DIAMOND : Une approche pour la conception de systèmes multi-agents embarqués

Jean-Paul Jamont

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N°attribué par la bibliothèque

THÈSE

pour obtenir le grade de

DOCTEUR DE l'INPG

Spécialité: Informatique

préparée au Laboratoire de Conception et d'Intégration des Systèmes,
dans le cadre de l'Ecole Doctorale
Mathématiques Sciences et Technologies de l'Information

présentée et soutenue publiquement par

Jean-Paul JAMONT

le **29 Septembre 2005**

DIAMOND: UNE APPROCHE POUR LA CONCEPTION DE SYSTEMES MULTI-AGENTS EMBARQUES

Directeur de thèse:

Michel OCCELLO

JURY

Mme	Chantal	ROBACH	Président
M.	Olivier	BOISSIER	Rapporteur
M.	René	MANDIAU	Rapporteur
M.	Michel	OCCELLO	Directeur de thèse
M.	André	LAGRÈZE	Examinateur

A mes parents
A mes grand-parents

The most exciting phrase to hear in sciences, the one that heralds new discoveries, is not 'Eureka!' (I found it!) but rather, 'hmm.... that's funny...'

La phrase la plus excitante à entendre en sciences, celle qui annonce de nouvelles découvertes, n'est pas 'Eureka' (j'ai trouvé !), mais plutôt 'Tiens, c'est marrant...'

Issac Asimov

Remerciements

Cette thèse s'est effectuée au Laboratoire de Conception et d'Intégration des Systèmes (LCIS) de l'INPG à Valence, dans l'équipe COSY (Cooperative Complex Systems).

Je remercie donc en premier lieu, Chantal Robach, directrice du LCIS, pour m'avoir accueilli dans son laboratoire et pour m'avoir fait l'honneur de présider ce jury.

Pour avoir, dans un premier temps, partagé avec moi sa vision du domaine des systèmes multi-agents, dans un second temps, avoir été un directeur de thèse actif et patient, je tiens à remercier tout particulièrement Michel Occello, professeur à l'Université Pierre Mendès-France. Merci d'avoir permis ce travail pluri-disciplinaire, pour le temps que nous avons passé et que nous passerons à travailler ensemble.

Je remercie grandement Olivier Boissier, professeur à l'Ecole Nationale Supérieure des Mines de Saint-Etienne, et René Mandiau, professeur à l'Université de Valenciennes et du Hainaut-Cambrésis, d'avoir accepté d'être rapporteurs de mon travail et d'avoir apporté des jugements très constructifs permettant l'amélioration du manuscrit.

Je remercie également André Lagrèze, Maître de Conférence à l'Université Pierre Mendès France, pour sa participation à ce jury. Merci pour ce projet très riche qu'est le projet EnvSys et merci d'avoir su susciter en moi un grand intérêt pour l'informatique industrielle lorsque j'étais étudiant.

Je souhaite également remercier tout les membres du LCIS et plus particulèrement les membres de l'équipe Cosy (Jean-Luc Koning et Hind Fadil). Je ne pourrais bien sûr pas oublier le personnel enseignant des départements "Informatique" et "Réseau et Télécommunication" de l'Institut Universitaire de Technologie de Valence, pour s'être intéressé à mon travail et avoir été compréhensif lors de la fin de ma thèse.

Merci aussi au personnel du Centre de Ressources Informatiques de l'I.U.T. de Valence pour son amabilité et sa grande efficacité.

Je ne peux m'empêcher d'avoir une pensée, pour ceux qui, les premiers, m'ont intéressé à la recherche, m'ont fait confiance et autorisé à suivre la formation du DEA Informatique en double-cursus avec ma formation d'Ingénieur à l'Ecole Supérieure d'Ingénieur d'Annecy, M. Philippe Bolon, M. Jean-Jacques Curtelin, M. Jacques Dufour, M. Flavio Oquendo et M. Lionel Valet.

Je remercie la société EREIMA, entreprise prestataire de services en -informatique industrielle, automatique et électronique- dans laquelle j'ai été salarié à mi-temps durant les deux premières années de ma thèse, me permettant ainsi de travailler sur ce sujet de recherche. Un grand merci, aussi, aux sociétés Girard et C.F.L. du groupe L.G.R., pour leur compréhension de mon calendrier particulier. A ce sujet, je

remercie tout particulièrement M. Eric Lachaze, M. Jean-Luc Vidalenche et M. Gérald Vigne.

Je remercie, pour l'intérêt qu'ils ont porté à EnvSys et à mon travail, les différents stagiaires qui ont apporté leur contribution au projet EnvSys, notamment Olivier Schira, Alexandre Sanchez, Florent Trolat, Vincent Bouchet, Cédric Reynier, Philippe Guillaud ainsi que Florent Bouchy.

Enfin, mes remerciements vont à ceux qui ont participés, plus ou moins indirectement à ma thèse.
Merci à mes parents pour leur soutien,
Merci à ma soeur,
Merci à ma famille,
Merci à mes amis (Grégory et Amandine, Laurent et Odile -les futurs mariés-, Damien et Christelle, David et Laura, la Sanchez-Family (incluant évidement Stéphanie), Sophie et Nicolas, Jean-Christophe et Rachelle, Nico et tout les autres) pour vous être intéressés à mon travail et m'avoir soutenu tout au long de ma thèse,
Merci à Marcel et Jacqueline, pour l'intérêt que vous avez porté à mon travail,
Merci à Maëlle pour avoir subi plus que les autres ma thèse et pour sa présence inestimable à mes cotés.

Merci à tous.

Table des matières

Table des figures

Liste des tableaux

INTRODUCTION

Motivations

Parmi les domaines de l'informatique qui connaissent un regain d'intérêt figurent les systèmes embarqués. Sous cette dénomination, nous classons tous les systèmes mixtes matériel/logiciel autonomes. Les systèmes embarqués font partie intégrante de notre vie de tous les jours sans qu'on en ait toujours conscience. En tant que particulier, ils nous permettent les communications sans fil (téléphonie), nous assistent dans la conduite de nos véhicules, contribuent à notre confort en régulant intelligemment les différents paramètres de notre environnement (température, humidité) etc. Dans les entreprises, ces systèmes contribuent à la surveillance des installations à risques, permettent le suivi temps réel de production, contrôlent les différents flux de données ou de matières des usines, assurent une traçabilité complète de la production. De plus en plus, les applications envisagées (applications relevant de l'informatique ubiquitaire ou pervasive, de l'instrumentation, des réseaux etc.) mettent en présence plusieurs systèmes autonomes conférant ainsi un caractère distribué et réparti au système global. Les interactions, les collaborations et les dynamiques qui en découlent font que ces systèmes sont des systèmes complexes physiques ouverts.

Le développement de telles applications fait apparaître plusieurs défis. Certains sont liés à l'opérationalisation des entités, la gestion du contrôle et l'observation du système. Les systèmes multi-agents sont une solution proposée par la communauté de l'intelligence artificielle distribuée qui semble adaptée à de nombreux cas. Permettant la mise en oeuvre de l'émergence de structures, ils peuvent proposer une approche efficace pour une grande catégorie de problèmes liés à cette hétérogénéité et à cette distribution des ressources.

Une autre part des défis provient des besoins en réutilisabilité et en évolution (généricité) de ces systèmes qui mêlent à la fois logiciel et matériel. Ces besoins rendent obsolètes les méthodes habituelles qui consistent à séparer la conception du logiciel et du matériel dès le début du cycle de vie du système. En effet, il faut désormais unifier le processus de développement du logiciel et du matériel. Cela relève du co-design, un récent et très actif domaine de recherche .

