution en parallèle. Dans cette manière de voir les tâches, la demande de rendez-vous est une synchronisation puisque les service demandeur doit attendre jusqu'à ce que la tâche appelée lui fournisse le service demandé.

Chaque tâche possède une interface, contenant les entrées visibles par l'utilisateur de la tâche, et un corps définissant les actions de la tâche.

L'autre contribution d'ADACTIF est la mise en œuvre de règles détectrices permettant la composition d'événement (aussi complexes que nécessaire à partir du moment où c'est exprimable par un algorithme) et la production d'événements composés.

Le Modèle d'exécution d'Adactif

Nous présentons ici de manière plus conséquente le modèle d'exécution qui (avec l'analyseur de commandes que nous ne décrirons pas dans cette thèse, le lecteur se référera à [Tawbi 96] pour de plus amples informations) permet de rendre actif un SGBD.

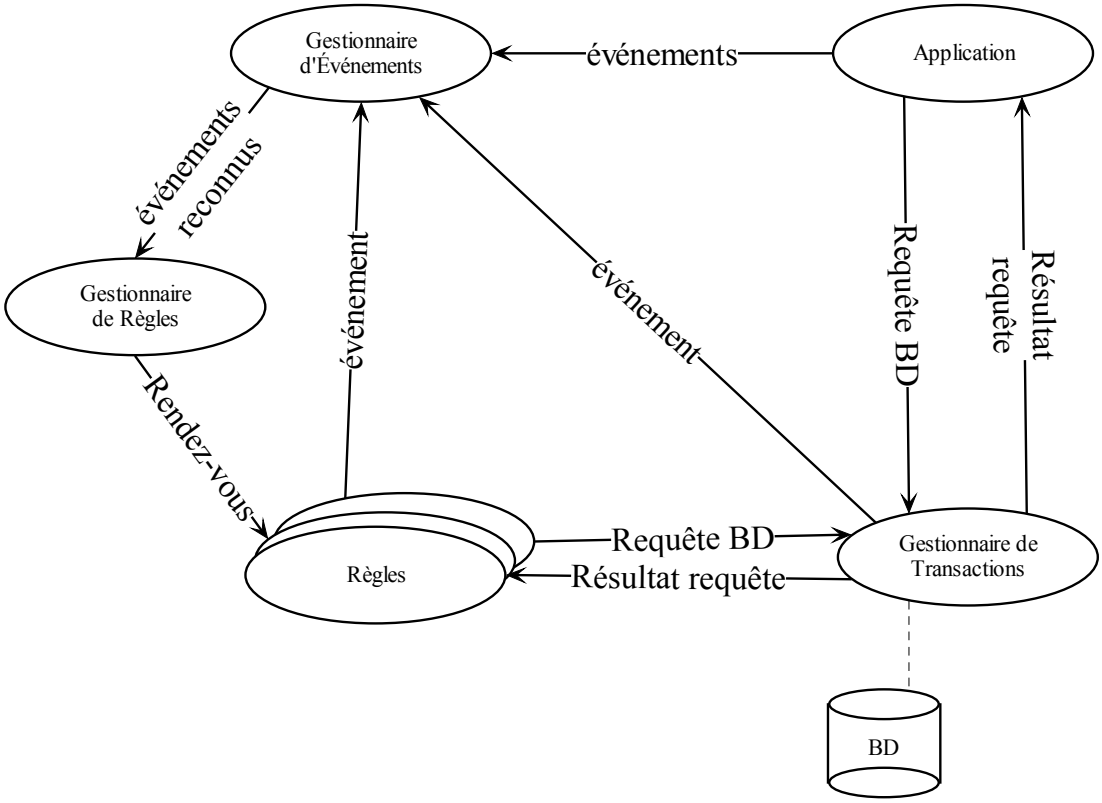


Figure 52 : Le modèle d'exécution d'ADACTIF

Tout comme les règles, les divers gestionnaires (d'événements, de règles, …) sont des tâches s'exécutant en parallèle. En simplifiant les choses, l'organisation dans le modèle est la suivante :

Lorsqu'un événement se produit dans le système, il est envoyé au Gestionnaire d'Événéments (GÉ). S'il est reconnu, il est alors envoyé au Gestionnaire de Règles (GR) qui prendra rendez-vous avec les règles concernées par cet événement.

Les règles (ÉCA ou détectrices dans le cas d'événements composés) évaluent une condition et réalisent éventuellement une action pouvant être une opération BD, un contrôle d'intégrité, et peuvent même lever des événements qui seront à leur tour détectés puis envoyés au GÉ … repartant ainsi dans le cycle du modèle d'exécution.

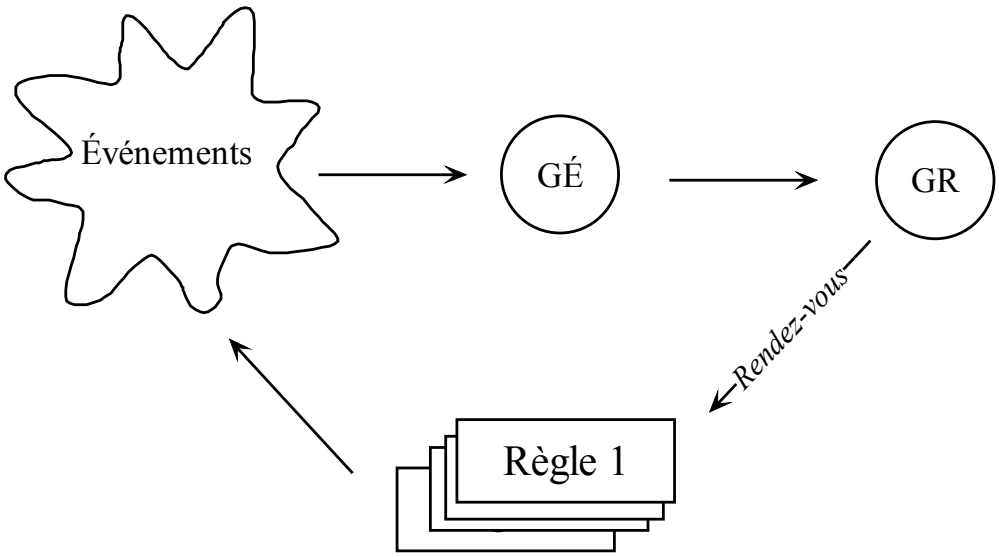


Figure 53 : Circulation des événements dans ADACTIF

Les règles ainsi que les applications peuvent réaliser des transactions sur la BD par l'intermédiaire du Gestionnaire de Transactions (GT). En fonction des requêtes à réaliser, il est susceptible de lever des événements qui feront réagir à leur tour le système en suivant le cycle bien établi : ❶GÉ… ❷GR… ❸Règles.

Ce modèle est souple et possède de nombreux avantages : une architecture modulaire (GÉ, GR, GT) s'exécutant en parallèle, des règles, s'exécutant elles aussi en parallèle et pouvant interagir avec le système, ainsi qu'une gestion efficace des événements (possibilité de composer des événements entre eux pour en produire de plus expressifs).

Les résultats obtenus ont laissé penser à l'équipe que l'approche qu'ils avaient eu pouvait être étendue à des domaines comme le travail coopératif. Ainsi, nous avons étendu le modèle d'ADACTIF pour qu'il réponde à nos besoins exprimés lors de l'analyse. Il faut pour cela :

- Étendre la gestion (détection, production) des événements aux éléments de coopération (événements, message),
- Permettre une gestion distribuée des éléments de coopération,
- Mettre en œuvre des mécanismes de gestion des groupes de travail dynamiques.

Ainsi, nous proposons une extension de l'architecture et des principes de règles et de rendez-vous d'ADACTIF adaptés au domaine du réseau et à celui du travail coopératif.

Nous devons en premier lieu actualiser les outils d'ADACTIF c'est à dire :

- Permettre le déclenchement de tâches par l'intermédiaire du réseau, ce qui sous entend une extension réseau d'ADACTIF afin de diffuser les éléments de coopération,
- Étendre de la notion de règles détectrices initialement réservées aux événements afin qu'elles puissent prendre en compte et produire les différents éléments de coopération,
- Autoriser la circulation des différents éléments de coopération entre les différents gestionnaires.

Les parties suivantes vont tacher d'expliquer comment nous réalisons cette extension du modèle d'exécution afin qu'il réponde à nos besoins. Nous démarrons par l'introduction de primitives.

La BD-ÉTAT contient un ensemble de tables permettant de réaliser la correspondance Module ⇔ Groupes. Des primitives internes à la plate-forme coopérative sont fournies afin d'être utilisées par les gestionnaires, mais aussi par les règles. Elles pourront :

- Obtenir la liste des membres d'un groupe, *membreDe(nomGroupe)* permettant de vérifier l'état (au sens position dans les groupes) des modules et des groupes. L'intérêt est de vérifier avant de réaliser l'entrée ou la sortie d'un module d'un groupe s'il n'est pas déjà présent où au contraire s'il est bien présent dans ce groupe. En obtenant l'état du groupe, c'est à dire la liste de ses composants, cela permet d'éviter par exemple, l'incorporation d'un module dans un groupe de travail suffisamment fourni ce qui aurait pour effet soit de ne rien améliorer, soit au contraire de diminuer les performances du groupe si celui-ci risque de puiser dans les ressources limitées des autres.
- Récupérer le nom du ou des groupes auxquels appartient un module, *estDansGroupe(nomModule)*
- Joindre un groupe de travail, *joindreGroupe(nomGroupe, nomModule)*. Dans le cas où le module est déjà dans ce groupe, cette primitive est sans effet.
- Et enfin, quitter un groupe de travail, *quitterGroupe(nomGroupe, nomModule)*. Là aussi, si la primitive est appelée alors que le module précisé n'appartient pas au groupe mentionné, la primitive est sans effet.

Nous ne décrirons pas le schéma conceptuel de données correspondant puisqu'il suffit de deux tables (une pour les groupes, une pour les modules) à mettre en relation. Ce niveau de détail n'apporterait rien ici. Nous allons maintenant nous intéresser à l'organisation des gestionnaires et à la manière dont circulent les éléments de coopération au sein de la plate-forme coopérative.

Modèle d'exécution d'ELKAR

Le modèle d'exécution va nous permettre de représenter en détail le cheminement des éléments de coopération à l'intérieur de la plate-forme coopérative.

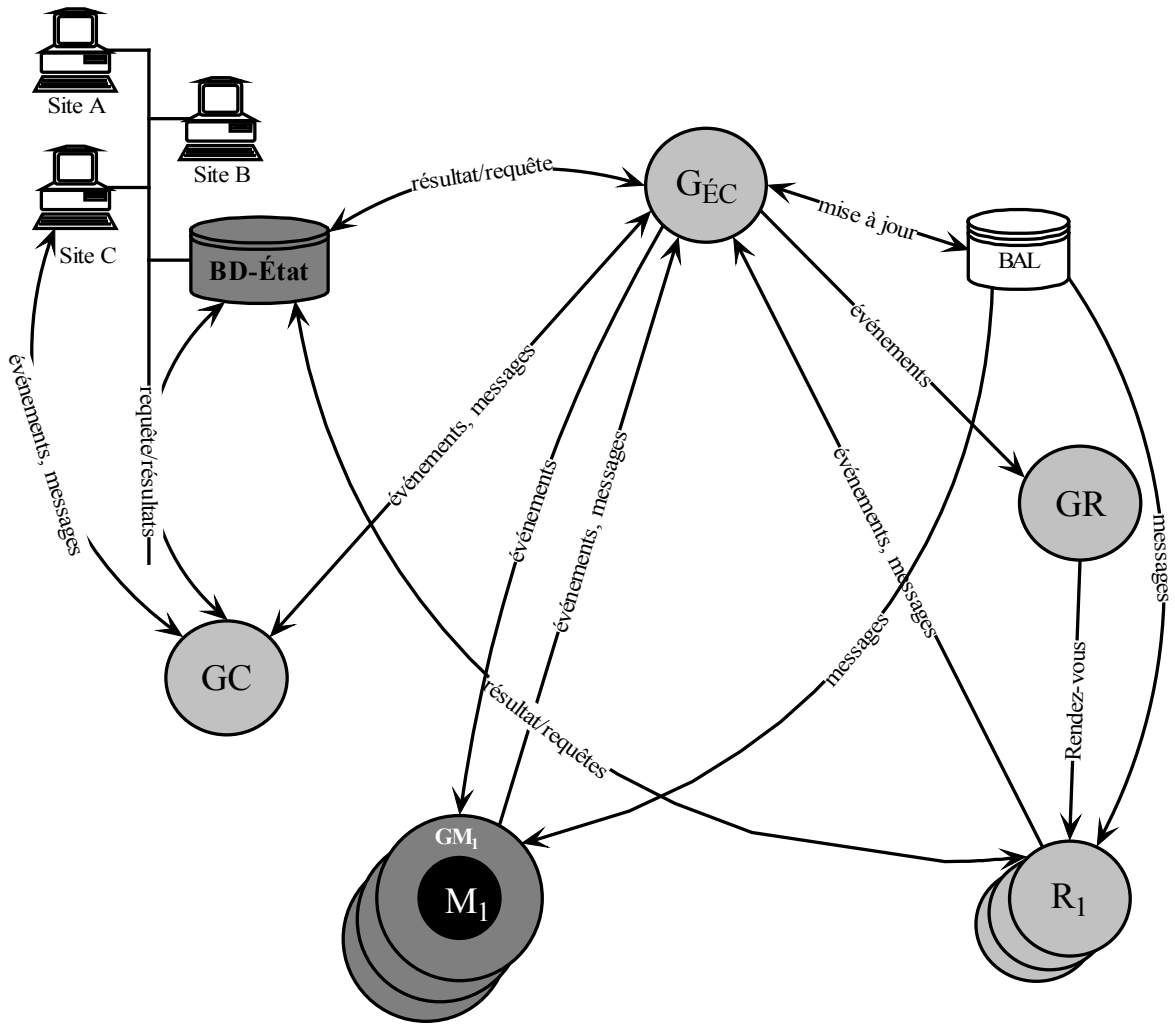


Figure 54 : Modèle d'exécution de la plate-forme coopérative

2.1 Le Gestionnaire d'Éléments de Coopération

C'est par lui que passent tous les éléments de coopération détectés au sein de la plate-forme coopérative, que ce soient des événements ou des messages puisque les données sont directement manipulées dans le SI de l'application. Lorsqu'un élément de coopération identifié au cours de

l'analyse est produit, il est envoyé au $G_{ÉC}$ par l'émetteur étiqueté par son origine. Le premier travail du $G_{ÉC}$ est alors de regarder si l'élément de coopération a été répertorié comme étant une information devant être localement traitée ou pas.

Une fois l'élément de coopération récupéré reconnu, deux cas de figures se présentent :

- *L'ÉC est un événement*. Il est directement transmis au GM des modules concernés s'il y a lieu, et au GR qui prendra rendez-vous avec les règles concernées. Si cet événement doit être diffusé, il est transmis au GC.
- *L'ÉC est un message*. Dans ce cas, la gestion est légèrement différente. A chaque message ayant été répertorié au cours de l'analyse (via le langage de spécification), un événement a été associé. Le $G_{ÉC}$ stocke le message dans une boite à lettres (BAL) locale. En réponse, il transmet l'événement associé au message au GM et/ou aux règles concernées qui réagiront en conséquence. Si ce message doit être diffusé, il est transmis au GC.

Lorsqu'un élément de coopération est transmis au $G_{ÉC}$ par une règle ou un GM local, celui-ci doit décider du traitement à lui faire subir en fonction des destinataires de cet élément de coopération. Différents cas se présentent alors.

- Le destinataire est local
  - C'est une règle (*Pour Règle R*). Si c'est un événement, il sera transmis au gestionnaire de règles. Si c'est un message, il sera stocké dans la BAL et le destinataire sera informé par un événement qu'il doit aller cherche le message.
  - C'est un module (*Pour Module M*). Si c'est un événement, il est envoyé au gestionnaire de module correspondant afin que celui-ci l'incorpore dans l'environnement de son module. Si c'est un message, il sera stocké dans la BAL et le GM en est averti par un événement.

- Le destinataire est distant :
  - C'est un module (Pour Module M). Le $G_{ÉC}$ complète sa qualification en ajoutant la destination puis le transmet au GC qui diffusera l'information aux sites concernés via leur GC respectif.
  - C'est une règle. Le $G_{ÉC}$ complète sa qualification en ajoutant la destination puis le transmet au GC qui diffusera l'information aux sites concernés via leur GC respectif.