Objectifs

Les systèmes multi-agents connaissent actuellement un certain engouement, en raison de leur capacité à aborder les systèmes complexes, ce qui conduit naturellement à s'intéresser aux systèmes physiques via l'informatique ubiquitaire. Cependant, ce domaine est dépourvu de méthodes complètes pour l'analyse et la conception de systèmes multi-agents physiques ouverts.

Notre travail consiste à associer les concepts de systèmes multi-agents, d'auto-organisation et de systèmes complexes physiques ouverts. Une démarche et des outils intégrant des éléments de co-design nous permettront ainsi d'unifier le développement du logiciel et du matériel des applications de ce type. La thèse que nous défendons est donc qu'il est possible de créer une méthode co-design pour concevoir des applications relevant de ce domaine. Nous utiliserons pour cela un cycle unifié afin de spécifier le système et afin que le partitionnement ait lieu le plus tard possible. En effet, cette activité où le concepteur décide des parties qui doivent devenir logicielles ou matérielles, fige trop de contraintes de conception empêchant une bonne exploration des différents compromis logiciel/matériel.

Nous nous intéresserons plus particulièrement aux systèmes constitués d'entités autonomes d'un point de vue énergétique et décisionnel. Ces objets physiques nécessitent de mettre en oeuvre la coopération. Ils utilisent donc des primitives de communications permettant l'envoi et la réception de messages. Les entités que nous traitons expriment, au niveau physique, la décentralisation par leur nature d'architectures matérielles autonomes, mobiles et reliées entre elles par des liaisons sans fil. La gestion des communications présente donc des similitudes avec les réseaux sans fil traditionnels, mais la spécificité réside dans la pro-activité imposant l'intégration de la gestion des communications dans les processus décisionnels des entités.

Les applications qui pourront être conçues avec notre méthode utiliseront l'approche multi-agents à deux niveaux. Tout d'abord au niveau de l'applicatif lui même, mais aussi dans la gestion indépendante des communications. Un modèle décentralisé intelligent de communication sera mis en oeuvre dans les fonctions de communication. Les agents mis en jeu dans nos systèmes pourront être hétérogènes mais, si ils veulent pouvoir s'échanger des messages, ils devront adopter cette couche de communication (voir figure 1).

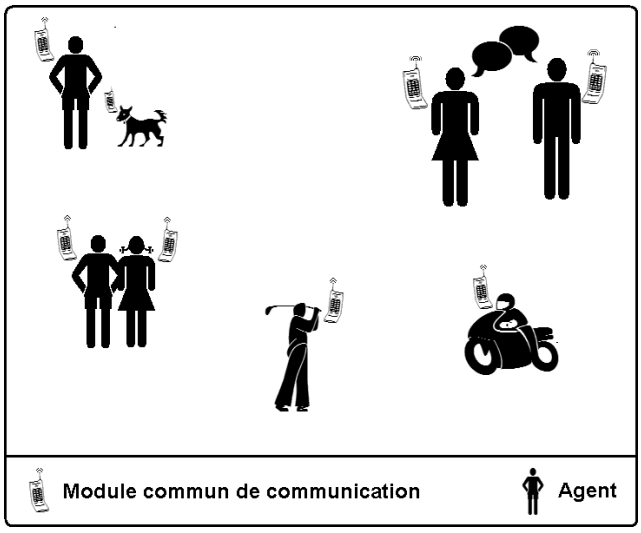


FIG. 1 – *Système multi-agents hétérogène avec des primitives de communications communes*

Contributions visées

Ce travail de thèse va permettre le rapprochement de plusieurs domaines de recherche, pour certains, très différents - les systèmes multi-agents, le génie logiciel, le co-design, les réseaux ad-hoc - au sein de deux projets pluridisciplinaires.

L'originalité de cette recherche réside dans la création d'une méthode pour la conception de systèmes multi-agents physiques ouverts éventuellement auto-organisés. Cette méthode a pour vocation d'être utilisée pour résoudre des problèmes de la famille des systèmes complexes physiques ouverts. Aussi, les contributions se situent au niveau de la démarche, des modèles et des outils.

Les apports méthodologiques consistent, d'une part, en une décomposition système multi-agents d'un problème physique réel, et en une application de l'architecture d'agent hybride e-ASTRO et, d'autre part, en une méthode complète pour résoudre des problèmes complexes à l'aide de systèmes multi-agents physiques. La méthode DIAMOND (Decentralized Iterative Approach for Multiagent Open Networks Design) couvre toutes les activités du cycle de vie d'un système du recueil des besoins à l'implantation.

Les contributions au niveau des modèles concernent le support pour la gestion des communications dans les systèmes physiques complexes ouverts que nous proposons. Ils proviennent aussi de la comparaison entre des techniques d'auto-organisation, intégrant des éléments "cognitifs" tels que les stratégies de gestion de l'énergie, aux techniques utilisées par les protocoles traditionnels. Nous avons entre autres constaté que l'approche totalement décentralisée a grandement diminué la complexité de la réalisation pratique du système multi-agents.

Dans toutes nos applications, il est nécessaire de reconsidérer le maintien de l'intégrité fonctionnelle du réseaux sans fil d'entités physiques suivant une approche décentralisée, collective et hautement adaptative. Les résultats obtenus ont été confrontés à des approches globales traditionnelles.
Les apports pratiques se traduisent par la création d'un outil de développement et par la création d'une plate-forme de simulation pour réseaux ad-hoc permettant de "plugger" simplement de nouveaux types de stations. La mesure des performances permet de quantifier l'émergence induite par l'approche multi-agents. Enfin, les résultats de l'évaluation de l'approche auto-organisée présentent de bonnes performances par rapport à une approche classique par protocole.

Plan du manuscrit de thèse

Le manuscrit de thèse est organisé en cinq parties (voir figure 2).

La première partie expose notre contexte de travail. Dans un premier chapitre nous présentons les systèmes complexes physiques ouverts. Nous commençons par y aborder la classe des systèmes complexes puis insistons sur l'aspect physique ainsi que sur le caractère ouvert des systèmes qui nous intéressent. Dans un deuxième chapitre nous introduisons le paradigme multi-agents. Cette approche est efficace pour la résolution des problèmes complexes ouverts. Nous concluons cette partie sur le constat que la modélisation des systèmes complexes physiques ouverts à l'aide de systèmes multi-agents génère des besoins méthodologiques et architecturaux spécifiques que nous nous proposons d'étudier afin de les intégrer à l'approche et de proposer les outils nécessaires.

La seconde partie du manuscrit présente notre première contribution qui a pour but de répondre aux besoins méthodologiques. Nous allons nous intéresser tout d'abord, dans un premier chapitre, aux procédés de production traditionnelle de logiciel. Dans le second chapitre de cette partie, nous nous penchons sur le cas des systèmes hybrides logiciel/matériel et présentons une manière originale de les développer: le co-design. Ce chapitre présente entre autres un tableau synthétique de notre revue des environnements de co-design existants. Nous présentons enfin notre méthode qui couvre le recueil des besoins (et leur structuration), la phase d'analyse, la phase de conception générique logicielle/matérielle ainsi que la phase de réalisation dans le troisième chapitre de cette partie. Cette méthode, baptisée DIAMOND, sera illustrée par un cas d'étude relevant de la robotique collective. Pour conclure cette partie une discussion confrontera les points introduits dans notre contexte par DIAMOND aux différentes méthodes SMA existantes.