- Le destinataire est un groupe (*Pour Groupe G*). Trois solutions s'offrent à nous :
    - Tous les modules de ce groupe sont locaux au site. Dans ce cas, l'information ira à l'ensemble des modules locaux à ce groupe par l'intermédiaire de leur gestionnaire de module respectifs.
    - Tous les modules sont extérieurs au site. Le $G_{ÉC}$ complète la qualification de l'élément de coopération (celui-ci contenait déjà les informations quant à son origine) en ajoutant sa destination puis le transmet au gestionnaire de communication afin qu'il l'envoie sur le ou les sites hébergeant au moins un module membre de ce groupe.
    - Seule une partie des modules sont locaux. Ce cas est le plus complexe qui regroupe en fait les deux points précédents. L'élément de coopération sera transmis tel quel aux GM locaux mais sera aussi communiqué au GC avec la qualification nécessaire à sa diffusion.

2.2 Le Gestionnaire de Règles

Son rôle est de trouver toutes les règles concernées par un événement qui lui aura été transmis par le $G_{ÉC}$. Nous pouvons avoir dès lors trois cas de figure :

- Une ou plusieurs règles ÉCA sont concernées : le GR lance puis prend rendez-vous avec la ou les règles ÉCA correspondantes,
- L'événement entre dans la composition réalisée par une ou plusieurs règles détectrices : comme elles sont actives en permanence, le GR va prendre rendez-vous avec chacune des règles concernées pour le leur notifier. Si l'événement correspond à un message, la règle interrogera la Boite à Lettre (BAL) locale de manière à récupérer son contenu.
- L'événement concerne à la fois des règles ÉCA et des règles détectrices : le GR lance et prend rendez-vous avec les règles ÉCA concernées et communique l'information (sans avoir à les lancer) aux règles détectrices correspondantes.

Pour connaître les règles à qui notifier les événements, le GR récupère dans le langage de spécification les parties le concernant, c'est à dire dès que *Pour Règle* apparaît pour la déclaration d'un élément de coopération. Le GR va aussi bien utiliser les déclarations d'événement (*DéfinirÉvénement*) que celles

de messages (*DéfinirMessage*). Pour cette dernière, les événements déclenchés n'apparaissent pas lors de l'analyse mais sont automatiquement produits par l'arrivée du message.

2.3 Le Gestionnaire de modules

A chaque module est associé un de Module. C'est un point délicat de l'application coopérative puisque c'est lui qui va être chargé de récupérer les éléments de coopération dans l'environnement du module, et c'est lui aussi qui les incorporera dans ce même environnement.

Lorsqu'un module produit un message, le GM l'intercepte et le transmet au $G_{ÉC}$ qui le traitera conformément à la description que nous avons réalisée au paragraphe précédent. Celui-ci terminera éventuellement sa qualification pour le transmettre au GC ou le placera dans la BAL et enverra l'événement correspondant aux GM concernés. Celui-ci n'aura plus qu'à le récupérer et l'introduire dans l'environnement du module de manière à ce qu'il puisse être pris en compte de la manière désirée.

Nous avons déjà longuement expliqué les types d'éléments de coopération, mais nous n'avons pas encore expliqué comment les capturer. Nous allons donc ici tenter de présenter un éventail le plus complet possible des moyens utilisés pour recevoir/envoyer des éléments de coopération, sachant toujours que les modules peuvent être aussi bien physiques que logiciels. Nous n'allons pas entrer ici au niveau de l'implémentation mais tenter de présenter les moyens classiques d'interaction avec les modules.

2.3.1 Utilisation de primitives

La ré-ingénierie sans la moindre modification a ses limites. Il sera donc parfois nécessaire, lorsque c'est possible, de faire une légère entorse à ce que nous avons toujours défendu depuis le début de cette thèse…aucune intervention sur les modules.

Dans le cas où toute modification est impossible, c'est à dire où l'introduction de primitives (par un moyen ou par un autre) n'est pas permise, nous devrons faire avec ce qu'il y a. La ré-ingénierie est un domaine particulier et il n'est pas certain toujours arriver exactement à ce que l'on désire. De deux choses l'une, ou bien le résultat obtenu est insuffisant et dans ce cas là il n'y a pas grand chose à faire, ou bien il est possible par des moyens détournés d'arriver à une solution suffisamment proche pour que ce soit satisfaisant.

Lorsque l'introduction de primitives est possible, il faut que les mises à jour que nous allons proposer n'altèrent en rien le fonctionnement du module. Aussi allons-nous définir des primitives simplement chargées de prévenir la plate-forme coopérative lors de la détection de l'élément de coopération, comme par exemple un clic sur un bouton.

Ainsi, à chaque production par le module d'un élément de coopération nécessitant d'être notifié, et ne pouvant être détecté depuis l'extérieur de ce module, une primitive est insérée. Réciproquement, lorsqu'une entrée était réalisée par le biais de la coopération externe, cette entrée est remplacée par l'appel à la primitive permettant au GM de réaliser l'incorporation de l'élément de coopération dans l'environnement du module.

Les primitives sont les suivantes :

ProduireÉvénement (*nomÉvénement*)
ProduireMessage(*message*)
RécupérerÉvénement(*bloquant*)
RécupérerMessage(*bloquant*)

Lorsqu'elles sont insérées, les deux premières primitives permettront au GM de récupérer les éléments de coopération (événement, message) non détectables. Les deux secondes permettront au contraire d'injecter dans le module un élément de coopération, là aussi, un événement ou un message, avec deux possibilités : appel bloquant ou non bloquant. Dans les cas où la poursuite de l'exécution du module est cautionnée par l'arrivée (ou le contenu) de l'élément de coopération, nous aurons un appel dit « bloquant », permettant une synchronisation avec cette information.

Nous allons illustrer ces primitives par un exemple :

Exemple . *Nous avons vu que nous pouvions créer des entrées indirectes au niveau des modules. Nous l'avons fait avec les IHM et l'événement E_{CS}. Ainsi, il sera possible d'insérer la primitive (non blocante) « RécupérerÉvénement(faux) » permettant aux interfaces d'être mises au courant par le GM lorsqu'un colis a été sélectionné par l'une des deux autres interfaces.*

Les gestionnaires de modules sont des éléments clés de l'application coopérative. Comme nous l'avons expliqué, ils sont chargés d'interagir avec l'environnement de leur module respectif. Ils vont donc dépendre fortement de l'environnement du module, et sont donc d'un niveau assez bas. Ce sont les seules entités de l'architecture coopérative qui soient dépendantes de l'implémentation.

Gardons à l'esprit que nous faisons de la ré-ingénierie et donc nous ne pouvons pas forcément réaliser toujours ce que l'on voudrait. Ainsi, le gestionnaire de module va fortement dépendre du niveau d'intervention possible. Il ne sera pas toujours possible d'obtenir l'information exacte que l'on désire, néanmoins, on pourra peut-être en obtenir une autre suffisante s'y rapprochant.

Exemple . *Imaginons que nous désirons réaliser un comptage des colis passant sur les tapis roulants, mais que ceux-ci ne sont pas pourvus de compteurs. L'événement d'arrivée de colis ne peut donc pas être produit. Néanmoins, ces colis passent par la suite par les balances et les scanners de mesure de volume. Nous n'avons pas là l'information exacte désirée, mais une autre suffisamment proche.*

Nous savons maintenant comment récupérer les éléments de coopération, mais il nous manque encore quelque chose. Nous avons expliqué dans le chapitre précédent qu'il était nécessaire de qualifier les événements de coopération. Ce travail permet d'obtenir des précisions sur qui a produit l'élément de coopération ainsi que sur le ou les destinataires. Ce travail ne peut être fait que par le GM puisque c'est lui qui prélève les éléments de coopération de leur environnement. Ainsi, lorsqu'il détectera une information devant être signalée, il l'étiquettera avec sa source, c'est à dire le module émetteur ainsi que le groupe de travail dans lequel est incorporé ce module. Ceci sous entend que le gestionnaire de module soit au courant de l'état de l'application coopérative, c'est à dire qu'il sache dans quel groupe est inclus son module. Il consulte pour cela la base de donnée de gestion de la coopération : *BD-ÉTAT*.

Ainsi, lors de l'appel des primitives *ProduireÉvénement* et *ProduireMessage*, le GM complétera l'événement ou le message en le qualifiant. Nous aurons ainsi les primitives complètes, c'est à dire possédant une source et une destination :

ProduireÉvénement (nomÉvénement **De Module** *M* **Dans Groupe** *G*)
ProduireMessage(message **De Module** *M* **Dans Groupe** *G*)

2.3.2 Interventions indirectes

Nous allons maintenant proposer diverses solutions permettant de capturer et introduire des éléments de coopération dans les modules sans intervention directe sur ces derniers. Il s'agit de pouvoir, depuis l'extérieur, sans perturber le fonctionnement du module, se substituer à son environnement en ce qui concerne ses échanges avec celui-ci.

2.3.2.1 Capturer et produire des éléments de coopération pour des modules physiques

Dans le cas de modules physiques (automates, cellules de production, machines outils, cartes dédiées, …), le moyen le plus efficace consiste à « écouter/activer » leurs lignes de communication. Il ne s'agit pas de toutes les lignes mais uniquement de celles qui nous intéressent.

Par exemple, un module M_1 a besoin de savoir lorsqu'une cellule de production (M_2) émet une alarme. Parmi les lignes (et elles peuvent être très nombreuses) présentes dans cette cellule, l'une d'entre elles correspond à l'alarme. En la scrutant, il est possible de détecter le signal (il s'agit par exemple d'un

front montant destiné à allumer une lampe), et donc de réaliser la capture de l'information. Lorsque ce signal est détecté, le GM le transmet à la plate-forme coopérative en tant qu'élément de coopération, c'est ici typiquement un événement. Il est ensuite transmis au gestionnaire de module de M_1 qui se chargera de l'inclure dans l'environnement du module selon le format voulu par celui-ci.

Afin de clarifier ceci et de montrer que tout ceci est bien réalisable, nous allons nous appuyer sur un petit exemple :

Dans de très nombreux cas, les communications entre deux micro-ordinateurs, un micro-ordinateurs et un automate sont (encore) réalisés à l'aide d'un simple câble série. Sur le micro-ordinateurs, ce câble série est relié à un contrôleur de communication asynchrone. Nous n'allons pas entrer dans le détail des caractéristiques de ce contrôleur, mais nous allons tout de même montrer le schéma de relation entre un micro-ordinateurs et un automate :

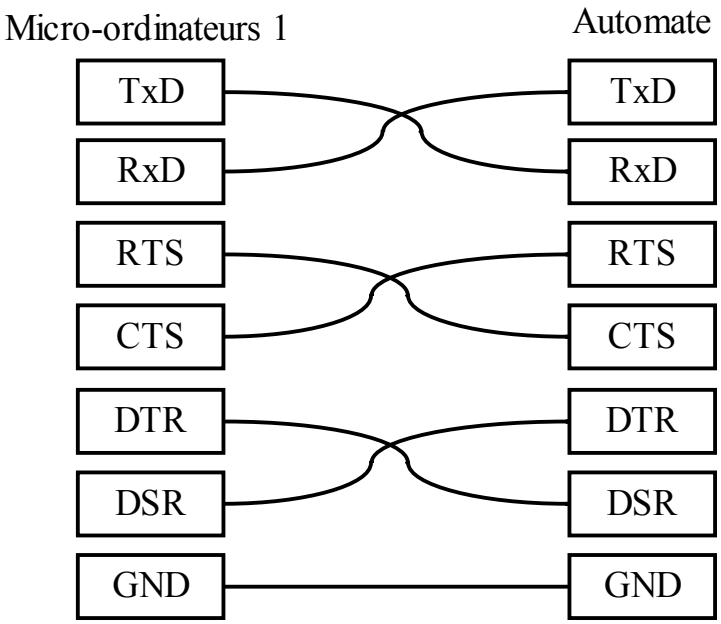


Figure 55 : Liaison physique du 82050

Sans entrer dans le détail, nous pouvons observer les liaisons RxD ⇔ TxD qui permettent aux données de transiter entre les deux contrôleurs. CTS ⇔ RTS permet de signaler si la machine reliée à l'autre bout du câble est prête à recevoir, et enfin, DSR ⇔ DTR qui signale si la machine distante est en service ou hors service, si elle est en état de fonctionner.

Imaginons que nous désirions connaître les informations émises par la l'automate afin de détecter par exemple une certaine séquence de caractères. Il suffira de réaliser une dérivation sur la broche correspondante puis de scruter ce qui se passe. Ce travail sous entend que nous sommes capable de connaître le protocole de communication entre les deux entités (vitesse, parité, bits stop, …).

Ainsi, une fois que la séquence recherchée est trouvée, le gestionnaire de module pourra réagir en conséquence en signalant l'élément de coopération correspondant (par exemple, un message contenant la séquence détectée).

Remarque : L'exemple présenté ici est une liaison série, mais cette technique pourrait tout aussi bien être utilisée avec les lignes d'E/S d'un automate.

2.3.2.2 Capturer et produire des éléments de coopération pour des modules logiciels

Le travail est plus complexe dans le cas des modules logiciels. En effet, un logiciel peut produire et récupérer des informations selon des moyens très variés comme nous l'avons déjà exprimé :

- Par variables partagées ou d'environnement,
- Par fichiers,
- Par bases de données,
- Par connexion réseau,
- Par action sur les interfaces,
- Par les périphériques de saisie et de sortie,

Le gestionnaire de module doit donc pouvoir être capable de capturer et de produire les informations sur l'ensemble de ces médiums.

Nous allons donc à présent montrer quelques solutions (l'éventail des solutions n'est pas exhaustif) adaptées à chaque type médium :

Variables partagées ou d'environnement

Bien que peu élégante, il existe une solution facilement réalisable qui est de scruter en permanence (le choix de la granularité de la vitesse de scrutage est laissée au soin du programmeur) cette variable, lorsqu'elle changera, il suffira de réagir en conséquence. La réaction peut se faire sur le simple fait que cette variable a été modifiée ou bien selon sa nouvelle valeur.

Fichiers

Par comparaison de dates, il est possible de savoir si un fichier a été modifié (dans le cas où la date est une précision suffisante, c'est à dire que l'unité de temps est suffisante), sinon, en réalisant une copie et en comparant ou encore par calcul de checksum, il est possible de savoir s'il a été ou non modifié et

donc de tirer des conclusions. Il est aussi possible de modifier les tables des appels systèmes afin par exemple d'être tenu au courant de chaque appel en lecture, mise à jour, … Prenons par exemple le système d'exploitation UNIX avec l'appel système *open* correspondant à l'ouverture d'un fichier. En faisant pointer l'entrée *_NR_open* (*_NR_nom_service*) de la table *sys_call_table* contenant les pointeurs des fonctions associées aux appels système sur une fonction de notre cru, il est possible d'être prévenu à chaque appel. A chaque appel, la fonction pourra par exemple mémoriser le nom du fichier pour lequel une demande d'ouverture a été réalisée, puis elle appellera réellement la fonction *open* classique afin de réaliser l'ouverture du fichier concerné, et ceci de manière entièrement transparente.

Ces propositions de solutions ne sont pas optimales et l'on pourra en développer de nombreuses autres plus appropriées selon les cas considérés.

Bases de données

Nous avons ici un point important. L'analyse différencie trois types d'éléments de coopération : événements, messages et données. Concernant ces dernières un traitement particulier a été prévu. En effet, les données ne sont pas des informations qui circulent réellement dans une application. Elles sont directement mises à jour dans le SI par les entités concernées.

Ce ne sont pas ces données là qui sont concernées ici puisqu'elles sont déjà partagées. Toutefois, certains modules peuvent gérer une BD locale ne faisant pas partie du SI de l'application et il est parfois souhaitable de pouvoir utiliser ces informations comme éléments de coopération.

Nous avons vu qu'ADACTIF, et plus particulièrement son gestionnaire de transaction, permet de transformer un SGBD « classique » en SGBD actif. Nous pouvons utiliser cette opportunité pour modifier le SGBD correspondant. Les événements d'accès à la BD détectés par ce SGBD devenu Actif seront ensuite transmis au GM qui en fera le tri (événement significatif ou non) et les qualifiera avant de les faire parvenir à la plate-forme coopérative.

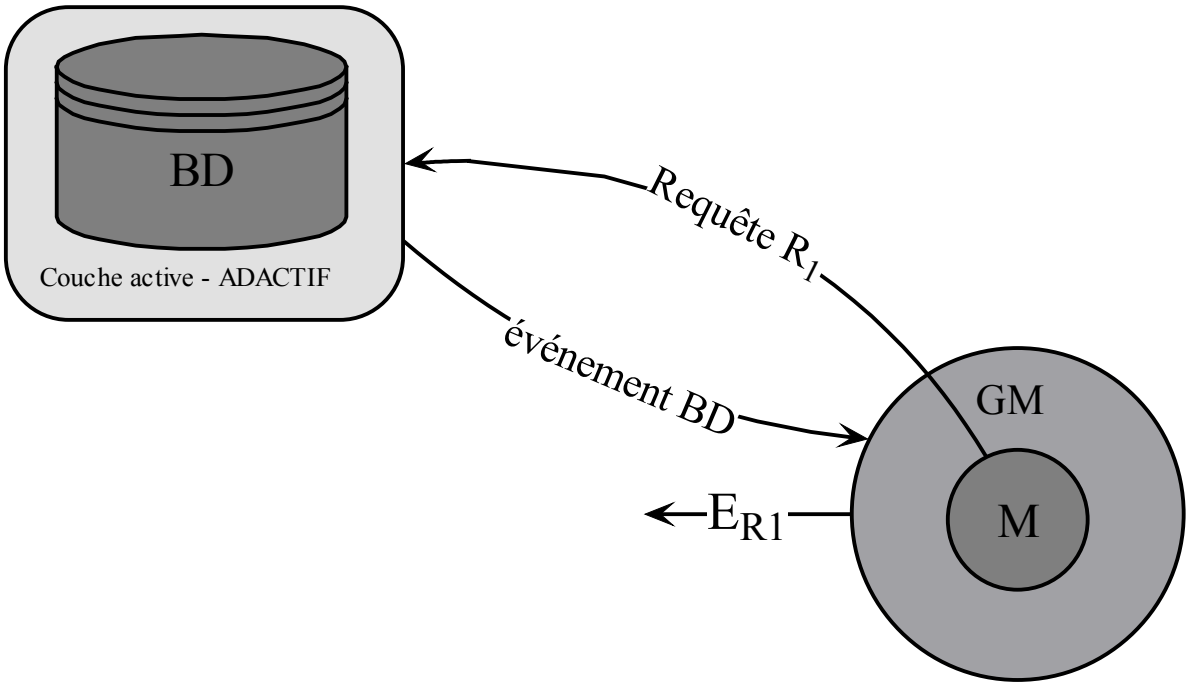


Figure 56 : Levée d'événements correspondants à des accès BD

Connexions réseau

Les techniques à utiliser ici sont un peu plus compliquées, mais sans pour autant être impossibles. En effet, il existe de nombreuses solutions permettant de récupérer des données transitant sur un réseau :

- L'utilisation d'une prise vampire branchée sur le câble réseau transmet les signaux qu'il faut alors interpréter.
- L'utilisation de logiciels appelés « sniffers » (renifleurs) qu'utilisent les administrateurs système permettent de pister et de récupérer certaines trames dont tout ou partie du contenu est identifié.
- À partir des commutateurs du réseau, il est possible de créer des réseaux virtuels (V-LAN - Virtual Local Area Network) autorisant le « port-mirroring », c'est à dire que tout ce qui est reçu sur un port est intégralement dupliqué sur un autre.
- Il est aussi possible d'intercaler un hub entre deux machines permettant de récupérer de manière totalement transparente l'intégralité de leurs échanges.

Interface

Si intervenir sur l'interface ne pose pas de problèmes majeurs (il est par exemple aisément possible de créer de nouvelles boîtes de dialogue), il est déjà beaucoup plus difficile, voire impossible de détecter un clic sur un bouton particulier, ou la mise à jour d'une valeur sans intervenir, même de façon

minime sur le code. Même si cela peut dans certain cas se réaliser (lorsque l'interface est ouverte), il est, dans le cas contraire, possible d'utiliser la technique de détournement des appels systèmes décrite pour la gestion des fichiers. On pourra par exemple détourner l'appel système correspondant au clic souris lorsque celui-ci concernera une fenêtre particulière.

Nous arrivons ici aux limites de la ré-ingénierie et nous sommes bien obligés de faire avec les contraintes et réalités de ce domaine. Ainsi, nous ne prétendons pas qu'il sera dans tous les cas possible de récupérer les informations désirées. Nous nous sommes déjà expliqués là dessus, mais dans le cas où une informations désirée ne pourrait pas être récupérée, il sera nécessaire d'en trouver une autre suffisamment proche pour la remplacer.

2.4 Le Gestionnaire de Communication

Comme nous l'avons déjà mentionné, le GC est la porte permettant de communiquer avec les autres sites et donc permettant de diffuser événements et messages aux autres plates-formes et donc physiquement aux autres sites. C'est donc par lui que transiteront tous les éléments de coopération nécessitant d'être diffusés sur le réseau, et c'est aussi par lui que passeront tous les éléments de coopération en provenance des autres sites.

Nous rappelons que tout élément de coopération diffusé (et donc envoyé et/ou reçu) est entièrement qualifié, c'est à dire qu'il possède un nom, un émetteur, et une destination.

Chaque site étant relié à tous les autres par l'intermédiaire d'un réseau, il est nécessaire d'avoir un GC sur chacun d'eux :

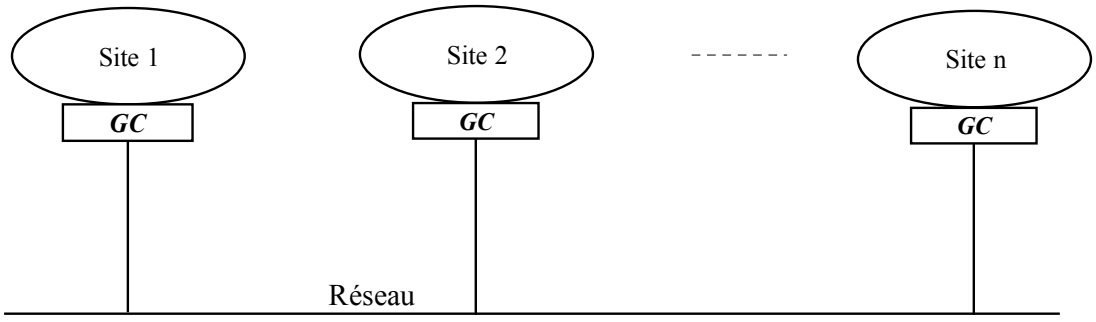


Figure 57 : Le gestionnaire de communication

Le GC permet d'associer des noms logiques (noms de groupes, noms de modules) à des adresses physiques (pouvant changer au cours de l'exécution). Lorsqu'un événement ou un message lui est

transmis par l'intermédiaire du $G_{ÉC}$ de son site, il consulte la BD-ÉTAT afin de connaître l'adresse physique de chacun des destinataires. Nous pouvons ainsi compléter la description de la structure de la BD-ÉTAT entamée précédemment par les champs nécessaires à l'identification physique de chaque module.

Lorsque le destinataire est un groupe, il se chargera de retrouver tous les modules appartenant à ce groupe afin de leur communiquer le message. On utilisera pour cela la primitive *membreDe(nomGroupe)*. Cette étape réalisée, il diffusera l'élément de coopération vers le ou les destinataires au travers des GC correspondants.

De manière symétrique, lorsqu'un événement ou un message arrive du réseau au GC, et donc envoyé par le GC du site émetteur, celui-ci le fait immédiatement transiter au $G_{ÉC}$ qui se charge de le traiter s'il est reconnu.

2.5 Les règles

Comme nous l'avons précisé, l'ensemble des gestionnaires présents dans le modèle d'activité sont des tâches et donc s'exécutent en parallèle. Les règles ne dérogent pas à cela et sont, elles aussi, des tâches. Au niveau de la plate-forme coopérative, chaque règle est un processus léger (*Thread*) Java entièrement autonome. La conséquence est une exécution entièrement parallèle à la fois des différents gestionnaires, mais aussi de l'ensemble des règles, qu'elles soient ÉCA ou détectrices.

D'un point de vue de la localisation, pour chaque module, nous avons autant de règles ÉCA de gestion de la dynamique que de groupes auxquels il peut appartenir. Ces règles seront présentes sur le site sur lequel est identifié le module. Là aussi, nous faisons intervenir la BD-ÉTAT qui possède en permanence la liste des règles identifiées, les informations devant leur être signalés, l'élément de coopération qu'elles signalent et enfin leur localisation physique. Dans le cas où une règle détectrice réalise une composition à partir de plusieurs informations distantes, elle sera située sur le site où l'information qu'elle produit est utilisée. Dans le cas où cette même information est nécessaire sur plusieurs sites, elle sera dupliquée sur l'ensemble des sites concernés.

D'un point de vue l'implémentation, les règles ÉCA s'exécutent à chaque fois qu'elles reçoivent leur événement déclenchant. Une fois la condition validée et l'action réalisée, elle disparaissent de l'architecture coopérative. Il est à noter que, pour un même événement, plusieurs règles ÉCA différentes peuvent être déclenchées. Inversement, les règles détectrices sont présentes tout au long de l'exécution de l'application. Lorsqu'un élément entrant dans leur composition est produit, il leur est signalé, à chaque fois que l'algorithme de composition est validé, l'élément de coopération identifié pour cette règle est produit.

Puisque les règles sont des tâches elles doivent avoir une interface ainsi qu'un corps. Nous avons :

RègleÉCA ECA_1 {

Interface :

- Événement déclencheur

Corps :

- Évaluation et entrée/sortie éventuelle de groupe du module correspondant.

}

RègleDétectrice RD_1 {

Interface :

- Ensemble des éléments de coopération entrant dans la composition

Corps :

- Évaluation de la composition
- Production de l'élément de coopération composé : événement, message ou donnée

}

Nous ne reviendrons pas sur les caractéristiques des règles que nous avons déjà longuement décrites mais allons plutôt nous attacher à expliquer leur manière de fonctionner dans le cadre de la plate-forme coopérative.

Remarque : lorsque le rôle de la règle détectrice est de réaliser la traduction, la mise en forme ou bien de compléter un message, elle recevra l'événement correspondant à l'arrivée d'un message et devra consulter la BAL du $G_{ÉC}$ afin de le récupérer.

2.5.1 Les règles ÉCA

Contrairement aux règles détectrices qui produisent un élément de coopération en réponse à une composition d'éléments de coopération, les règles ÉCA n'acceptent qu'un événement en entrée et sont utilisées dans un but unique: gérer la dynamique des groupes.

Les règles ECA ont un rôle déterminant au sein de l'application coopérative. Ce sont elles qui font entrer un module dans un groupe ou bien l'en font sortir en fonction de l'événement qu'elles reçoivent et de la condition qu'elles évaluent. Afin de réaliser cela, elles disposent de primitives spécifiques comme nous l'avons déjà vu.

La levée de l'événement déclenchant doit provoquer l'évaluation de la condition d'entrée/sortie de groupe pour un module. Ainsi, lors de la levée de cet événement, la plate-forme coopérative (et plus précisément le GR) va prendre rendez-vous avec la règle ÉCA correspondante. Une fois le rendez-vous accepté et si la condition est vérifiée, la règle réalisera l'entrée ou la sortie de groupe du module. Pour cela, elle mettra à jour la BD-ÉTAT avec les nouvelles données. Ainsi, ce module ne recevra plus les informations du groupe qu'il vient de quitter ou recevra celles du groupe auquel il vient d'adhérer.

Comme nous l'avons expliqué au paragraphe précédent, pour chaque module dynamique, nous aurons autant de règles de gestion de la dynamique que de groupes auquel il peut appartenir. Le changement des groupes réalisé par chacune des règles sera fait en utilisant les primitives *joindreGroupe(nomGroupe, nomModule)* et *quitterGroupe(nomGroupe, nomModule)* précédemment décrites.

2.5.2 Les règles détectrices

Pour résumer, le but d'une règle détectrice est d'être réceptive à un ensemble d'éléments de coopération. Ce sont les informations nécessaires à l'évaluation de l'algorithme de composition.

Son rôle est d'obtenir des informations avec une sémantique plus riche que celles à partir desquelles la composition est réalisée. Elle permet, par exemple, à partir de plusieurs messages d'en n'en obtenir qu'un faisant la synthèse des autres ou bien ne contenant que les parties nécessaires à un autre module/règle.

Ces règles sont actives en permanence et acceptent dans une boucle infinie les rendez-vous avec le GR correspondant aux éléments de coopération à composer. Elles peuvent accéder au système d'information de l'application, à la BAL pour les messages et à la BD-ÉTAT pour contrôler l'état de la coopération, de sorte que l'algorithme de composition permette de tenir compte de tous les éléments utiles.

2.6 Cheminement des éléments de coopération

Ce paragraphe va présenter un résumé sur la manière dont circulent les éléments de coopération et comment est réalisé le lien entre le langage de spécification et la plate-forme coopérative.

Voici deux extraits de spécifications sur lesquelles nous allons nous appuyer :

- **DéfinirÉvénement** E_{PA} **De Module** IHM_1 **Dans Groupe** *Expédition Régionale* **Pour Règle** RD_{PR}
- **DéfinirÉvénement** $E_{C'}$ **Pour Module** *B* **Dans Groupe** *Service Commercial* **as** E_{PA} **De Module** IHM_1 **Dans Groupe** *Expédition Nationale*

Lorsque le module IHM_1 lève l'événement E_{PA}, le GM le récupère et vérifie qu'il est bien dans le groupe *Expédition Régionale* (primitive *estDansGroupe(« IHM_1 »)*). Dans la négative, rien n'est fait. Si le groupe de travail est bien *Expédition Régionale,* il va étiqueter l'événement avec les informations suivantes : **De Module** IHM_1 **Dans Groupe** *Expédition Régionale*, et l'envoyer au $G_{ÉC}$.

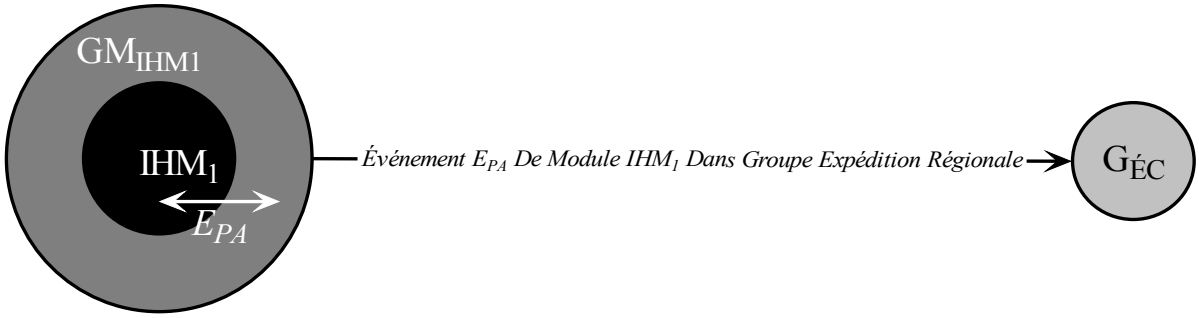


Figure 58 : Notification capture et qualification partielle d'un événement

Le $G_{ÉC}$ va ensuite consulter sa base pour savoir quelles sont entités (règles et modules) concernées par cet événement. Il va trouver la règle RD_{PR} qui est locale et le module B situé dans le groupe Service Commercial. E_{PA} sera donc transmis au GR. Il sera aussi transmis au GC après complétion de la qualification, c'est à dire :

- **Événement** E_{PA} **De Module** IHM_1 **Dans Groupe** *Expédition Régionale* **Pour Module** *B* **Dans Groupe** *Service Commercial*

A l'aide de ces informations, le GC consultera la BD-ÉTAT. Si le module B est effectivement dans le groupe *Service Commercial*, alors il récupérera l'adresse physique du site où est hébergé B puis transmettra l'événement sous le nom de $E_{C'}$.