La troisième partie présente notre contribution qui concerne des besoins spécifiques en architectures. Ils portent entre autres sur une abstraction de la gestion des communications entre les entités autonomes qui composent le système. Nous présenterons dans un premier chapitre les solutions qui existent pour permettre la communication dans les systèmes complexes physiques constitués d'entités sans fils. Ce chapitre présente une revue de protocoles utilisés dans les réseaux sans fil. Dans un second chapitre nous étudierons la notion de maintien d'intégrité fonctionnelle en orientant l'analyse sur notre problématique. Ce chapitre présente une revue des techniques d'auto-organisation. Le troisième et dernier chapitre de cette partie présentera notre architecture multi-agents pour le management des communications : l'intergiciel MWAC (Multi-Wireless-Agent Communication).

Dans cette quatrième partie nous présentons une utilisation de la méthode DIAMOND. Un premier chapitre décrit le problème considéré -le projet EnvSys- qui a pour but l'instrumentation d'un réseau hydrographique souterrain ainsi que notre analyse du problème. Dans un second chapitre nous présentons l'étape de simulation ainsi que son implémentation réelle.

La cinquième partie du manuscrit présente les extensions et des développements que ce travail a d'ores et déjà initié et qui ouvrent des perspectives riches tant au niveau méthodologique qu'applicatif. Ainsi dans un premier chapitre nous présentons ces perspectives méthodologiques et dans une second chapitre nous présentons une nouvelle application sur laquelle nous travaillons dans le domaine des systèmes d'informations collaboratifs : la gestion d'une compétition sportive de danse.

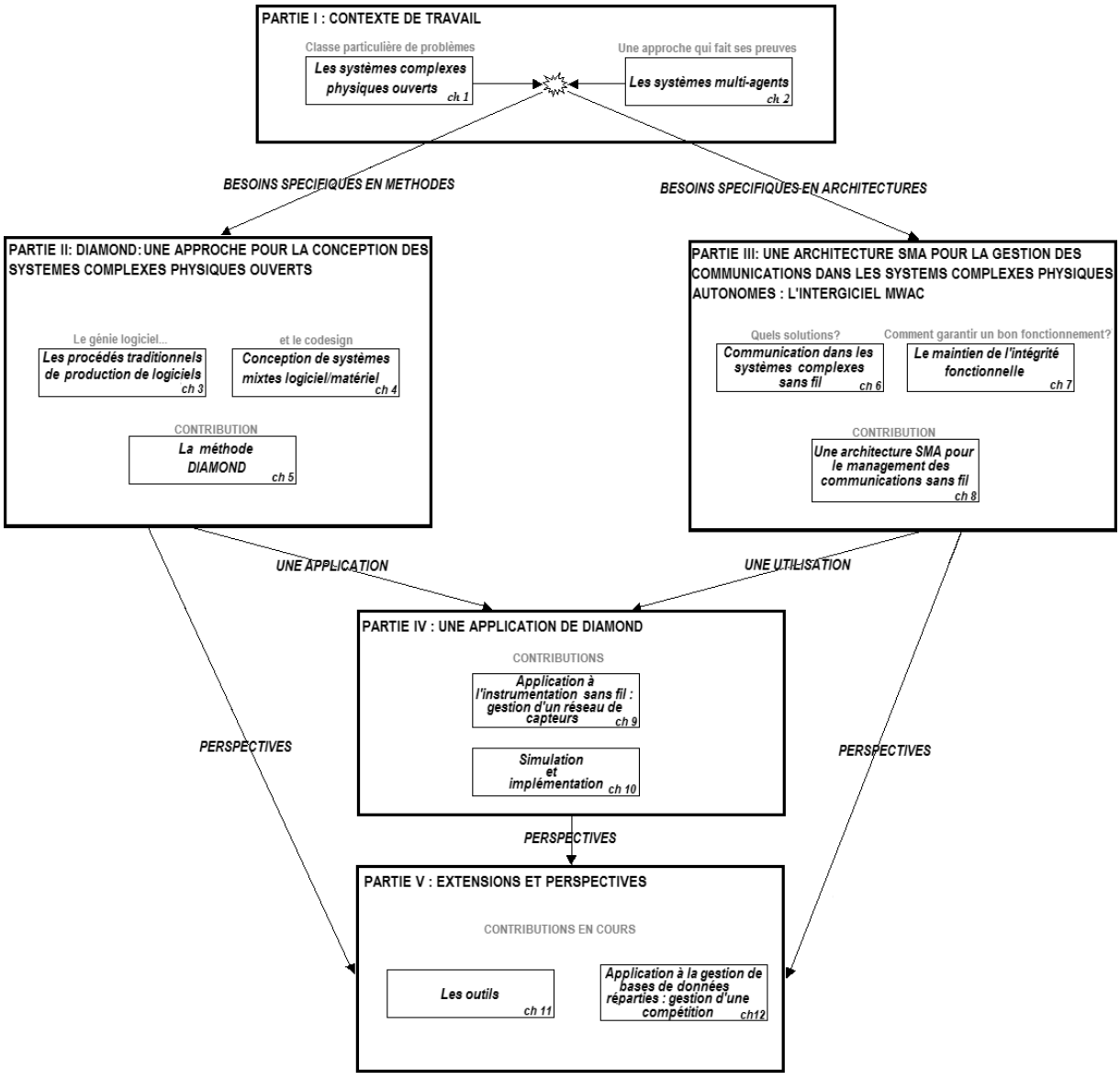


FIG. 2 – *Organisation du manuscrit*

première partie

CONTEXTE DE TRAVAIL

Introduction

Cette première partie expose notre contexte de travail.

Pourquoi s'intéresser aux systèmes complexes ?

Les évolutions des besoins des utilisateurs, toujours de plus en plus gourmands en services et en informations, les avancées des différentes technologies (matériaux, électronique etc.), la mondialisation des informations et des services entre autres via Internet, la baisse des coûts de production des systèmes matériels, et encore tant d'autres paramètres ont rendu complexes de nombreux problèmes que l'informaticien doit traiter. Le passage à grande échelle qui touche quasiment tous les domaines de l'informatique, de la plus logicielle à la plus matérielle est en partie à l'origine de ce nouveau problème qui doit être traité par ce que l'on appelle dès lors l'informatique ubiquitaire [Weiser, 1993], l'informatique diffuse ou encore l'informatique ambiante. Son champ d'application est très vaste allant de l'instrumentation massive aux applications collaboratives multimédia.

Pourquoi s'intéresser aux systèmes multi-agents ?

Les systèmes multi-agents sont une approche privilégiée pour s'intéresser aux systèmes complexes. Leur démarche totalement décentralisée les rend particulièrement adaptés pour ce type de systèmes. Cela vient du fait que leur complexité est souvent fonction du niveau d'observation. Les systèmes multi-agents nous permettent de travailler sur le fonctionnement global d'un système en s'intéressant aux entités qui le compose et à leurs interactions.

Dans un premier chapitre nous présentons les systèmes complexes (artificiels) physiques ouverts qui sont la classe de problèmes qui nous intéressent. Nous commençons par y aborder la classe des systèmes complexes puis insistons sur l'aspect physique ainsi que sur le caractère ouvert des systèmes sur lesquels nous nous concentrons. Dans un second chapitre nous introduisons le paradigme multi-agents. Cette approche est efficace pour la résolution des problèmes complexes ouverts. Nous concluons cette partie sur le constat que la modélisation des systèmes complexes physiques ouverts à l'aide de systèmes multi-agents génère des besoins méthodologiques et architecturaux spécifiques que nous nous proposons d'étudier afin de les intégrer à l'approche et de proposer les outils nécessaires.

Chapitre 1

Les systèmes complexes physiques ouverts

Une réalité complexe reste toujours inachevée et incomplète. On ne peut jamais la comprendre ou la maîtriser entièrement ou définitivement. Dès qu'on pense avoir trouvé des schémas explicatifs globaux, ils se modifient, laissant place à d'autres modèles tout aussi éphémères.

Dominique Génelot, extrait de "Manager la complexité"

Ce chapitre présente les systèmes complexes physiques ouverts qui sont la catégorie de problèmes qui nous intéresse dans ce travail. L'objectif est de définir les systèmes complexes physiques ouverts et de proposer des éléments de caractérisation. Pour ce faire, après avoir défini notre interprétation du terme "système", nous insisterons sur la classe des systèmes complexes, sur celle des systèmes physiques puis sur le caractère ouvert de ces systèmes.

1.1 Les systèmes

Francis Le Gallou [Gallou and Bouchon-Meunier, 1994] définit un système comme étant "un ensemble, formant une unité cohérente et autonome d'objets réels ou conceptuels (éléments matériels, individus, actions, ...) organisés en fonction d'un but (ou d'un ensemble de buts, objectifs, finalités, projets, ...) au moyen d'un jeu de relations (interactions mutuelles, interactions dynamiques...), le tout immergé dans un environnement.".

Il existe de nombreuses autres définitions [Larvet, 1994, Le Moigne, 1990] qui s'accordent généralement sur différents points. Tout d'abord un système est un *ensemble d'éléments*. Selon les domaines, ces éléments sont appelés constituants, composants, objets ou agents. Un composant peut être lui même un système. Dans ce cas on parlera alors de la caractéristique hiérarchique d'un système ou de récursion. Cet ensemble d'éléments est *doté d'une structure* qui précise la nature des liens (relations, interfaces etc.). Ils sont *en interaction entre eux* c'est-à-dire qu'ils peuvent s'échanger des informations, de la matière de l'énergie. Ils sont aussi *en interaction avec l'environnement* car les systèmes ne sont jamais isolés au sens physique du terme. Ainsi un système peut être sollicité par son environnement et y répondre. Il peut donc *réaliser des fonctions*, assurer un ensemble d'activités. Ainsi le système *transforme de l'information, de la matière ou de l'énergie*, il *évolue dans le temps* et il possède des *objectifs*.

1.2 Les systèmes complexes

Définition. Les systèmes complexes [Simon, 1991, Durand, 1998] font l'objet d'études actives dans bien des domaines : la physique, la biologie, les sciences humaines et sociales, les sciences cognitives. Il s'agit donc d'un domaine d'études très pluridisciplinaire. Les mathématiciens, les physiciens théoriciens mais aussi les informaticiens ont permis la création de méthodes, de formalismes et d'outils de modélisation.

D'un point de vue terminologique, le terme "complexe" vient du latin *complexus* qui signifie embrasser et englober. Un problème est qualifié de complexe lorsqu'on éprouve de la difficulté à le rendre intelligible. Cette difficulté peut provenir d'un manque de connaissance, de méthode, de modèle ou de temps. A l'origine de la complexité serait donc l'interaction et la globalité. La définition d'un système ayant été abordée précédemment, le système serait dont complexe de part la diversité et la multitude des interactions.

La figure 1.1 illustre les systèmes complexes en procédant à une décomposition topologique. Elle met en évidence les différents niveaux d'interaction qui permettent à des éléments simples d'être agrégés en des composants plus évolués qui donnent eux-mêmes naissance à des structures organisées et hiérarchisées. Les structures qui émergent alors ne peuvent pas être comprises simplement à partir des entités mises en jeu. La figure fait apparaître le nom des différents concepts inférés - Composition, Auto-organisation, Structures hiérarchiques, Emergence - au niveau où ils interviennent.

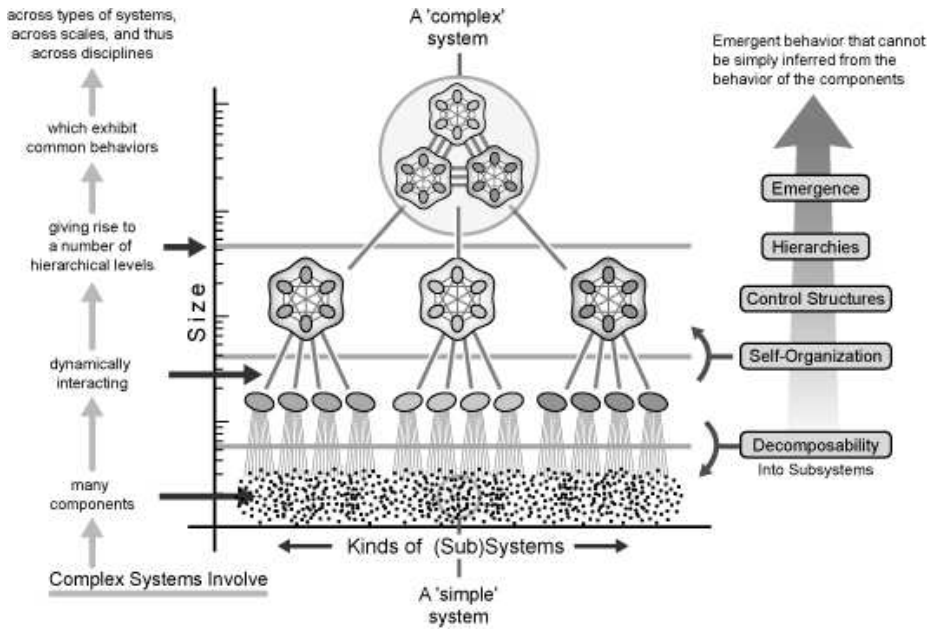


FIG. 1.1 – *Topologie d'un système complexe (illustration tirée de [Idiagram Consulting, 2005])*

Nous adoptons comme définition d'un système complexe "un système composé de nombreux éléments qui interagissent fortement entre eux et sur leur environnement". Ces interactions sont souvent non-linéaires et contiennent généralement des boucles de rétroaction. Au niveau global, ces systèmes se caractérisent par l'émergence de phénomènes non observables au niveau des éléments constitués : un observateur externe appréhendera et comprendra différemment le système qu'un observateur interne au système. Il se caractérise donc par l'émergence au niveau global de propriétés nouvelles et par une dy-

namique de fonctionnement global difficilement prédictible à partir de l'observation et de l'analyse des interactions élémentaires.

Généralement cette famille de systèmes se distingue des autres par l'impossibilité d'identifier tous les éléments et/ou de comprendre les interactions mises en jeu. Cela entraîne généralement l'absence de contrôle total et l'irréversibilité (aucune action ne peut inverser la dynamique pour retrouver avec certitude un des états d'équilibre précédents).

Elements de caractérisation. Les premières études (notamment celles de von Bertalanffy durant les années 1950) étaient centrées sur les concepts de structure, d'information, de régulation et d'organisation. Par la suite, dès les années 1970, elles intègrent deux autres concepts : la communication et l' autonomie. Le NECSI (New England Complex Systems Institute) propose une série de concepts (voir figure 1.2), en considérant la majorité des applications possible dans le domaine scientifique et professionnel, pour donner une idée générale du terme [Bar-Yam, 1997].