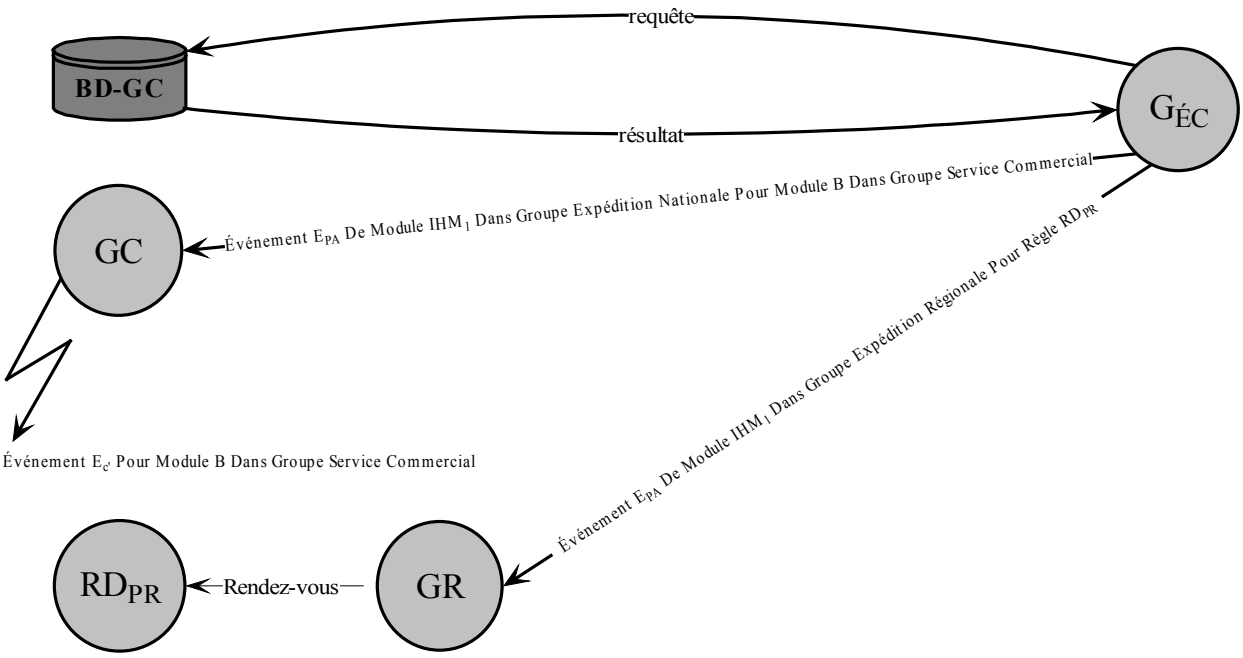


Figure 59 : Aiguillage des éléments de coopération par le $G_{ÉC}$

Une fois que l'événement est localement reçu par le GR (site 1), celui-ci va consulter la BD-ÉTAT pour connaître les règles pour lesquelles cet événement est une entrée. Il va trouver la règle RD_{PR} (c'est une règle détectrice) et prendre rendez-vous avec elle.

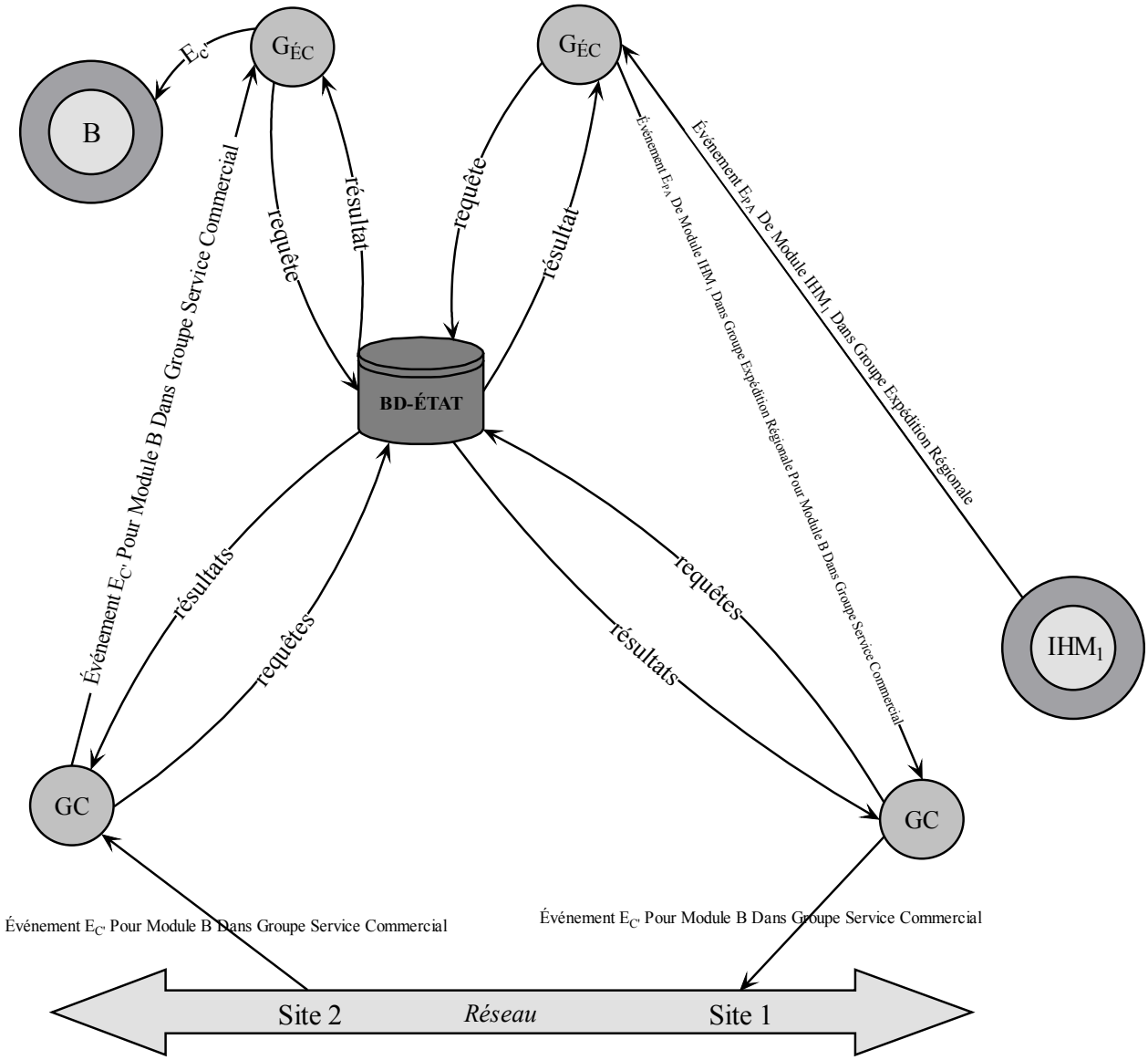


Figure 60 : Notification d'événements au travers du réseau

A l'autre bout du réseau, sur le site où est physiquement présent le module B (site B), le GC va recevoir l'événement $E_{C'}$. Une fois reçu, il va le transmettre à son $G_{ÉC}$ (**Événement** $E_{C'}$ **Pour Module** *B* **Dans Groupe** *Service Commercial).* Celui-ci va le transmettre au GM qui va le mettre selon une forme compréhensible par le module puis va l'incorporer dans son environnement de manière à ce que le module puisse en tenir compte.

3. Le prototype

Nous avons maintenant toutes les informations nécessaires à l'implémentation de notre prototype. Nous avons vu comment nous pouvions passer de l'analyse au langage de spécification puis, du

langage de spécification à la dérivation en règles pour la plate-forme coopérative. Nous avons décrit celle-ci en expliquant pour chacun de ses composants quels étaient leur rôle et comment ils communiquaient entre eux. Il nous reste à voir l'ultime étape de nos travaux : l'implémentation du prototype.

Cette partie va débuter avec un rappel et quelques précisions sur le middleware ainsi que sur les outils et techniques récemment apparus. Nous insisterons un peu sur ce thème puisqu'au delà de la plate-forme coopérative que nous avons décrite, l'ensemble des échanges entre les gestionnaires, et de manière plus large, entre les différentes entités simulées seront réalisées en utilisant des techniques et outils que nous allons voir.

3.1 Présentation générale

L'objectif de notre prototype n'est pas de coller parfaitement à la réalité mais de valider nos propositions. Néanmoins, dans un souci d'approche globale, en plus de la plate-forme coopérative, nous allons nous attacher à simuler l'exemple développé au cours de l'analyse en veillant à ne reprendre que des éléments mis en évidence dans cette étude. Nous allons particulièrement centrer l'élaboration du prototype sur :

- La réalisation des interfaces graphiques simulant les IHM, les arrivées et la sélection de colis. C'est l'aspect le plus visible de la simulation.
- La mise en réseau afin de permettre aux gestionnaires de communication de diffuser et de récupérer les informations ; elle permettra mais aussi aux différents composants ($G_{ÉC}$, GR, …) de communiquer entre eux. C'est le fonctionnement même de l'architecture coopérative qui se trouve ici réalisé. Nous proposerons un moyen de suivre le cheminement des éléments de coopération et montrerons aussi les différents gestionnaires en action au fur et à mesure que transitent ces informations.

Afin de programmer cela, nous avons décidé d'utiliser deux outils particuliers qui, bien qu'en pleine évolution, ont fait leurs preuves et sont porteurs de nombreux concepts intéressants : JAVA et CORBA.

Java est un langage maintenant connu de tous, nous n'allons pas le décrire ici, le lecteur pourra se référer à la littérature très abondante sur ce sujet. Puisque ce langage nous fournit de nombreuses API (*Application Programming Interface*) graphiques, il nous sera très utile pour réaliser aisément les interfaces avec l'utilisateur. Outre ces aspects purement graphiques, les aspects réseaux sont

particulièrement bien intégrés, ce qui facilite encore plus son utilisation pour la mise en œuvre de la plate-forme coopérative. Conjointement aux capacités de Java, nous utiliserons Corba.

CORBA fait partie de ces outils appelés middleware (« *classe de logiciels qui assurent l'intermédiaire entre les applications et le transport des données par les Réseaux* » [Trique 99]). C'est un outil idéal à qui veut relier entre elles des entités distribuées sur un réseau. Concrètement pour nous, il servira à relier les différents gestionnaires (gestionnaire d'éléments de coopération, de communication, …) mais aussi les règles.

Tout comme Java, Corba est en plein développement actuellement ce qui fait que ses spécifications sont en évolution et peuvent donc ne plus être totalement conformes au moment de la publication de cette thèse (de nouveaux services étant régulièrement ajoutés). On se référera à www.omg.org pour avoir les dernières mises à jour.

Corba est aussi fortement lié aux composants, domaine duquel nous ne sommes pas très éloignés en fin de compte. Afin d'être le plus complet possible, nous ferons aussi une brève présentation de la notion de composant logiciel mais sans pour autant entrer dans le détail de ce vaste et intéressant domaine.

3.2 Outils utilisés

Le point sensible de notre prototype est la partie communication. Afin de la réaliser du mieux possible, nous nous sommes appuyés sur des techniques et des outils récents qui ont fait leur preuves depuis quelques années. C'est ainsi qu'il a fallu nous pencher sur le middleware et plus particulièrement sur CORBA ainsi que sur les composants logiciels et les communications entre composants.

3.2.1 Rôle du middleware

Le développement de l'informatique distribuée entraîne la connexion sur les réseaux de machines et de systèmes d'exploitation différents. Il faut pouvoir faire appel, à distance, aux fonctions et aux données de n'importe quelle application, sur n'importe quelle machine : c'est l'objectif du middleware. Avec l'avènement des systèmes ouverts et du client-serveur, le middleware devient une pièce maîtresse pour gérer la complexité et l'hétérogénéité des logiciels en fournissant des interfaces ou des services de base qui assemblent et donnent une unité aux différents composants d'une architecture de ce type.

Sur le plan théorique, le mode de fonctionnement est le suivant : l'application sur le poste client nécessite des données ou des services qui sont hébergés sur le réseau, souvent sur des serveurs qui fonctionnent avec des systèmes d'exploitation différents, avec des bases de données et des langages

spécifiques. L'application *cliente* à besoin d'un middleware qui cherche ces données ou ces services, pour les lui retransmettre après les avoir re-configurées au format désiré.

3.2.2 Les services d'un middleware

Définition :

Le middleware est en général une couche logicielle (une API) qui joue le rôle d'interface entre le client et le serveur ou entre serveur et serveur. Il est indépendant des logiciels de communication et de gestion de réseaux [Evariste 99].

L'application est constituée de deux parties qui communiquent en utilisant des interfaces (*API*) non nécessairement identiques et pouvant être ouvertes. Le middleware masque l'existence et le fonctionnement du protocole de transport (par exemple TCP/IP).

Selon le cabinet anglais *Ovum* [Ovum 99], les applications de type middleware se décomposent en quatre familles de produits :

1. Les logiciels d'accès aux données. C'est l'application du middleware aux bases de données (relationnelles). La plupart des applications en client-serveur nécessitent que le client puisse accéder aux données d'une base relationnelle sur un serveur.
2. Les logiciels gérant les procédures d'appel à distance (*RPC* ou *Remote Procedure Call*). Il s'agit de l'accès distant de client à serveur, mais non dédié à une base de données particulière. Ceci permet la mise en œuvre d'environnements client-serveur plus évolués, avec un accès possible à des bases de données quelconques. Dans ces cas de figure (comme dans de nombreux autres), le client et le serveur doivent utiliser le même protocole de communication. Lorsque la transaction est lancée, le client se met en attente de la réponse du serveur devenant ainsi inactif tant qu'il n'a rien reçu.
3. Les produits de messagerie applicative. Ils permettent de rendre asynchrone les communications entre client et serveur, c'est à dire, ils permettent de supprimer la phase d'attente (et donc d'inactivité) du client. Les messages peuvent être traités en temps réel ou stockés et envoyés en temps voulu. Les messageries middleware permettent des échanges dans des environnements distribués « plus hétérogènes ».
4. Les services d'échange entre objets : les ORB (*Object Request Broker*) organisent la circulation des messages entre objets (applicatifs). Contrairement au RPC et à la messagerie, les ORB peuvent contenir des informations plus complexes. L'approche objet s'appuie sur le standard CORBA (de l'*OMG* ou *Object Management Groupe*) et sur le langage IDL (*Interface Definition Language*) ainsi que sur une API permettant l'envoi et la réception des objets sur le réseau.

Les logiciels de middleware servent à la fois d'intermédiaires entre des applications logicielles différentes, et de tampon entre le logiciel d'application et l'architecture du réseau qu'ils essaient de rendre transparente. Ils doivent enfin permettre de rajouter de nouveaux services sans remettre en cause l'existant, tant au niveau matériel que logiciel et doivent garantir l'évolutivité du système d'information.

3.2.3 Les composants logiciels

Les composants logiciels sont des instances de classes à la manière de ce qui se fait dans les modèles à base d'objets. Ainsi, une classe de composants est définie par des fonctionnalités et une interface (permettant d'accéder aux fonctionnalités).

L'interface est composée de différents éléments :

- Les services. Ce sont des fonctions (ou procédures) fournies ou requises par le composant. Afin qu'il puisse fonctionner. C'est l'équivalent des méthodes dans les modèles à objets.
- Les notifications. Ce sont des événements levés par le composant vers l'extérieur.
- Les attributs. Ce sont des variables typées publiques accessibles depuis les autres composants.

Il est possible de réaliser des collections (ensemble nommé d'instances d'un même composant). Cette possibilité permet de décrire la dynamique de la structure interne d'un composant. A l'exécution, la structure interne d'un composant varie selon la création/destruction d'instances de composants dans une collection ainsi que l'ajout/retrait d'interactions entre ces instances.

Les composants logiciels réalisent des interactions avec d'autres composants. Pour cela, ils utilisent des connecteurs qui sont des entités décrivant les interactions entre composants et donc permettant d'établir un lien de communication entre ces composants. La description d'un connecteur permet de communiquer de manière conforme, c'est à dire, permet la définition d'un protocole de communication (communication synchrone ou asynchrone, diffusion d'événements ou appel de procédures distantes). Cette description permet en outre de spécifier le comportement sous forme d'un ensemble de contraintes décrivant la qualité de service attendue. On pourra se référer par exemple à [Allen 94] pour de plus amples informations sur les connecteurs.

3.2.4 Intégration des composants logiciels

Les modèles à base d'objets sont adaptés à l'encapsulation de logiciels existants. Ils permettent de décrire l'interface des composants logiciels de manière totalement indépendante de l'implémentation.

On peut distinguer deux approches majeures dans l'intégration de composants logiciels. Les courtiers d'objet et les bus logiciels. Nous allons ici réaliser une brève description de chacune de ces approches.

3.2.4.1 Bus de messages

Un bus logiciel met à disposition un service de communication de type asynchrone basé sur les événements ou l'échange de messages.

Les interactions entre composants sont réalisées par un service particulier (le service de communication) qui met en relation les composants émetteurs et récepteurs.

3.2.4.2 Courtier d'objets

Les courtiers d'objets (*object brokers*) sont basés sur un modèle d'interaction de type client-serveur et sur l'utilisation de langages de description d'interfaces (IDL). Cette approche est maintenant largement développée en particulier grâce aux travaux de l'OMG autour de la norme CORBA (décrite en détail ci-après).

Microsoft développe de son coté une approche similaire avec son modèle COM (Component Object Model) et DCOM (Distributed COM) [Microsoft 94].

La limitation principale de l'approche par courtiers d'objets est qu'elle repose en fait sur le modèle client-serveur qui atteint ses limites pour décrire des schémas de coopération plus élaborés. Des services complémentaires (contrôle de la concurrence, communication par événement, …) sont en cours d'élaboration.

Les deux approches décrites ci-dessus sont des outils permettant de mettre en place des interactions élémentaires entre des groupes isolés de composants. Néanmoins ils ne permettent pas de réaliser l'ensemble de l'organisation d'une application, ni en terme de composants logiciels, ni en terme de schémas d'interactions complexes entre ces composants. Néanmoins, ce sont des outils très intéressants et en constante évolution et méritant toute notre attention.

3.2.5 CORBA

L'Object Management Groupe (OMG) est un consortium international regroupant plus de 850 « grands » du monde informatique, avec entre autres IBM et Sun pour le matériel, Netscape et Inprise

pour les logiciels, Boeing et Alcatel pour les utilisateurs et enfin la NASA, l'INRIA, et le LIFL pour les institutions et les universités. L'objectif de ce groupe, créé en 1989, est la mise en œuvre de standards pour l'intégration d'applications distribuées hétérogènes à partir de technologies orientées objet.

L'OMG met en avant des concepts clés qui sont la réutilisabilité, l'intéropérabilité et enfin la portabilité de composants logiciels. L'OMG est surtout connu pour son middleware orienté objet : CORBA (*Common Object Request Broker Architecture*) permettant la réalisation d'applications réparties hétérogènes en facilitant les communications entre composants logiciels [Geib 97].

3.2.5.1 Introduction à CORBA

Le bus CORBA est un modèle orienté objet client/serveur permettant l'abstraction et la coopération entre applications réparties. Les applications peuvent exporter certaines fonctionnalités (services) sous la forme d'objets CORBA (partie abstraction du modèle). Ainsi, les interactions entre les applications sont réalisées par l'invocation à distance de méthodes d'objets (coopération).

On retrouve ici la notion de modèle objet client/serveur. En effet, l'application implémentant l'objet demandé est le serveur, celle voulant l'utiliser est le client. Il est bien évidemment possible à une application d'être à la fois client et serveur.

3.2.5.2 Le modèle objet Client/Serveur

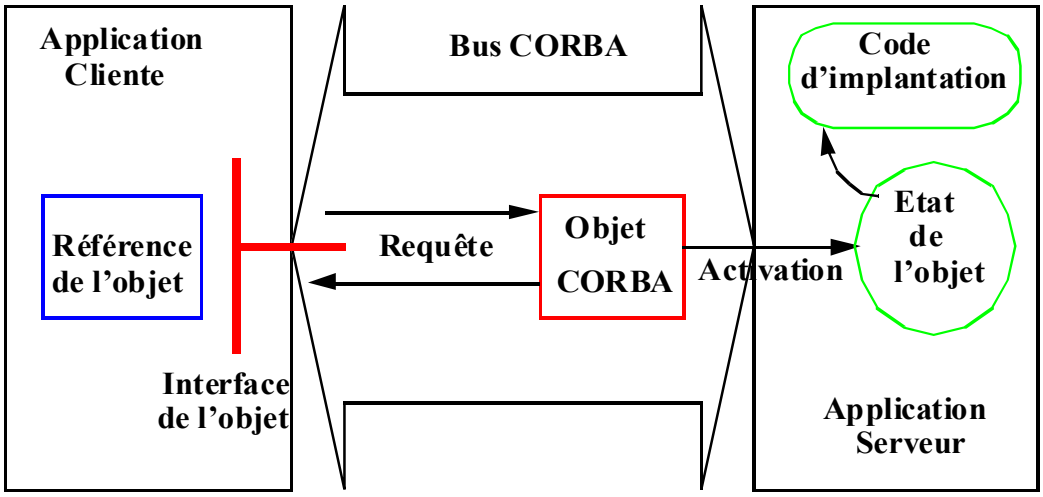


Figure 61 : Notions intervenant dans le modèle objet client/serveur [Geib 97]

Dans ce schéma, nous pouvons identifier les 10 éléments suivants :

1. *L'application cliente*. C'est un programme invoquant les méthodes des objets au travers du bus CORBA.

2. *La référence de l'objet*. Elle permet de désigner l'objet CORBA tout en contenant les informations nécessaires à sa localisation sur le bus.

3. *L'interface de l'objet*. Elle représente le type abstrait de l'objet CORBA (méthodes, attributs) et est décrite à l'aide du langage IDL.

4. *La requête*. C'est le mécanisme permettant l'invocation d'une méthode ou l'accès à un attribut de l'objet.

5. *Le bus CORBA*. Il permet d'acheminer les requêtes de l'application cliente vers les objets CORBA (sans que le client ait à gérer les problèmes de langage, de système d'exploitation, de matériel, de réseau, …).

6. *L'objet CORBA*. C'est une entité virtuelle gérée par le bus CORBA. Il représente un composant logiciel cible.

7. *L'activation*. C'est le mécanisme permettant d'associer un objet CORBA à l'objet implémenté.

8. *L'implémentation de l'objet*. C'est l'entité codant l'objet CORBA et gérant un état temporaire de l'objet. Il est à noter qu'au cours d'une exécution, un objet CORBA peut se voir assigner différentes implémentations.

9. *Le code d'implantation*. Il regroupe les traitements associés aux opérations de l'objet CORBA (par exemple, une classe Java/C++, des fonctions C, …).

10. *L'application serveur*. Elle représente la structure d'accueil de l'implémentation et de l'exécution des objets.

3.2.5.3 Le bus CORBA

Le bus d'objets répartis est la base de CORBA. Son rôle est d'assurer l'acheminement des diverses requêtes réalisées entre tous les objets CORBA. Il représente ainsi l'intermédiaire au travers duquel les objets vont communiquer.

Le bus permet donc de mettre en liaison des objets. Ceux-ci peuvent (pour l'instant) être réalisés en C, C++, SmallTalk, Ada, Cobol et Java. L'OMG travaille actuellement à la prise en compte d'autres langages comme Lisp.

Un avantage indéniable est la transparence des appels. Les requêtes sont réalisées comme si elles étaient locales, le bus se chargeant de les acheminer. Pour de ne pas surcharger le système, les objets sont gérés de manière dynamique n'étant en mémoire que s'ils sont utilisés par des applications clientes. Enfin, afin d'améliorer la portabilité, les objets dont décrits à l'aide d'un même langage de description (IDL). C'est au travers de cette interface qu'il est fait appel aux objets.

Enfin, depuis la norme CORBA 2.0 [OMG 94], il est possible d'interconnecter des bus CORBA provenant de fournisseurs différents par le biais de protocoles génériques comme GIOP (*General-ORB Protocol*) ou sa version Internet IIOP (*Internet Inter-ORB Protocol*).

3.2.5.4 La communication dans CORBA

Les applications clientes nécessitant d'être informées de modifications utilisent généralement la technique du « *polling* », c'est à dire qu'elles scrutent à intervalle régulier le serveur (changement d'état, ajout, modification, …). Si la période est trop courte, il y a un encombrement inutile du réseau. Dans le cas contraire, cela peut entraîner des problèmes d'incohérence entre l'état du serveur et les connaissances du client. Ainsi, afin d'éliminer ces problèmes, CORBA met à disposition un ensemble de services [OMG 98] :

- Le service d'*événements* (*Event Service*) permet d'envoyer des événements à destination d'objets par le biais de canaux d'événements. Deux modes sont possibles :
    1. Le mode « *push* ». L'envoi de l'événement est à l'initiative du producteur. Lorsqu'il est envoyé, il est notifié au consommateur.
    2. Le mode « *pull* ». L'envoi de l'événement est à l'initiative du consommateur qui réalise une demande explicite en sollicitant le producteur.
- Le service de *notification* (*Notification Service*) est en fait une extension du service d'événements. Il permet, à l'aide de filtres, de ne notifier au consommateur que les événements intéressants. Ce service limite le trafic induit par la propagation des événements.
- Le service *messagerie* (*CORBA Messaging*) permet la gestion de requêtes persistantes, c'est à dire que les objets appelants et appelés n'ont pas à être présents simultanément sur le bus.

3.2.5.5 Gestion du temps

Il est un autre service très intéressant dans un contexte de communication et de coopération qu'il est utile de mentionner, c'est le service *temps* (*Time Service*). Il fournit des interfaces permettant d'avoir en permanence sur le bus une horloge globale facilitant ainsi par exemple la synchronisation d'objets ainsi qu'une mesure de temps.

Ce service permet à l'utilisateur d'obtenir l'heure actuelle avec une estimation d'erreur associée. Il permet de plus de fournir un certain nombre de facilités comme :

- S'assurer de l'ordre dans lequel les événements sont intervenus.
- Générer des événements basés sur le temps calés sur des horloges ou des alarmes.

- Calculer l'intervalle entre deux événements.

Ce service est basé sur la représentation de l'UTC (*Universal Time Coordination*) à partir du service *X/Open DCE Time Service* et possède les caractéristiques suivantes :

- Unité de temps : 100 nanosecondes (10^{-7} secondes).
- Date/Temps de référence : 15 octobre 1582 00:00:00.
- Intervalle de temps supporté : +/- 30 000 ans.

3.3 Implémentation de la plate-forme coopérative

Comme nous l'avons déjà annoncé, nous avons décidé d'utiliser le langage Java. Nous n'allons pas revenir sur les avantages propres à ce langage (portable, réseau, prototypage aisé, …), puisqu'au delà de ses caractéristiques internes, ce qui a pesé dans notre choix a été la possibilité d'intégrer les dernières avancées en terme de middleware ainsi que de développer des composants logiciels (JavaBeans). La portabilité et l'intégration future dans un cadre industriel s'en trouveront ainsi grandement facilitées.

Le GC, $G_{ÉC}$, GR, GM ainsi que l'ensemble des règles sont des composants logiciels pour lesquels nous avons prévu des moyens de communication évolués. Dans notre prototype, de manière à suivre le fonctionnement de chacun, chaque gestionnaire, chaque règle se déclenchant (ou en cours d'exécution) se voit représentée à l'écran par une fenêtre.

Lorsqu'un composant réalise une action quelconque, celle-ci se matérialise à l'écran par un message dans la fenêtre associée. Ainsi, par exemple, chaque fois que le GR prend rendez-vous avec une règle, nous verrons un message s'afficher dans sa fenêtre. Dans le cas présent, il indiquera l'information reçue et quelle est la règle avec laquelle il prend rendez-vous. Si la règle n'était pas active (cas d'une règle ÉCA), nous verrons apparaître une nouvelle fenêtre contenant la description de son exécution : évaluation de sa condition, exécution de l'action.

3.3.1 *Le Gestionnaire de Modules*

Le gestionnaire de modules est un composant plus délicat à gérer. En effet, ce gestionnaire est le seul qui soit réellement dépendant de l'implémentation du module. Ainsi, le composant réalisé pour le GM possède l'ensemble des caractéristiques permettant de communiquer avec les autres, c'est à dire signaler l'occurrence d'un événement, l'envoi d'un message, l'accès au système d'information, mais il ne possède aucune caractéristique propre au module. Ce travail d'interaction avec le module ne pourra

se faire que lors de la mise en place de l'architecture et, contrairement aux autres gestionnaires, sera très étroitement lié au module concerné.

3.3.2 Le Gestionnaire d'Éléments de Coopération

Le $G_{ÉC}$ récupère dans la BD-ÉTAT l'ensemble des caractéristiques de chacun des éléments de coopération (source, destination).

Lorsqu'une information lui est transmise soit par le GC, le GM ou directement par une règle il regarde si elle nécessite d'être transmise au GC, à un GM ou au GR. Dans le cas où une diffusion est nécessaire, le $G_{ÉC}$ la signale au GC qui l'envoie via le bus de communication à destination des entités cibles (les GC des autres sites où cette information doit être reçue).

Sans entrer dans le détail du code, voici l'algorithme du $G_{ÉC}$:

```
Répéter indéfiniment {
        Attendre élément de coopération
        Déclencher un Thread de gestion de cet ÉC
}
```

```
Thread ÉC {
        L'élément reçu est-il à gérer ?
        Si oui, le transmettre aux entités concernées via le bus de communication
}
```

3.3.3 Le gestionnaire de Règles

Ce gestionnaire possède une table lui permettant de réaliser les correspondances – Élément de coopération reçu – Règles concernées. Ainsi, à chaque fois qu'une information lui arrive, il va consulter cette table afin de regarder premièrement si des règles détectrices sont destinatrices et dans l'affirmative, leur communiquer l'information via le bus de communication. Dans le même temps, il regarde si un rendez-vous doit être pris avec des règles ÉCA. Dans l'affirmative, il les déclenchera et prendra un rendez-vous avec elles afin qu'elles réalisent le traitement nécessaire lorsque la condition est vérifiée.

Nous abordons ici un problème que nous n'avions pas encore vu auparavant, ainsi, dans le squelette de chacune des règles ECA, une primitive *exécutionAchevée(nomRègle)* est automatiquement appelée avant sa terminaison. Cet appel permettra au GR d'avoir en permanence l'état d'activation des règles.

L'algorithme du GR est le suivant :

```
Répéter indéfiniment {
        Attendre élément de coopération
        Si élément de coopération pour règle détectrice alors
                Transmettre l'élément de coopération à l'ensemble des règles
                concernées
        Si élément de coopération pour règle ECA alors
                Activer autant de Threads ÉC qu'il y a de règles ÉCA à déclencher.
}
```

```
Thread ÉC {
        Prendre rendez-vous
        Attendre fin de règle
}
```

3.3.4 Le Gestionnaire de Communication

Le gestionnaire de communication est la seule entité « tournée » vers les autres sites. Il existe différent moyens de l'implémenter :

- Le GC récupère dans la BD-ÉTAT les références de l'ensemble des entités de l'application coopérative ainsi que leur localisation physique, ce qui lui permettra de savoir où envoyer les éléments de coopération. Afin de respecter les recommandations réalisées auparavant lors de la description du modèle d'exécution, c'est la solution que nous avons retenu.
- Le GC ne s'occupe que des modules du site dont il a la charge. Lorsque l'information doit être diffusée, elle passe sur un bus de communication sur lequel sont reliés l'ensemble des GC.

L'implémentation des règles ne différant que très peu des squelettes déjà proposés, nous n'allons pas ici en faire une description plus précise.

3.3.5 Le bus de communication

Le schéma concernant du modèle d'exécution d'ELKAR présente une architecture maillée dans laquelle de nombreuses entités différentes communiquent entre elles. Nous aurions pu gérer les communications de chaque site ainsi que les communications entre GC de manière que nous allons qualifier de « classique », c'est à dire chaque entité étant à la fois *cliente* lorsqu'elle nécessite d'envoyer une information, et *serveur* lorsqu'elle est en attente d'information.

Nous avons choisi une autre approche qui s'appuie sur CORBA en utilisant un bus de messages. Ainsi, d'une architecture maillée, nous passons localement à une architecture de type bus.

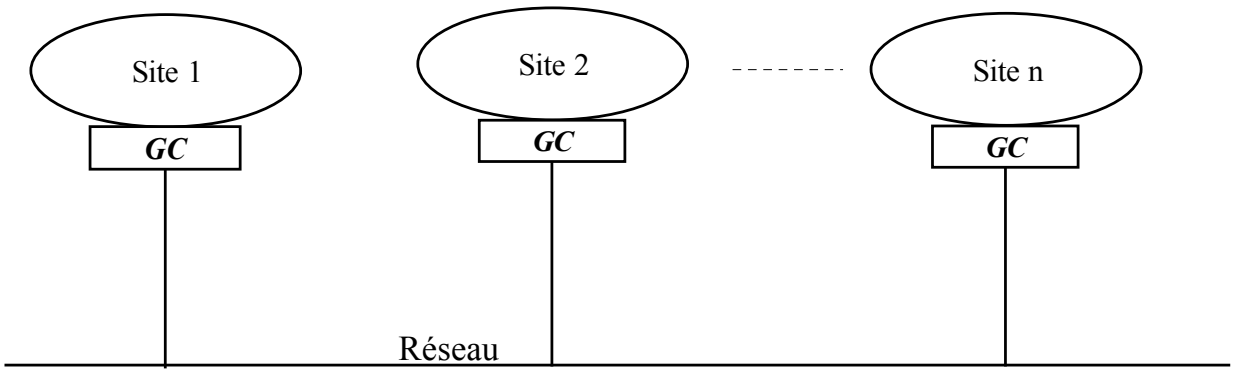


Figure 62 : Architecture réseau en Bus

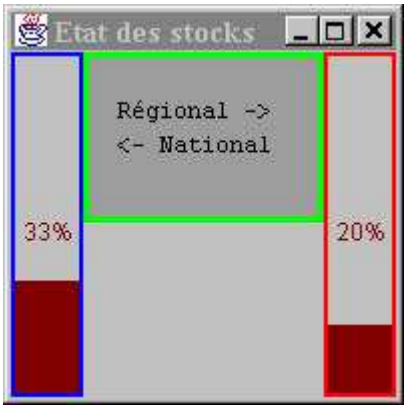
Les bases de l'implémentation de l'architecture coopérative étant maintenant posées, nous pouvons nous intéresser à la réalisation du prototype correspondant à l'exemple.