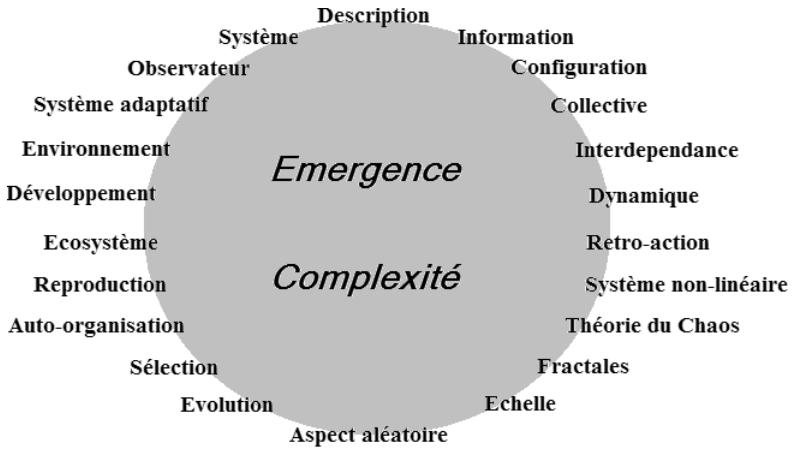


FIG. 1.2 – *Carte des concepts liés à la complexité*

G. Donnadieu et M. Karsky pensent qu'un système peut donner lieu à des complexités de natures différentes [Donnadieu and Karsky, 2002] :

– La *complexité spatiale*. L'interconnexion entre des ensembles de variables interconnectées entre elles et s'influençant mutuellement implique une notion d'influence spatiale s'exerçant et se modifiant dans le temps.

– La *complexité imprévisible*. C'est une complexité qui fait référence au chaos et à l'aléatoire qui décrit un comportement temporel incertain, quasiment imprévisible, ainsi qu'aux bifurcations.

– La *complexité dynamique*. Le comportement temporel d'un ensemble de variables est rendu complexe dès lors qu'apparaissent entre elles des boucles de rétroaction, dès leurs interventions au sein d'un système. Il est alors impossible de prédire le comportement dans le temps du système.

Nous nous permettons une caractérisation volontairement simplifiée des systèmes complexes dans la figure 1.3. Ce que certains appellent le chaos n'apparaît pas car il relève plus de la nature désordonnée des interactions et d'un manque d'organisation : il n'a donc pas réellement sa place dans ce graphique.

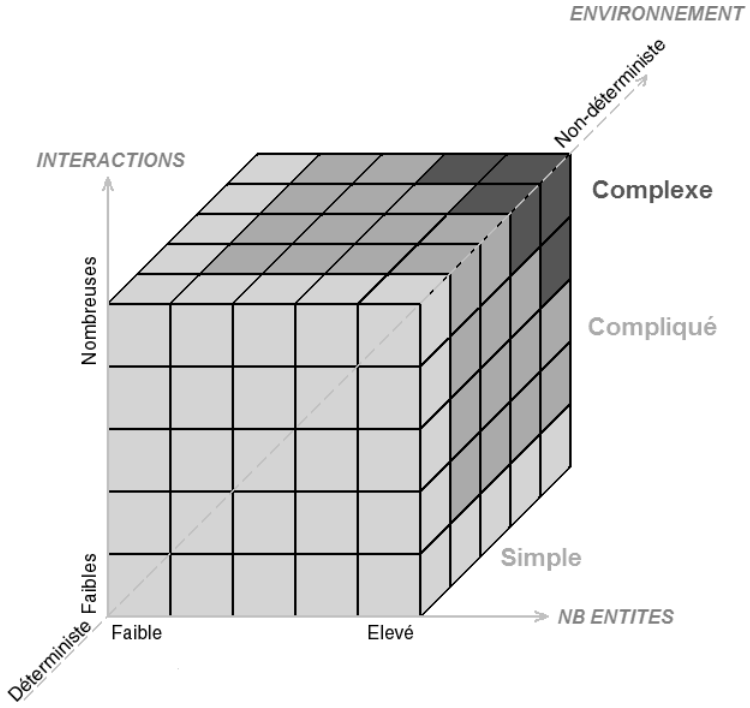


FIG. 1.3 – *Illustration graphique simplifiée des systèmes qui relèvent de la complexité*

1.3 Les systèmes complexes physiques

On entend par *système physique* tout système ayant une réalité physique, tout système non virtuel. Cette caractéristique ne concerne pas uniquement les entités mais aussi l'environnement dans lequel elles évoluent.

Dans notre contexte, ces systèmes sont constitués d'une partie matérielle (carte électronique, capteurs, effecteurs) et d'une partie logicielle (mais ils peuvent être aussi uniquement matériels).

Les systèmes physiques que nous considérons sont les systèmes suivants (ils peuvent cependant appartenir à plusieurs de ces catégories) :

- Les *systèmes de traitement* sont des systèmes qui traitent des informations i.e. des signaux. Il permettent notamment le traitement de la parole, le traitement d'image, le contrôle de production, l'asservissement de systèmes etc.
- Les *systèmes de contrôle/commande* suivent l'évolution de procédés physiques (consigne) et décident d'actions à effectuer (commande).
- Les *systèmes de communication* qui gèrent des flots de messages.
- Les *systèmes interactifs* qui assurent le dialogue homme-machine, permettent de consulter des bases de données, de s'adapter aux utilisateurs etc.

Le caractère physique va accentuer la nécessité d'être tolérant aux pannes. En effet, aux pannes logicielles que l'on rencontre traditionnellement peuvent s'ajouter des pannes "matérielles". Ces dernières peuvent provenir de la destruction de matériel, d'une interférence électromagnétique, d'un problème de température, à la suite de l'activité d'un autre composant du système etc. Au niveau global du système, des pannes peuvent perturber les communications, les perceptions des composants mis en jeux ou les actions faisables par ces dernières.

1.4 Les systèmes complexes physiques ouverts

L'ouverture comme caractéristique d'un système est une notion introduite en thermodynamique dans les années 1950. Mais c'est dans les années 1930 à 1940 avec les travaux sur le vivant de Cannon et de Bertalanffy [von Bertalanffy, 1932, Cannon, 1932] que cette notion s'est considérablement élargie.

Au sens physique, un système est dit ouvert s'il permet des échanges de matières, d'énergies et d'informations avec son environnement sous la forme d'entrée et de sortie. Un système ouvert a donc des frontières qui ne sont pas figées et sont perméables. Il en résulte généralement une plus grande interpénétration avec l'environnement ce qui accroît sa dynamique.

Une des propriétés les plus contraignantes des systèmes ouverts concerne l'ajout et le retrait dynamique d'éléments du système. Il en découle un accroissement de la dynamique dans le système. De plus, ces systèmes mettant généralement en jeu des entités hétérogènes, l'observabilité en est encore complexifiée rendant définitivement caduque tout contrôle global.

Nous appelerons *système complexe physique ouvert* un système:

- qui contient un grand nombre (variable) d'entités logicielles/matérielles,
- qui met en oeuvre de nombreuses interactions logiques/physiques,
- dont l'environnement est physique, dynamique et ouvert,
- qui posséde des objectifs globaux,
- dont le contrôle centralisé est impossible,
- et qui, pour toutes ces raisons, n'est pas entièrement spécifiable.