3.4 Application à l'exemple

Nous allons centrer la simulation sur les modules d'interface (IHM) mis en évidence au cours de l'analyse. Nous rappelons que de par l'organisation en groupes de travail, ces postes de travail pourront évoluer de manière dynamique appartenant selon les besoins du moment au groupe d'expédition nationale ou à celui de l'expédition régionale.

Comme cela se passe en pratique, la première étape dans notre processus de simulation sera l'arrivée d'un nouveau colis. Nous allons simuler ceci à l'aide de deux modules (un par groupe de travail).

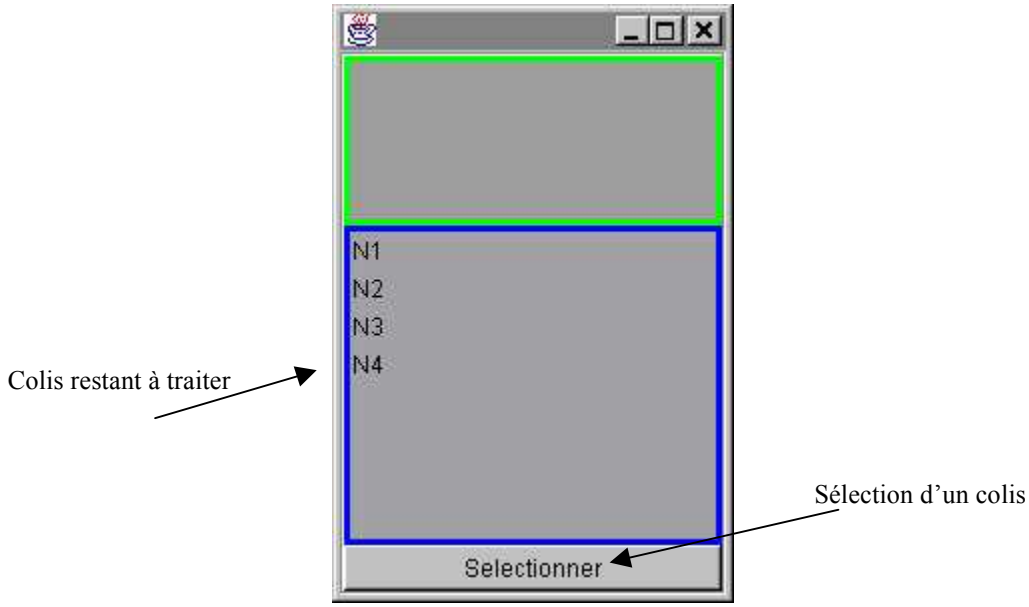
En parallèle à ces deux fenêtres, une troisième nous indique le taux de colis nationaux restant à traiter par rapport aux colis régionaux. Il est ainsi possible de connaître rapidement l'état des stocks.



Le but du prototype n'est pas de réaliser une simulation « temps-réel » mais de pouvoir montrer les mécanismes mis en œuvre. Ainsi, l'ajout de l'interactivité dans l'arrivée d'un colis va nous permettre d'influer sur le déroulement de la simulation en faisant arriver des colis avec diverses destinations afin d'observer le comportement de l'application. Nous pourrons ainsi voir ce qui se passe lorsque, par exemple, un nombre important de colis nationaux arrivent pour un nombre de colis régionaux faible. Nous pourrons voir de cette manière les évolutions des groupes de travail dynamiques.

Lorsque le colis arrive, il est renseigné (travail réalisé ici automatiquement par le prototype) puis est affiché sur des interfaces afin d'y être traité. Nous allons proposer deux interfaces de saisie. Elles afficheront l'ensemble des colis restant à traiter et ce, relativement à leur groupe. La seule action possible sera la sélection de l'un d'eux, ce qui aura pour effet de mettre à jour les autres interfaces de ce groupe.

Voici la capture d'écran de l'une d'elles appartenant au groupe National :



A chaque sélection d'un colis (N1, N2, …), l'architecture coopérative va récupérer les information relatives à cette sélection afin de les faire circuler vers les entités concernées provoquant éventuellement un changement de groupe lorsque l'événement déclencheur sera levé. Une fois le colis sélectionné, il disparaît de la liste des colis restant à traiter.

Lorsque l'équilibre entre les colis nationaux et régionaux n'est plus raisonnable, il y a un changement de groupe pour une ou plusieurs des interfaces conformément à ce qui a été dit lors des améliorations rendues possibles grâce à la mise en coopération de l'application.

À delà de ces aspects purement liés à l'application cible, le but du prototype est aussi de pouvoir valider l'architecture de la plate-forme coopérative. Nous désirons implémenter et vérifier la cohérence de nos propositions. Ainsi, parallèlement aux aspects de simulation, nous avons développé la plate-forme coopérative. De manière à la rendre visible et ainsi vérifier nos propositions, chaque gestionnaire, chaque élément de coopération, chaque règle se déclenchant se traduira de manière graphique. Nous pourrons ainsi suivre « à la trace » le cheminement des informations et donc vérifier qu'il est conforme aux descriptions que nous en avons faites.

Il n'est pas nécessaire d'afficher tous les gestionnaires, ainsi par lisibilité, nous allons nous restreindre au Gestionnaire d'Éléments de Coopération, au Gestionnaire de Communications, au Gestionnaire de Règles ainsi qu'aux règles (ÉCA et détectrices) elle mêmes.

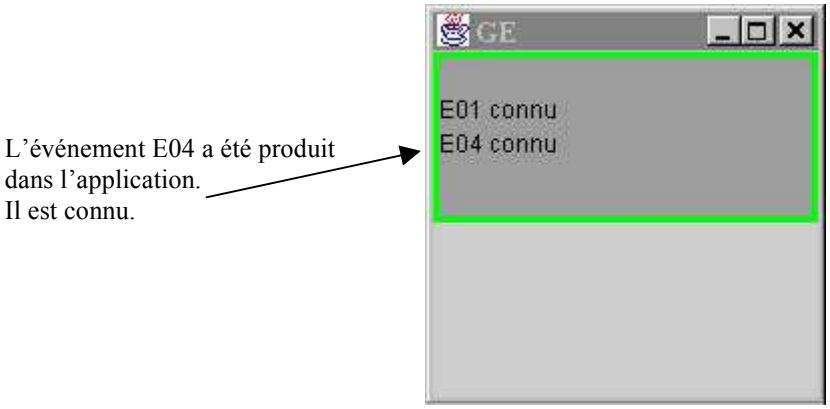


Figure 63 : Interface du Gestionnaire d'Éléments de Coopération

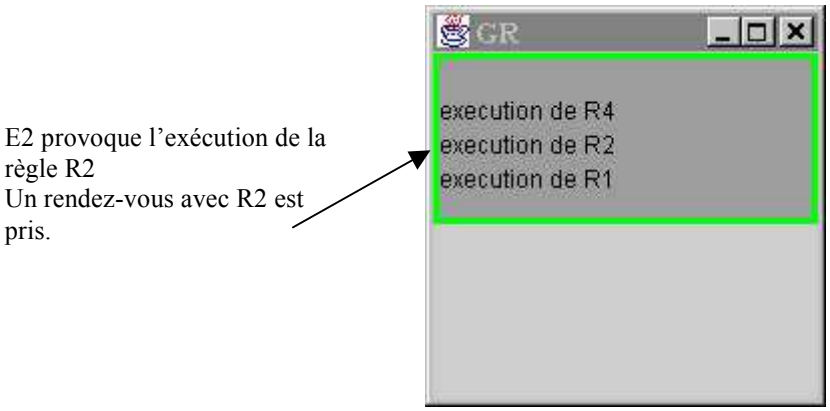


Figure 64 : Interface du Gestionnaire de Règles

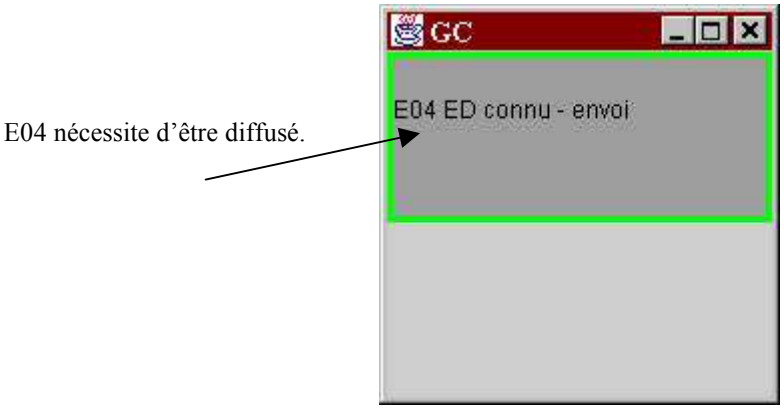


Figure 65 : Interface du Gestionnaire de Coopération

De manière à rendre « plus lisible » le prototype, nous avons également réalisé une représentation graphique de chacune des règles, qu'elle soit ÉCA ou détectrice. Ainsi, nous pouvons connaître les règles actuellement en cours d'exécution. Chacune d'elle s'affiche dans une fenêtre contenant un bref descriptif quant à son rôle. Contrairement aux détectrices, les règles ÉCA n'auront pas un affichage permanent, mais seront présentes à l'écran le temps de leur exécution (moyennant une temporisation afin d'avoir le temps de les visionner).

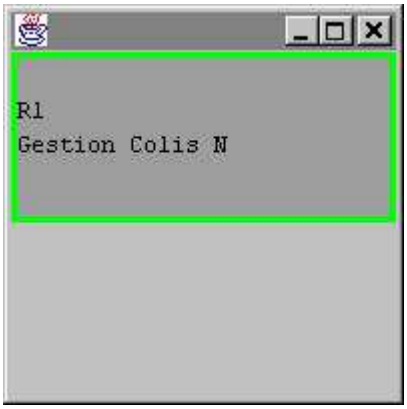


Figure 66 : Interface d'une règle détectrice

Nous allons maintenant expliquer le fonctionnement interne du prototype et plus particulièrement l'organisation et le mode de communication utilisés au sein de la plate-forme coopérative.

3.5 Éléments d'implémentation

L'implémentation a scrupuleusement suivi les recommandations définies lors de la description de l'architecture coopérative. Ainsi, les différents gestionnaires sont des tâches s'exécutant en parallèle, les communications entres les entités sont adaptées à l'information à échanger, chaque type d'élément de coopération correspond à un mode de communication.

L'implémentation a été entièrement réalisée en utilisant le langage de développement Java dans sa version 1.2. Ceci n'est pas dû à un phénomène de mode, mais parce que nous avons trouvé nombre d'avantages à son utilisation. Ce qui nous a principalement intéressé sont les possibilités offertes par les nombreuses librairies fournies, en particulier :

- Les accès réseau simplifiés,
- Le support de RMI, une implémentation « purement » Java de Corba,
- Un accès aux SGBD uniformisé, de même qu'un support de l'ensemble des ténors du domaine,
- La rapidité de prototypage.

L'ensemble de ces avantages fait qu'avec un seul outil, nous pouvons mettre en œuvre le prototype de A à Z. Nous n'allons pas présenter ici ce langage. Nous pensons qu'il a été suffisamment expliqué mis au devant de la scène depuis environ trois ans. Néanmoins, le lecteur pourra se référer à [Rodley 96] [Eckstein 98] [Dedieu 00]. Aussi nous n'allons pas nous attarder plus longtemps sur le langage et centrer nos efforts sur la communication entre ces entités.

Nous avons vu que nous avions trois types d'éléments de coopération, il va donc falloir mettre en œuvre autant de techniques de gestion :

- *Evénements*. Ils sont gérés à l'aide du service d'événements (*Event Sercice*) de CORBA. Parmi les deux modes *push* et *pull* décrits précédemment, c'est le *push* que nous avons choisi. En effet, dans notre manière de percevoir les événements, ceux-ci sont des notifications du producteur au consommateur et non une demande du consommateur sollicitant le producteur (mode *pull*). Nous n'avons pas utilisé le service de notification (*Notification Service*) car, par défaut, lorsqu'un événement est produit, l'entité productrice ne connaît pas le destinataire. Elle ne fait que le signaler. Ainsi, elle ne peut réaliser de filtrage permettant de ne notifier que les événements reconnus par le récepteur.
- *Messages*. Pour les gérer, le service de messageries proposé par CORBA (*CORBA Messaging*) correspond très bien à nos besoins puisqu'il permet aux entités (émetteurs, destinataires) de ne pas être présentes simultanément.
- *Données*. Pour les données, nous ne faisons plus appel à CORBA et allons utiliser la bibliothèque JDBC fournie en standard avec JAVA. Outre les fonctionnalités « standards » d'une bibliothèque permettant l'accès à un SGBD, elle gère en natif un certain nombre de SGBD parmi les plus répandus (Oracle, Informix, xBase, …) ainsi qu'un accès via ODBC (*Object DataBase Connectivity* de Microsoft).

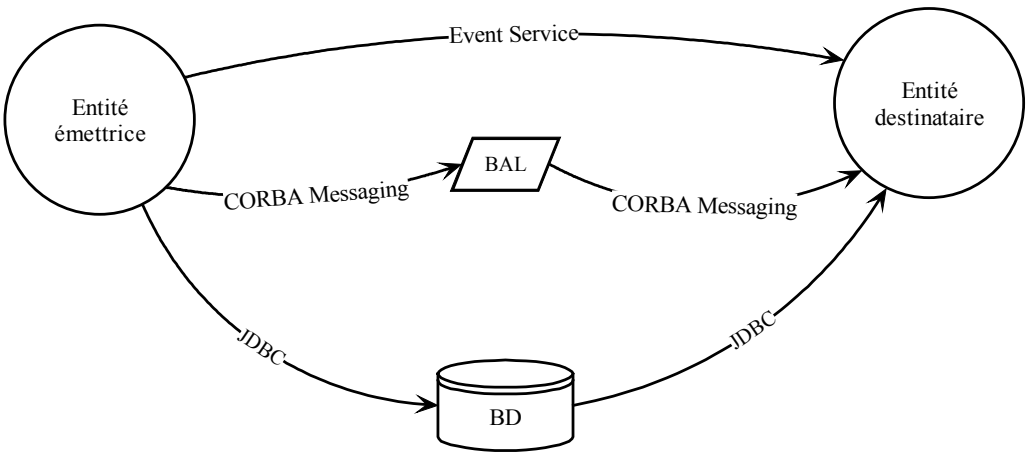


Figure 67 : Moyens de communications utilisés pour la circulation des éléments de coopération

Les entités de l'architecture coopérative peuvent à la fois produire des éléments de coopération, mais aussi en recevoir. Chaque gestionnaire devra donc à la fois implémenter les trois modes de communication, mais devra aussi mettre en œuvre l'ensemble des moyens nécessaires à la production/réception des informations.

3.6 Intégration des modules à la plate-forme coopérative

Pour simuler l'arrivé d'un colis à destination nationale ou régionale, l'opérateur clique sur le bouton de la fenêtre correspondante. Ceci a pour conséquence la levée d'un événement détecté par la plate-forme coopérative. Le $G_{ÉC}$ va le signaler en l'affichant dans sa fenêtre. Si cet événement est reconnu, le GR va inscrire dans sa fenêtre les règles concernées de même qu'il signalera lorsque ces règles nécessiteront d'être lancées. Enfin, si l'événement doit être diffusé, il apparaîtra dans la fenêtre du GC qui nous dira où il l'envoie.

De manière à avoir un fonctionnement contrôle de la dynamique des groupes, lorsqu'une interface aura un nombre de colis en stock supérieur à cinq, l'autre interface passera dans son groupe (si ce n'est pas déjà le cas). Visuellement nous verrons le contenu du stock être modifié, le titre de la fenêtre sera lui aussi modifié afin de savoir à tout instant dans quel groupe elle se situe.

Chacune des règles détectrices significatives se verra elle aussi dotée d'une interface afin d'y afficher les éléments de coopération qu'elle reçoit, qu'elle produit et afin qu'à chaque instant nous puissions savoir où elle en est. Lorsque rien ne s'y passe, elle disparaît pour réapparaître lorsque ce sera nécessaire.

Le prototype (fichiers Java 1.2 compilés, fichiers sources commentés ainsi que la documentation) est disponible sur le serveur du Laboratoire d'Informatique Appliquée http://lia.univ-pau.fr de l'Université de Pau et des Pays de l'Adour (LIA-UPPA).

Aller au delà de la présentation du prototype réalisée auparavant reviendrait à décrire les détails de l'implémentation elle même ce qui n'a que peu d'intérêt ici. Aussi, le lecteur intéressé se référera aux sources et aux commentaires.

CONCLUSION ET PERSPECTIVES

1. Conclusion

En quelques années, les techniques liées à l'informatique ont connu un grand bond en avant. Le travail coopératif a depuis toujours été une voie privilégiée des espoirs à la fois des utilisateurs mais aussi des informaticiens. Nous commençons depuis peu à voir apparaître des applications coopératives répondant aux besoins des utilisateurs.

Si elles ont pu être réalisées, c'est en grande partie parce qu'elles ont su exploiter les capacités des nouvelles technologies (autant matérielles que logicielles). Néanmoins, peu de personnes se sont penchées sur le problème des applications existantes. En effet, si de plus en plus de d'entreprises sont conscientes de l'apport que peut procurer le travail en coopération, elles n'ont pas toujours les moyens, ni les possibilités de le réaliser. Les responsables informatiques hésitent énormément à changer un système qui tourne et qui réalise bien ce pourquoi il a été originellement créé. Ces applications ont souvent été développées en interne et tournent de manière fiable depuis plusieurs années. On comprend aisément leurs réticences.

Malgré cela, ils sont conscients des avantages que la coopération peut leur apporter et souhaiteraient faire évoluer leurs applicatifs de manière à les rendre encore plus performants. Ils sont tout à fait conscients du « passifs » qu'elles ont et savent pertinemment que « l'on ne fait du neuf avec du vieux » . Malgré cela, les besoins se font sentir et, il nous semble nécessaire de proposer une réponse à leurs attentes.

C'est dans cet esprit de réutilisation d'applications que nous avons proposé dans cette thèse une démarche globale allant de l'analyse jusqu'à l'implémentation dans le domaine très particulier de la ré-ingénierie. La spécificité de ce champ d'application est que nous ne créons pas des applications mais que nous les modifions. Cette approche introduit de nombreuses contraintes puisque nous n'avons pas la liberté de réaliser tout ce que nous voudrions. Intervenir sur le code n'est en général pas souhaitable et de plus il n'est pas toujours possible d'obtenir les informations désirées. Il faut en permanence composer, d'où l'utilité d'une méthode souple permettant de revenir sur les étapes précédentes lorsque l'obtention d'une information n'est pas réalisable.

Nous avons délibérément choisi une approche imposant le moins possible (et dans l'idéal, pas du tout) de modifications de l'existant. L'adjonction d'une plate-forme coopérative autorise une nouvelle

organisation de l'application, tant au niveau fonctionnel que structurel, et permet de modifier les rôles joués par certains modules.

Notre choix a été, dès le départ, d'avoir une approche globale. Il nous fallait pour cela définir une méthode complète aisément dérivable afin d'en faciliter l'implémentation. Les résultats obtenus confirment nos choix puisque cette dernière est entièrement dérivable (et en grande partie de manière automatique) en règles.

Le développement de la plate-forme coopérative répond aux mêmes critères puisqu'elle permet l'intégration des règles et la circulation des informations ce qui aboutit à la mise en œuvre pratique de la coopération dans une application qui ne l'était pas.

Lorsque l'on s'intéresse à une application complexe ayant une longue vie derrière elle, il est très difficile d'en connaître tous les détails de structure et de fonctionnement. En ce qui concerne notre approche, il faut pouvoir non seulement proposer une structuration en groupes de travail viables, mais également entrer dans le détail de chaque module afin de déterminer les éléments de coopération qu'il peut produire et recevoir ainsi que la façon de les échanger avec lui.

L'approche n'est bien évidemment pas parfaite. Bien que complète, la méthode est lourde à utiliser. De plus, elle nécessite une étude préalable longue et détaillée de manière à connaître dès le départ à la fois l'existant et les besoins en terme de coopération.

Néanmoins, ELKAR fonctionne, propose une approche globale couvrant les aspects méthodologiques en allant jusqu'à la mise en œuvre concrète de la coopération.

2. Perspectives

Elles sont à la fois nombreuses, variées et prometteuses. La méthode proposée est ici entièrement centrée sur la ré-ingéniérie d'applications non coopératives constituées en modules. C'était l'objectif premier que nous nous étions fixé. Néanmoins, plusieurs évolutions sont envisageables de manière à la rendre plus formelle et moins « intuitive ».

Ainsi, envisageons nous le développement d'un atelier de génie logiciel permettant l'automatisation la plus complète possible. L'analyse pourrait alors être entièrement réalisée sur machine à la manière de ce qui est fait avec des outils comme Rational Rose (*Rational Rose Corporation*) pour UML. Le langage de spécification serait ainsi automatiquement généré de même que les vérifications de faisabilité des compositions. Les retours dans l'analyse s'en trouveraient grandement facilités. Enfin, la dérivation serait entièrement obtenue par une génération de code en Java (ou autre ?) du squelette des différentes règles détectrices et ÉCA.

De plus, afin de faciliter la tâche de migration de l'application qui doit pouvoir se faire sans paralyser l'entreprise, il serait fortement souhaitable de proposer un outil de simulation permettant de tester l'organisation coopérative proposée avant de la mettre définitivement en place. Cet outil devrait permettre de générer un modèle comparable à celui que nous avons réalisé pour mettre à l'épreuve notre prototype.

Les résultats obtenus laissent aussi à penser que notre démarche pourrait être reprise pour être adaptée à la création (et non plus la ré-ingénierie) d'applications coopératives basées sur la réutilisation de composants logiciels et matériels. L'accroissement de la liberté donné par la création permettrait de disposer de modes de communication plus efficaces comme par exemple la mise en œuvre de communications synchrones qui permettraient la mise en place de points de synchronisation et de reprise dans l'application.

La plate-forme coopérative elle même devrait alors être étendue en commençant, là aussi, par proposer des services de synchronisation (une évolution des services CORBA est en cours pouvant faciliter ce travail).

REFERENCES BIBLIOGRAPHIQUES

[Abiteboul 87]
S. Abiteboul, R. Hull
A Formal Semantic Database Model
ACM Transaction on Database Systems – 12(4), pp. 525-565 – Décembre 1987.

[ADA 83]
Department of Defense - ADA Joint Program Office
Reference Manual for the ADA Programming Language
ANSI/MIL-STD-1815A – Washington D.C. – Government Printing Office – January 1983

[Aiken 98]
P.H. Aiken
Reverse Engineering of data
Vol. 37, N°2 – IBM Systems Journal – 1998

[Allen 94]
Allen R, Garlan D
Formal Connectors
CMU-CS-94-115 – School of Computer Science – Carnegie Mellon University- Pittsburgh – PA 15213 – March 1994

[Aniorté 92]
Aniorté Philippe, Ghaleb Jaber
Un Système de Gestion de Bases de Données Relationnel Extensible et Typé »
Congrès Inforsid – Clermont-Ferrand – Mai 1992.

[Arango 94]
G. Arango
Domain Analysis Methods, in Software Reusability
Édité par W. Shäfer – R. Priéto-Diaz – M. Matsumoto
Ellis Horwood – 1994.

[Arnold 93]
André Arnold, Irène Guessarian
Mathématique pour l'informatique
Éditions Masson – ISBN 2-225-84011-3

[BenAtallah 95]
S. Ben Atallah, R. Kanawati, R. Balter, M. Riveill
CoopScan : une plate-forme générique pour le développement de collecticiels
Septièmes Journées de l'Ingénierie de l'Interaction Homme-Machine (IHM'95) - pp. 21-26 - Cépaduès-Éditions - Toulouse, octobre 1995.

[Babaoglu 93]
Osalp Babaoglu, Keith Marzullo
Consistent Global States of Distributed Systems : Fundamental Concepts and Mechanisms
Technical Report UBLCS 93-1 - Laboratory of Computer Science – University of Bologna – Bologna – Italy– Janvier 1993.

[Bailey 94]
Bailey James, Georgeff Michael, Kemp David, Kinny David, Ramamohanarao Kotagiri

Active Databases and Agent Systems – A comparison
Proc. of the 2nd Int'l Workshop on Rules in Databases Systems – Lecture Notes in Computer Science 1985 – pp. 342-356 – Athènes – Grèce – 1994.

[Balter 94]
Balter R., Lacourte S., Riveill M.
The Guide language : Design and Experience
The Computer Journal – Vol. 37– pp. 519-530 – Décembre 1994.

[Balter 95]
R. Balter, S. Krakowiak
Objectifs et plan de travail du projet Sirac
Rapport Sirac 1-95 – IMAG-LSR - Juin 1995.

[Balter 96]
R. Balter, S. Ben Atallah, R. Kanawati, M. Riveill
Collecticiels synchrones: analyse des besoins et étude des architectures
Rapport Technique 9-96, LSR-IMAG, Projet Sirac, mars 1996.

[Balter 96]
R. Balter, M. Riveill
Environnement de développement d'applications réparties : objectif et plan de travail
Rapport Sirac 7-96 – Février 1996.

[Belkhatir 91]
Belkhatir N., Melo W.L.
Adele2 : a support to large software development process
1èr Software Process Workshop – IEEE Press – Octobre 1991.

[Belkhatir 93]
Belkhatir N., Melo W.L.
Supporting Software Maintenance Processes in TEMPO
Int'l conf. On Software Maintenance – IEEE Press – Montreal – Canada - Septembre 1993.

[Belkhatir 94]
Belkhatir N., Melo W.L.
The Need to a Cooperative Model : The Adele/Tempo Experience
9ème Software Process Workshop – IEEE Press - Octobre 1994.

[Bellissard 95a]
L. Bellissard, S. Ben Atallah, A. Kerbrat, M. Riveill
Component-based Programming and Application Management with Olan
Workshop on Object-Based Parallel and Distributed Computation - Lecture Notes on Computer Science - Springer Verlag, 1996 – Tokyo - Juin 1995.

[Bellissard 95b]
L. Bellissard, F. Boyer, M.Riveill
Construction and Management of Cooperative Distributed Applications
Int'l Workshop on Object Orientation in Operating Systems (IWOOOS'95) - pp. 149-152 – IEEE - University of Lund – Sweden - August 1995.

[Bellissard 95c]
L. Bellissard, M.Riveill
Olan: a Language and Runtime Support for Disrtributed Application Configuration
Journées du GDR de Programmation – Grenoble – France - November 1995.

[Bellissard 96a]
Luc Bellissard, Slim Ben Atallah, Fabienne Boyer, Michel Riveil
Distributed Application Configuration

Proc. 16th Int'l Conf. on Distributed Computing Systems - pp. 579-585 - IEEE Computer Society - Hong-Kong - May 1996.

[Bellissard 96b]
L. Bellissard, M. Riveill,
Du Distributed Objects to Distributed Components: the Olan Approach.
Workshop Putting Distributed Objects to Work - ECOOP'96 – Linz – Austria - Juillet 1996.

[Berndtsson 98]
Mickael Berndtsson
Active Capabilities Support for Cooperation Strategies in Cooperative Information Systems
Thèse de l'université d'Exceter – 1998.

[Blaha 97]
M. Blaha
Dimensions of Relational Database Reverse Engineering
Proceedings of the Fourth Working Conference on Reverse Engineering - October 6-8 – 1997 – Amsterdam - The Netherlands - IEEE Computer Society Press (1997) - pp. 176-183.

[Booch 91]
Booch G. – *Object Oriented Design with Applications*
The Benjamin/Cummings Publishing Company, Inc. – 1991.

[Boyer 96]
F. Boyer, E. Lenormand, V. Marangozov
Un modèle d'événement pour le support de la coordination dans un système à objets répartis
Rapport Sirac 10-96 – Février 1996.

[Briand 95]
Cyril Briand
Conception Orientée-Objet et Basée Réseaux de Petri de la Conduite Temps Réel des Systèmes Flexibles de Production Manufacturière
Thèse de l'université Paul Sabatier – Toulouse - 1995.

[Campo 97]
M. Campo, A. Ortigosa, C. Marcos.
Framework comprehension and design patterns: A reverse engineering approach.
Proceedings of the 9th Int. Conf. On Software Engineering and Knowledge Engineering - SEKE'97 – Madrid – Espagne - June 1997.

[Casati 96]
F. Casati, S. Ceri, B. Pernici, G. Pozzi
Deriving Active Rules for Workflow Management
7th DEXA – Zurich – Suisse – Septembre 96.

[Cauvet 99]
C. Cauvet, F. Semmak
La réutilisation dans la réingénierie des systèmes d'information
Genie Objet-Analyse et Conception de l'évolution – sous la direction de C. Oussalah – Hermès – 1999.

[Ceri 97]
Ceri Stefano, Fraternali Piero
Designing Database Applications with Objects and Rules
Addison-Wesley – ISBN 0-201-40369 2 – 1997.

[Chakravarthy 93]
Charavarthy S.
Architectures and Monitoring Techniques for Active Databases : an Evaluation
Technical Report UF-CIS-TR-92-041 - University of Florida – 1993.

[Chakravarthy 95]
Chakravarthy S., Tamizuddin Z., Zhou J.
A visualization and Explanation Tool for Debugging ÉCA Rules in Active Databases
Proceedings of the second Intl. Workshop – RIDS'95 – Lecture Notes in Computer Science Vol. 985 – Springer – Athène – Grèce – Septembre 1995.

[Chailloux 89]
J. Chailloux
LE_LISP de l'INRIA Version 15.22, Manuel de référence
INRIA – Le Chesnay – Janvier 1989.

[Coleman 94]
D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes
Object-Oriented Development : The FUSION method
Prentice Hall International : Hemel Hempstead – England – 1994.

[Dayal 88]
Dayal U., Blaustein B., and Buchmann A.
The HiPAC Project : Combining Active Databases and Timing Constraints.
SIGMOD RECORD, vol. 17, no 1 - pp 51-70 - Mars 1988.

[Dedieu 00]
Olivier Dedieu
Support de cours
http://www-sor.inria.fr/~dedieu/java/cours/index.html

[Derycke 95]
Alain Derycke, Frédéric Hoogstoel
Le travail coopératif assisté par ordinateur : quels enjeux pour les concepteurs
Actes du congrès Inforsid 95 – Grenoble – France – 1995.

[Diaz 92]
Diaz O., Jaime A.
EXACT : an Extensible approach to ACTive object-oriented databases
Technical Report – University of San Sabastian – Espagne – Mai 1992.

[Dittrich 93]
Dittrich K.R. and Gatziu S.
Time Issues in Active Database Systems
International Workshop on an Infrastructure for Temporal Databases - Arlington, Texas - Juin 1993.

[Dogac 97]
Dogac, A., Gokkoca, E., Arpinar, S., Koksal, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P., Halici, U., Altinel, M.
Design and Implementation of a Distributed Workflow Management System: METUFlow
Proc. of NATO-ASI on Workflow Management Systems and Interoperability - pp. 60-90 - Août 1997.

[Dubois 98]
Eric Dubois, Philippe Dubois Eric Yu, Mickaël Petit
Du Early to Late Formal Requirements – a Process-Control Case Study
Proc. of the Int'l IEEE - IWSSD9
Isobe – Japan – Avril 1998.

[Eckstein 98]
Robert Eckstein, Marc Loy, Dave Wood
Java Swing
Éditions O'Reilly – ISBN 1-56592-455-X

[Eswaran 76]

Eswaran K.P.
Spécification , implementation and iterations of a trigger subsystem in an integrated DB system
IBM Research report RJ1820 - August 1976.

[Evariste 99]
Evariste Innovation Plus
Le Site de l'Innovation Industrielle et Technologique
http://www.evariste.com/

[Favre 95]
J.M. Favre.
Une approche pour la maintenance et re-ingenierie globale des logiciels
These de l'Institut National Polytechnique de Grenoble - November 1995.

[Ferber 95]
Jacques Ferber
Les systèmes multi-agents
InterEdition – 1995.

[Franklin 97]
Franklin S., Graesser A
Is it an agent or just a program ? A taxonomy for autonomous agents.
Intelligent Agents III - Proc. of the Third Int'l Workshop on Agent Theories, Architecture, and Languages – Vol. 1993 of Lectures Notes in Artificial Intelligence – Springer-Verlag – pp. 21-35 – 1997.

[Front-Conte 94]
Agnès Front-Conte
Règles de Déduction et Règles Actives dans un Modèle de Systèmes d'Information et dans un Langage de Programmation pour Bases de Données
Rapport de DEA Informatique - Institut National Polytechnique de Grenoble - Juin 1994.

[Front-Conte 97]
Agnès Front
Développement des systèmes d'information à l'aide de patrons – Application aux bases de données Actives
Thèse de l'Université Joseph Fourier (Laboratoire IMAG-LSR) – Décembre 1997.

[Gatziu 93]
Gatziu S. and Dittrich K.R.
SAMOS : an Active Object-Oriented Database System.
IEEE - Bulletin trimestriel sur l'étude des données (Data Engineering) - Janvier 1993.

[Gehani 92]
Gehani N.H., Jagadish H.V.
Ode as an Active Database : Constraints and Triggers
Proc. of the 17th Int'l. Conference on Very Large Data Bases – pp. 327-336 - Barcelone – Espagne – Septembre 1992.