Chapitre 2

Les systèmes multi-agents

Les parties du monde ont toutes un tel rapport et un tel enchaînement l'une avec l'autre que je crois impossible de connaître l'une sans l'autre et sans le tout.

Blaise Pascal

Ce chapitre a pour objectif d'introduire les systèmes multi-agents qui constituent un des piliers de notre travail sur les systèmes complexes physiques ouverts. Nous nous intéressons tout d'abord aux entités qui composent cette catégorie de système: les agents. Cette notion définie, nous nous intéresserons alors à la caractérisation des systèmes multi-agents.

Depuis une dizaine d'années, les systèmes multi-agents ont connu un grand essor et sont appliqués à des domaines très variés comme, par exemple, le domaine de la simulation et de la vie artificielle, la robotique, le traitement d'images.

Les systèmes multi-agents sont issus de l'intelligence artificielle distribuée (IAD), une branche de l'intelligence artificielle qui s'articule autour de trois axes :

- La résolution distribuée des problèmes qui s'intéresse à la manière de diviser un problème en un ensemble d'entités distribuées et coopérantes et à la manière de partager la connaissance du problème afin d'en obtenir la solution.
- L'intelligence artificielle parallèle qui développe des langages et des algorithmes parallèles pour l'intelligence artificielle (IA) visant ainsi l'amélioration des performances des systèmes d'IA.
- Les systèmes multi-agents qui privilégient une approche décentralisée de la modélisation et mettent l'accent sur les aspects collectifs des systèmes.

Les systèmes multi-agents, comme leur nom l'indique, sont des systèmes constitués de plusieurs agents. Il est donc nécessaire de commencer par définir ce que l'on appelle un agent. La notion d'agent définie, nous introduirons le concept de système multi-agents.

2.1 Les agents

Durant ces dernières années nous avons assisté à un fort et rapide développement des recherches sur les agents et les systèmes multi-agents. Le terme agent est un terme générique qui se rapporte à différentes entités [Franklin and Graesser, 1996]:

- Des entités biologiques : les agents associés sont appelés agents biologiques,
- Des robots autonomes,
- Des logiciels informatiques et leurs composants qu'ils soient intégrés dans des systèmes d'exploitation ou des systèmes informatiques complexes.

2.1.1 Définitions

L'agent. Il n'existe pas, actuellement, une définition de l'agent qui fasse foi dans le monde de l'intelligence artificielle distribuée. Il est donc nécessaire, pour avoir une bonne vision de ce concept, de confronter plusieurs de ces définitions. Nous allons exploiter trois d'entre elles.

1. Jacques Ferber [Ferber, 1995] définit un agent comme étant une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir (voir figure 2.1). Il est capable de communiquer avec d'autres agents et est doté d'un comportement autonome.

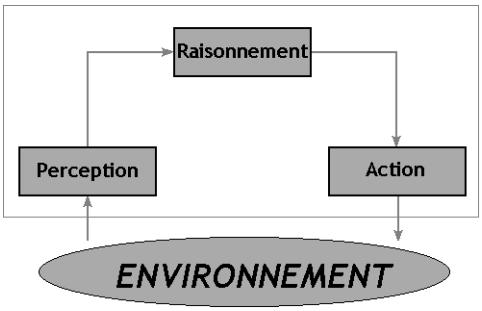


FIG. 2.1 – *Fonctionnement d'un agent*

   Cette définition aborde une notion essentielle : l'autonomie. En effet, ce concept est au centre de la problématique des agents. L'autonomie est la faculté d'avoir ou non le contrôle de son comportement sans l'intervention d'autres agents ou d'êtres humains. Une autre notion importante abordée par cette définition concerne la capacité d'un agent à communiquer avec d'autres.

2. Selon Yves Demazeau [Demazeau and Costa, 1996], un agent est une entité réelle ou virtuelle dont le comportement est autonome, évoluant dans un environnement qu'il est capable de percevoir et sur lequel il est capable d'agir, et d'interagir avec les autres agents.
   Cette définition introduit l'interaction[1] qui, comme nous le verrons par la suite, est le moteur des systèmes multi-agents. En effet, l'interaction suppose la présence d'agents capables de se rencontrer, de communiquer, de collaborer et d'agir.

1. Dans les systèmes multi-agents, ce terme englobe la communication (l'échange d'informations) et l'action sur le monde

3. Pour Mickael Wooldridge [Wooldridge, 1999], un agent est un système informatique capable d'agir de manière autonome et flexible dans un environnement.
   Par flexibilité on entend :
   - Réactivité : un système réactif maintient un lien constant avec son environnement et répond aux changements qui y surviennent.
   - Pro-activité : un système pro-actif (aussi appelé téléonomique) génère et satisfait des buts. Son comportement n'est donc pas uniquement dirigé par des événements.
   - Capacités sociales : un système social est capable d'interagir ou coopérer avec d'autres systèmes.

Ces définitions sont le résultat de différentes approches. Il existe en fait plusieurs type d'agents, qui selon les capacités qu'ils possèdent, seront qualifiés de réactifs, cognitifs ou hybrides.

Capacités cognitives. Les agents à *capacités cognitives* proviennent d'une métaphore du modèle humain. Ces agents possèdent une représentation explicite de leur environnement, des autres agents et d'eux-mêmes. Ils sont aussi dotés de capacités de raisonnement et de planification ainsi que de communication. Ces agents sont structurés en société où il règne donc une véritable organisation sociale. Le travail le plus représentatif de cette famille d'agent porte sur le modèle BDI (Believe Desir Intention) [Rao and Georgeff, 1995]. Les sources de ces travaux sont les sciences humaines et sociales.

Capacités réactives. Les agents à *capacités réactives* ne possèdent pas de moyen de mémorisation et n'ont pas de représentation explicite de leur environnement : ils fonctionnent selon un modèle stimuli/réponse. En effet, dès qu'ils perçoivent une modification de leur environnement, ils répondent par une action programmée. L'exemple le plus célèbre est celui de la fourmilière étudié par Alexis Drogoul [Drogoul, 1993]. Les sources des travaux sur ce type d'agents sont les sciences de la nature et de la vie.

Modèle d'agent hybride. Les agents *hybrides* sont des agents ayant des capacités cognitives et réactives. Ils conjuguent en effet la rapidité de réponse des agents réactifs ainsi que les capacités de raisonnement des agents cognitifs. Cette famille regroupe donc des agents dont le modèle est un compromis autonomie/coopération et efficacité/complexité. Pour illustrer cette famille, nous pouvons citer l'architecture ASIC [Boissier and Demazeau, 1996] utilisée pour le traitement numérique d'images, l'architecture ARCO [Rodriguez, 1994] créé dans le cadre de la robotique collective et l'architecture ASTRO [Occello and Demazeau, 1998] développée pour être utilisée dans les systèmes multi-agents soumis à des contraintes de type temporelles.

2.1.2 Niveaux de description d'un agent

Lorsque nous définirons les systèmes multi-agents nous parlerons de modèle d'agents, d'architecture d'agents et d'implémentation d'agents. Il est donc nécessaire d'expliciter ces termes qui constituent les différents niveaux de description d'un agent qui sont [Wooldridge and Jennings, 1995]:

- Le *modèle* qui décrit comment l'agent est compris, ses propriétés et comment on peut les représenter.
- L'*architecture* qui est un niveau intermédiaire entre le modèle et le contrôle et l'implémentation. Elle précise la création du système c'est-à-dire les propriétés qu'il doit posséder conformément au modèle et les liaisons avec les autres agents.