[Geib 97]
Geib Jean-Marc, Gransart Christophe, Merle Philippe
CORBA : des concepts à la pratique
Collection InterÉdition – Édition MASSON – Paris – France – *ISBN : 2-225-83046-0* - 1997.

[Georgakopoulos 94]
Georgakopoulos Dimitrios
Workflow Management Concepts, Commercial Products, and Infrastructure for Supporting Reliable Application Processing
Technical Report TR-0284-12-94-165 – GTE Laboratory – Décembre 1994.

[Geppert 95]

Geppert Andreas, Dimitrios Tombros, Markus Kradolfer
Realization of Cooperative Agents Using an Active Object-Oriented Database Management System
2nd workshop on Rules in Databases (RIDS) – Athène – Grèce – 1995.

[Geppert 98]
Geppert Andreas, Dimitrios Tombros
Event-based Distributed Workflow Execution with EvE
Proc. of the Middleware '98 - The Lake District – Angleterre - Septembre – 1998.

[Glaser 96]
Glaser Norbert
Contribution à l'acquisition et à la modélisation de connaissances dans un cadre multi-agents : l'approche CoMoMAS
Thèse de l'Université Henri Poincaré – Nancy I – Novembre 1996 – France.

[Gouardères 98]
Éric Gouardères, Jean-Marie Condom
Pilotage des systèmes de production et approche multi-agents : une expérimentation
2° jounées Acteurs, Agents & Apprentissage – Bayonne - Septembre 1998.

[Iglesias 98}
Carlos Iglesias, Mercedes Garijo, José C. Gonzalez, Juan R. Velasco
Analysis and design of multi-agents systems using MAS-CommonKADS
In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, Intelligent Agents IV (LNAI Volume 1365) - pp. 313-326 – Springer-Verlag – Berlin – Germany - 1998

[Ignatin 92]
Gary R. Ignatin
Let The Hackers Hack : Allowing the Reverse Engineering of Copyrighted Computer Programs To Achieve Compatibility
Raport Interne - Smith Lyons (http://www.smithlyons.ca/Publications/Articles/IT_99_10_2.htm) - 1992

[Ilog 91]
Société Ilog
RPC Link version 3.25.2, Manuel de référence
Société Ilog – Gentilly – 1991.

[InfoMag 97]
Informatique Magazine – N°37 – p.82 - novembre 1997.

[ITU 94]
CCITT specification and description language (SDL) – Technical Report ITU-T – Juin 1993.

[Jablonski 98]
Stefan Jablonski
A Software Architecture for Workflow Management Systems
9th Workshop of Database and Expert Systems Applications – Vienne - Autriche, Août – 1998.

[Jagannath 97]
Mahesh K. Jagannath
A workflow management and simulation system and its supporting object-oriented knowledge base management system
Thesis of the university of Florida - 1997.
[Kaashoek 91]
M.F. Kaashoek, A.S. Tanenbaum - *Groupe Communication in the AMOEBA distributed system* – Proc. IEEE 11th Int'l. Conf. On Computer Systems - p.222-230 – Mai 1991.

[Kanawati 97]
Rushed Kanawati
Construction de Collecticiels : étude d'architectures logicielles et de fonctions de contrôle

Thèse IMAG-INRIA – 1997.

[Kazman 98]
Rick Kazman, Steven S. Woods, S. Jeromy Carrière
Requirements for Integrating Software Architecture and Reengineering Models: CORUM II.
5th Working Conference on Reverse Engineering - WCRE '98 - October 12-14 – 1998 – Honolulu – Hawai - USA. IEEE Computer Society Press – 1998 - ISBN 0-8186-8967-6 154-163

[Kinny 97]
D. Kinny, M. Georgeff
Modelling and Design of multi-agents systems
In J.P. Müller, M.Wooldridge and N.R. Jennings, editors
Intelligent Agents III (LNAI Volume 1193), pp. 1-20 – Springer-Verlag – Berlin – Germany – 1997.

[Krishnamurthy 95]
Krishnamurthy B., Rosemblum D.S.
Yeast : a general purpose event-action system
IEEE Transaction on Software Engineering – 21(10) – Octobre 1995.

[Koshel 98]
Arne Koshel, Peter C. Lockemann
Distributed Events in Active Database Systems – Letting the Genie out of the Bottle
Journal of Data and Knowledge Engineering (DKE) – Vol. 25 – pp. 11-28 – 1998.

[Lebastard 93]
Franck Lebastard
CHOOE : Un gestionnaire d'environnement distribué
Rapport technique 93-22 – Sophia Antipolis – CERMICS - INRIA - Décembre 1993

[Mc Carthy 89]
Mc Carthy D.R., Dayal U.
The architecture of an Active Data Base Management System
Proceeding ACM-Sigmod Conference – pp. 215-224 – Portland – Mai 1989.

[Maiden 91]
N. Maiden
Analogy as a paradigm for specification reuse
Software Engineering Journal 6(1) – 1991

[Martineau 96]
P. Martineau, J-L. Labesse, C. Carmier, F. Cotard, C. Proust
A generic model for flexible manufacturing systems
(FMS) modelling, Workshop on Production Planning and Control – (WPPC'96), FUCaM/EIASM/ICM - pp. 246-250 - Mons – Belgique – Septembre 1996.

[Martineau 97]
P. Martineau, C. Tacquard, A. Rouillon
Génération automatique de la conduite d'un système flexible de production
E3i Université F. Rabelais - Rapport interne n°197 - Tours - Octobre 1997.

[Melo 93]
Walcélio Louzada Martins Melo
TEMPO : Un environnement de Développement Logiciel Centré Procédés de Fabrication.
Thèse de l'Université Joseph Fourier – Grenoble – 1993

[Microsoft 94]
Microsoft
The component Object Model : Technical Overview
Octobre 1994

[Molet 98]
Hugues Mollet
Conception et gestion de systèmes de production : un exemple de coopération entre concepteurs et exploitants
Journée Prosper – La coopération dans la production - Université de Toulouse II - Octobre 1998.

[Muller 97]
P.A. Muller
Modélisation Objet avec UML
Eyrolles – 1997

[OMG 94]
J. Siegel
Common Object Request Broker : Architecture and Specification
Volume I, N°94-1-1 – Mars 1994.

[OMG 98]
http://www.omg.org/corba/sectrans.html

[Ovum 99]
http://www.ovum.com

[Owezarski 96]
Philippe Owezarski
Conception et formalisation d'une application de visioconférence coopérative – Application et extension pour la télé-formation
pp.25-36 - Thèse du LAAS – Toulouse – décembre 1996.

[Owns 94]
Owns K., Adams S.
Oracle 7 Triggers : The Challenge of Mutating Tables
Database Programming & Design – Octobre 1994.

[Pandey 93]
Ranju Pandey, James C. Browne
Event-Based Composition of Concurrent Programs
6th Languages and Compilers for Parallel Computing Workshop – 1993.

[Paton 94]
Paton N.W., Diaz O.
Active Database Systems
Technical Report – Novembre 1994.

[Percival 99]
Robert L. Percival
Standing on the Shoulders of Giants : The Reverse Engineering of Computer Software and the Law of Copyright In Canada
Rapport de l'Université du Maryland – USA - (http://www.law.umaryland.edu/faculty/percival.asp)
Octobre 1999.

[Priéto 90]
Priéto-Diaz R., Freeman P.
Domain analysis : an introduction
ACM SIGMOD – Software Engineering Notes Vol. 15 – N°2 – Avril 1990.

[Rieu 99]
D. Rieu, L. Gzara, M. Tollenaere.
Un Référentiel Générique De Données Techniques à Des Fins De Réutilisation En Ingénierie Des Systèmes D'information Produit

Actes du 3ème Congrès International de Génie Industriel - 26-28 Mai 1999 – Montréal – Canada - pages 141-151.

[Rodley 96]
John Rodley
Writing Java Applets
Coriolis Groupe Books – ISBN 1-883577-78-0

[Rolland 88]
Rolland C., Foucault O., Benci G.
Conception des Systèmes d'Information : La méthode REMORA
Eyrolles – 1988.

[Roubellat 95]
T. Villemur
Conception de services et de protocoles pour la gestion de groupes coopératifs.
Thèse de l'Université Toulouse III – Janvier 1995.

[Rumbaugh 91]
J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen
Object Oriented Modeling Modeling and Design
Yourdon Press – Prentice-Hall – 1991.

[Schael 97]
Thomas Schael
Théorie et Pratique du Workflow .
Éditions Spring – 1997 – ISBN 3-540-63456-8.

[Sedgewick 91]
Robert Sedgewick
Algorithmes en langage C
Éditions Addison-Wesley Europe – pp. 496 - ISBN 2 7296 0254 2

[Shreiber 99]
Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Robert de Hoog, Nigel Shadbolt, Walter Van de Velde, Bob Wielinga
Knowledge Engineering and Management The CommonKADS Methodology
ISBN 0-262-19300-0 – 1999.

[Schwarz 94]
Reinhard Schwarz, Friedemann Mattern
Detecting Causal Relationships in Distributed Computations: Dans Search of the Holy Grail.
Journal of Distributed Computing – Vol. 7(3) – pp. 149-174 – 1994.

[Semmak 98]
F. Semmak
Réutilisation de composants de domaine dans la conception des systèmes d'information.
Thèse de Doctorat de l'Université Paris 1 – Février 1998.

[Shands 94]
Deborah Shands
A Formal Method for Classifying Distributed Systems
Technical Report TR27 – Computer and Information Science - Department of the Ohio State University – 1994.

[Tagg 97]
Roget Tagg, Chris Freyberg
Designing distributed and cooperative information systems
Thomson Computer Press - p. 280-297 – ISBN 1-85032-165-5 - 1997.

[Tanaka 92]
Asterio Kiyoshi Tanaka
On conceptual design of active databases
Thèse de la Faculty of the Division of Graduate Studies – Institut de Technolgie de Georgie – USA - Décembre 1992.

[Tawbi 95]
Tawbi Chawki, Jaber Ghaleb, Damau Marc
Activity Specification Using Rendez-Vous
2nd Int'l Workshop on Rules in Database Systems – RIDS'95 – Springer – Lecture Notes in Computer Science – Vol. 985 – Athens – September 1995.

[Tel 96]
Gérard Tel, Bernadette Charron-Bost, Friedemann Mattern
Synchronous, Asynchronous, and Causally Ordered Communication.
Journal of Distributed Computing Vol. 9(4) – pp. 173-191 – 1996.

[Teisseire 94]
Teisseire Maguelonne
Le modèle IFO_2 : de la modélisation comportementale à la dérivation
Thèse de l'Université d'AIX – Marseille II – 1994.

[Trique 99]
Roland Trique, Frédéric de Solliers, Nat Makarévitch
Le Jargon Français v3.0
http://www.linux-france.org/prj/jargonf/

[Villaumié 98]
M. Villaumié, Roubellat F.
Ordonnancement en temps réel d'atelier : un exemple d'outil, de son appropriation et de son évolution par retour d'expérience
Journée Prosper CNRS – La coopération dans la production – Toulouse II – Octobre 1998.

[Villemur 95]
Villemur T.
Conception de services et de protocoles pour la gestion de groupes coopératifs
Thèse de l'université Toulouse III – Janvier 1995.

[Widom 91]
Widom J., Filkelstein S.
Set-Oriented Production Rules in Relational Database Systems
SIGMOD RECORD- Vol 19 (2) – pp. 259-270 – 1991.

[Wooldridge 95]
Wooldridge Micheal., Jennings Nicolas .R.
Intelligent agents : theory and practice.
Knowledge Engineering Review 10(2) – pp.115-152 – 1995.

[Wooldridge 97]
M. Woolridge
AgentBased Software Engineering
IEEE Proc. on Software Engineering – 144(1) pp. 26-37 – Février 1997.

[Wooldridge 99]
M. Woolridge, N.R. Jennings, D. Kinny
A methodology for agent-oriented analysis and design
Proc. Of the 3rd Int'l Conf. On Autonomous Agents (Agents 99) – pp. 69-76 – Seattle – USA – 1999

[Woodlridge 00]
Wooldridge Micheal., Jennings Nicolas .R., Kinny David
The Gaia Methodology for Agent-Oriented Analysis and Design
Int. Journal of Autonomous Agents and Multi-Agent Systems – n° 3 - 2000.

[Yu 95]
Éric Yu, Philippe Dubois, Eric Dubois, John Mylopoulos
Du Organization Models to System Requirements – A « Cooperation Agents » Approach
Proc. 3rd International Conference on Cooperative Information Systems - CoopIS-95, Vienna (Austria), May 9-12, 1995. pp. 194-204.

[Yu 95b]
Eric Yu
Models for Supporting the Redesign of Organizational Work
Proc. of the Organizational Computing Systems (COOCS 95) Conference Milpitas – California – USA - August 1995.

[Yu 97]
E. Yu, E. Dubois, G. DeMichelis, M. Jarke, J. Mylopoulos, M. Papazoglou, K. Pohl, J. Schmidt, C. Woo
Cooperative Information Systems : A Manifesto
Cooperative Information System : Trends ans Directiob – Academic Press – pp. 315-363 – 1997

REFERENCES BIBLIOGRAPHIQUES PERSONNELLES

Nationale

- [Roose 97a] Roose Philippe – « *Prototype d'interface active en* JAVA » - Mémoire de DEA (Communication Homme-Machine & Ingénierie Éducative) de l'Université du Maine sous la direction du Pr. Martial Vivet - Le Mans - Juillet 1997.

Internationales avec comité de lecture

- [Roose 97b] Aniorté P., Dalmau M., Roose P. – « *Supporting Collaborative Work in Intranet Using an Active DBMS implemented with Java »* - Proc. WebNet'97 – pp. 935-937 - Toronto - Canada - Novembre 1997.
- [Roose 98a] Aniorté P., Dalmau M., Roose P. – « *Using Active Database Mechanisms to Build Cooperative Applications »* - Proc. Integrated Design on Process Technology (IDPT) 98 - Berlin - Allemagne - Juillet 1998.
- **[Roose 98b]*** Aniorté P., Dalmau M., Roose P. – « *Mechanisms for Synchronous and Asynchronous Communication in Distributed Cooperative Work* » – Proc. 6th Int'l Conference on Information Systems Methodologies (ISM) 98 – Salford – Angleterre – Springer Edition – ISBN 1-85233-079-1 – pp. 102-116 - Septembre 1998.
- [Roose 99c] Aniorté P., Dalmau M., Roose P. –*« How to achieve synchronous and asynchronous communication in distributed cooperative systems using ECA mechanisms* » - Simposio Español de Informatica Distribuida (SEID) 99 – Tórculo Edición ISBN 84-8408-060-9 – pp. 333-343 - Santiago de Compostela - Espagne – Février 1999.
- **[Roose 00a]*** Aniorté P., Dalmau M., Roose P. – *« A method for designing cooperative distributed applications »* - Coop'2000 – 4th Int'l Conference on the Design of Cooperative Systems – IOS Press – *« Designing Cooperative Systems - The use of Theories and Models* » - ISBN 1-58603-042-6 – pp. 369-383 - Sophia Antipolis – France – Mai 2000.
- [Roose 00b] Aniorté P., Dalmau M., Roose P. – *« Turning non-cooperative applications into cooperative ones by adding dynamics to the information system »* - 5th Annual Conference of the UKAIS (UK Academy for Information Systems) – Mc Graw-Hill Publishing Company – ISBN 0 07 709755 6 – pp. 57-66 - Cardiff – U.K. - Avril 2000.
- [Roose 00c] Aniorté P., Dalmau M., Roose P. – *« A method for re-engineering/redesigning cooperatively distributed applications »* - 5th Int'l conf. On Integrated Design & Process Scienc[1]e (IDPT/IDEATE) - Dallas - USA - Juin 2000.

[1] Les publications en gras avec un astérisque sont les plus importantes.

- [Roose 00d] Aniorté P., Dalmau M., Roose P. – *« Redesigning Distributed Applications To Provide Cooperation »* - 14th European Conference on Object-Oriented Programming (ECOOP) - Sophia Antipolis et Cannes - France -12 – 16 Juin 2000

Revue nationale

- [Roose 99a] Aniorté P., Dalmau M., Roose P. – « *Modélisation de la coopération dans les applications distribuées en milieu industriel : prémices d'une méthode* » - Revue Études et Recherches – Vol. 99-02 – 1999.

Revue internationale

- **[Roose 99b]*** Aniorté P., Dalmau M., Roose P. – « *Using Active Database Mechanisms to Build Cooperative Applications* » –Transaction of the SDPS (Society of Design and Process Science) Journal – Vol. 3, N° 1 – pp. 1-14 - Mars 1999.