- L'*implémentation* qui s'occupe de la réalisation pratique de l'architecture des agents à l'aide de langages de programmation.

2.2 Les systèmes multi-agents

Nous avons vu précédemment que les agents fonctionnaient dans des environnements qui peuvent être réels ou virtuels. Ces environnements peuvent permettre à des agents de se rencontrer et donner ainsi naissance à des communications ou encore des interactions. La communication consiste à envoyer des messages ; l'interaction, quant à elle, permet d'aller plus loin et d'établir des conversations structurées entre agents. C'est donc, comme le souligne Jacques Ferber [Ferber, 1995], une mise en relation dynamique de plusieurs agents par le biais d'actions réciproques qui influenceront le comportement futur des différents intervenants.

2.2.1 Définitions

Système multi-agents. Un système multi-agents est un ensemble d'agents qui évoluent dans un environnement commun. Dans [Weiss, 1999], Gerhard Weiss définit l'intelligence artificielle distribuée comme étant l'étude, la conception et la réalisation de systèmes multi-agents qu'il présente comme étant des systèmes dans lesquels des agents intelligents interagissent et poursuivent un ensemble de buts ou réalisent un ensemble d'actions.

On peut donc se demander comment, à partir d'un ensemble d'agents pas forcément intelligents, on arrive à un système complexe où semble se dégager de l'intelligence. Dans un système multi-agents l'intelligence provient de l'émergence d'un comportement global [Gasser, 1992] comme nous le verrons dans le paragraphe suivant.

Les systèmes multi-agents, depuis leur création, font l'objet d'un véritable engouement. Les raisons exprimées dans [Bourdon, 2001] indiquent qu'il ne s'agit pas simplement d'un phénomène passager. En effet, l'importance de l'approche *système multi-agents* croît considérablement et s'explique essentiellement par :

- Le fait que les systèmes d'information soient de plus en plus distribués, ouverts, à grande échelle et hétérogènes. Cela rend les interconnexions tellement compliquées et croissantes qu'elles dépassent la compréhension globale que peut en avoir un être humain.
- Le fait que cette approche soit utile pour expérimenter des réflexions sociologiques et psychologiques pour ce qui concerne par exemple les relations entre personnes dans les sociétés modernes.

Système multi-agents ouvert. Un système multi-agents *ouvert* [Vercouter, 2000] partage les caractéristiques des systèmes ouverts. Pour ce qui concerne les entités élémentaires du système que sont les agents, ils n'ont pas la possibilité d'avoir une représentation complète de l'environnement. Pour ce qui est du système dans sa globalité, il doit être modulaire et extensible. La modularité concerne le fait que le système multi-agents sont composé de plusieurs sous-systèmes mis en relation. Ces sous-systèmes ont chacun leur propre mode de fonctionnement. L'extensibilité se traduit par le fait que le système multi-agents supporte l'ajout et le retrait dynamique d'éléments.
A l'inverse, le qualificatif *fermé* signifie que l'ensemble des agents qui compose le système reste le même.

Système multi-agents homogène/hétérogène. Un système multi-agents *homogène* est composé d'agents homogènes. Deux agents ont cette particularité s'ils sont identiques du point de vue de leur modèle et de leur architecture. Le qualificatif *hétérogène* est utilisé pour préciser que le système multi-agents est composé d'agents différents du point de vue de leurs modèles et de leurs architectures.

2.2.2 L'émergence

Comme nous l'avons constaté précédemment, un système multi-agents est un ensemble d'agents situés dans un même environnement et qui interagissent. De ces interactions peuvent émerger différents phénomènes que l'on peut classer en trois catégories [Marcenac, 1996] : l'émergence structurelle, l'émergence de comportements et l'émergence de propriétés. L'émergence traite donc de l'apparition soudaine non programmée et irréversible de phénomènes dans un système (éventuellement multi-agents) confirmant ainsi la maxime grecque "le tout est plus que la somme de chaque partie".

Il est difficile de qualifier la caractéristique émergente d'un phénomène, cependant Jean-Pierre Müller en propose une définition intéressante [Müller and Van Dyke Parunak, 1998]. S'inscrivant dans le prolongement des travaux relatés dans [Lenay, 1996, Jean, 1997], il affirme qu'un phénomène est émergent si :

- on a un ensemble d'agents interagissant via un environnement dont l'état et la dynamique ne peuvent pas être exprimés dans les termes du phénomène émergeant à produire mais dans un vocabulaire ou une théorie D,
- la dynamique des agents en interaction produit un phénomène global comme par exemple une trace d'exécution ou un invariant.
- Le phénomène global est observable soit par les agents (sens fort) soit par un observateur extérieur (sens faible) en des termes distincts de la dynamique sous-jacente D c'est-à-dire un autre vocabulaire ou une théorie D'.

Pour fournir à un système multi-agents une fonctionnalité particulière, on peut donc avoir recours à une autre méthode que celle utilisée traditionnellement et qui consiste à procéder à une décomposition fonctionnelle du problème en un ensemble de primitives qui seront implantées dans les agents. L'alternative proposée par Luc Steels [Steels, 1990] a pour objectif de faire émerger cette fonctionnalité à partir des interactions entre les agents (ou sous-systèmes multi-agents dans le cadre de système multi-agents récursif, c'est-à-dire composé de plusieurs système multi-agents). Les avantages de l'approche "fonctionnalité émergente" sont tout d'abord un renforcement de la robustesse du système : il est moins sensible aux changements de l'environnement. La raison est que contrairement au cas d'une fonctionnalité programmée (approche traditionnelle), il n'est pas nécessaire que le concepteur envisage toutes les possibilités afin qu'à chaque situation, le système ait une réaction que lui-même a jugé appropriée. D'autre part, le fait que justement, dans une approche "fonctionnalité émergente", on n'ait pas à envisager tous les cas de figure, peut simplifier l'analyse.

2.2.3 La décomposition multi-agents

Nous adopterons la vision d'un système multi-agents introduite par Yves Demazeau et communément admise aujourd'hui [Demazeau, 1995]. La décomposition Voyelles identifie un axe **A**gent, un axe **E**nvironnement, des **I**nteractions et une structure **O**rganisationnelle explicite ou non.

$$SMA = Agent + Environnement + Interaction + Organisation \tag{2.1}$$

De plus l'intelligence d'un système multi-agents étant plus que la somme de l'intelligence des agents qui la compose, un principe a été énoncé pour rendre compte du surcroît d'intelligence observé, de l'intelligente émergente.

$$Fonction(SMA) = \sum fonction(Agent) + fonction\ collective \tag{2.2}$$

Enfin, un système multi-agents peut être vu comme un agent d'un système multi-agents global. Il s'agit du principe de la récursivité.

$$Recursion : A = A_{élémentaire}/SMA \tag{2.3}$$

2.2.3.1 Les agents (A)

Le concept d'agent a été défini en §2.1.

Cette partie de l'analyse AEIO regroupe les éléments nécessaires à la construction des agents à savoir leur modèle, leur architecture, leur implémentation...

2.2.3.2 L'environnement (E)

Dans un système multi-agents, on appelle environnement l'espace commun aux agents du système. Un environnement peut être [Russell and Norvig, 1995, Wooldridge et al., 2000]:

- *Accessible* si un agent peut, à l'aide des primitives de perception, déterminer l'état de l'environnement et ainsi procéder, par exemple, à une action. Si l'environnement est *inaccessible* alors il faut que l'agent soit doté de moyens de mémorisation afin d'enregistrer les modifications qui sont intervenues.
- *Déterministe*, ou non, selon que l'état futur de l'environnement ne soit, ou non, fixé que par son état courant et les actions de l'agent.
- *Episodique* si le prochain état de l'environnement ne dépend pas des actions réalisées par les agents.
- *Statique* si l'état de l'environnement est stable (ne change pas) pendant que l'agent réfléchit. Dans le cas contraire, il sera qualifié de *dynamique*.
- *Discret* si le nombre des actions faisables et des états de l'environnement est fini.

Cette partie de l'analyse consiste donc à trouver les éléments nécessaires à la réalisation des interactions extérieures au système comme par exemple la perception de cet environnement, les actions que l'on peut y faire.

2.2.3.3 Les interactions (I)

Les interactions proviennent de la mise en relation dynamique de plusieurs agents par le biais d'un ensemble d'actions réciproques. Il existe plusieurs types d'interactions, qui dépendent de trois paramètres que sont les buts, les ressources et les compétences comme l'illustre le tableau référencé 2.1 tiré de l'ouvrage de Jacques Ferber [Ferber, 1995].

Illustrons ces différentes situations d'interactions par des exemples [Bourdon, 2001]:

- L'*indépendance* est une situation d'interaction similaire à celle de deux personnes qui se rencontrent dans une rue assez large pour qu'elles puissent passer en même temps.
- La *collaboration simple*

BUTS	RESSOURCES	COMPETENCES	TYPE DE SITUATION	CATEGORIE
Compatibles	Suffisantes	Suffisantes	Indépendance	Indifférence
		Insuffisantes	Collaboration simple	Coopération
	Insuffisantes	Suffisantes	Encombrement	
		Insuffisantes	Collaboration coordonnée	
Incompatibles	Suffisantes	Suffisantes	Compétition individuelle pure	Antagonisme
		Insuffisantes	Compétition collective pure	
	Insuffisantes	Suffisantes	Conflits individuels pour des ressources	
		Insuffisantes	Conflits collectifs pour des ressources	

TAB. 2.1 – *Types d'interactions d'après [Ferber, 1995]*

- Pour illustrer l'*encombrement* on peut prendre l'exemple du trafic aérien.
- Le cas d'un système intégrant un mécanisme de régulation distribuée ou le cas d'une société de robots autonomes peuvent illustrer ce qu'on appelle la *collaboration ordonnée*.
- La course à pied illustre bien ce qu'est une *compétition individuelle pure*.
- Une épreuve tel que le relais 4*100m illustre bien la *compétition collective pure* car les équipes ne se gênent pas.
- Les *conflits individuels pour des ressources* sont représentables par le cas d'animaux se battant pour leur territoire.
- Les coalitions sont des exemples de *conflits collectifs pour des ressources*.

Cet axe de l'analyse AEIO comprend donc les éléments qui servent à la structuration des interactions externes entre les agents (langage d'interaction, protocole de communications...).

2.2.3.4 L'organisation (O)

Le concept d'organisation est vaste mais on peut tout de même le définir comme étant une structure décrivant les interactions et autres relations qui existent (dans le but d'assouvir un objectif commun) entre les membres de la dite organisation [Fox, 1981]. L'organisation peut donc apparaître comme une structure de coordination et de communication [Malone, 1987]. Jacques Ferber [Ferber, 1995] va dans ce sens en affirmant que les organisations constituent à la fois le support et la manière dont se passent les inter-relations entre les agents, c'est-à-dire dont sont réparties les tâches, les informations, les ressources et la coordination d'actions. Il précise que ce qui rend l'organisation si difficile à cerner est qu'elle est à la fois le processus d'élaboration d'une structure et le résultat de ce processus. Cette citation met en avant la dualité du terme organisation : en effet, il y a l'aspect statique et l'aspect dynamique. Dans [Hübner et al., 2002] l'organisation est décrite via une spécification structurelle, une spécification fonctionnelle et une spécification déontique.

L'aspect dynamique, appelé aussi plus simplement organisation, est relatif aux relations entre les éléments du système, qui sont soumis à des changements dynamiques. L'aspect statique, appelé structure organisationnelle provient de la raison d'être des organisations qui est la nécessité pour des individus de se regrouper pour repousser leurs propres limites, en terme de capacités [March and Simon, 1958]. Il existe de nombreuses structures organisationnelles [Baeijs and Demazeau, 1996]. On en dénombre essentiellement trois grandes familles (les groupes, les hiérarchies et les marchés) basées sur les trois processus de base de coordination [Le Strugeon, 1995, Agimont, 1996] étudiés par Mintzberg [Mintzberg, 1982] et qui sont :

- l'ajustement mutuel lorsqu'un ou plusieurs agents se mettent d'accord entre eux pour partager une ressource afin de réaliser un but qui leur est commun,
- la supervision directe qui fait intervenir une relation hiérarchique entre les agents : le gérant grâce à son contrôle sur les autres agents peut réguler la consommation faite par les agents qui sont soumis à son autorité,
- La standardisation où l'agent gérant, qui a autorité sur les autres, met en place des procédures à réaliser par les autres agents dans des cas concrets.

L'axe O de la méthode d'analyse AEIO comprend donc les éléments permettant d'ordonner les ensembles d'agents en des organisations par la détermination des rôles des agents.

2.3 Un peu d'histoire...

Pour conclure ce chapitre, il est intéressant d'observer la place qu'occupent les systèmes multi-agents dans l'évolution des paradigmes de l'informatique.

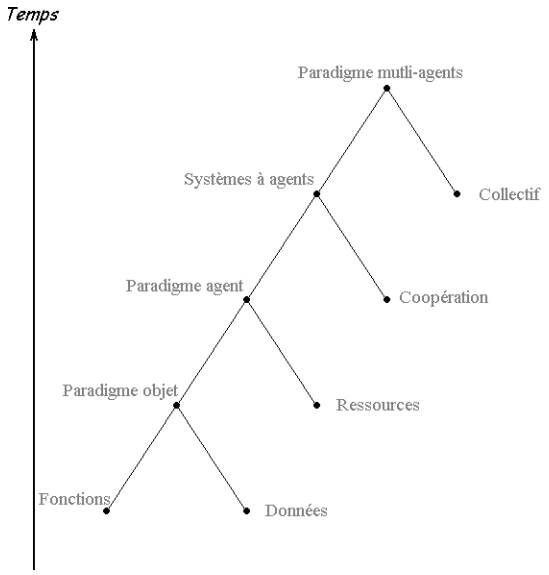


FIG. 2.2 – *Evolution des paradigmes de programmation*

Ce graphique tente de présenter un agencement dans le temps des concepts et à la fois le positionnement, tel que nous le voyons, du domaine des multi-agents. Il s'inspire de celui présenté dans l'ouvrage de génie logiciel multi-agents de Jürgen Lind [Lind, 2001] qui s'est arrêté à l'introduction du concept d'agent. Nous sommes allés plus loin en introduisant dans cet arbre les *systèmes à agents* pour lesquels

plusieurs agents sont plongés dans un même environnement afin de coopérer (voir 2.2.3.3). Le *paradigme multi-agents*, où le collectif permet la résolution du problème, est lui aussi ajouté au graphique initial.

Cette évolution des concepts est en fait très logique. Dans un premier temps l'objectif des personnes qui utilisaient l'outil informatique fut de simplifier la programmation : l'élaboration des langages dit évolués passa par l'agrégation d'instructions en procédures puis en fonctions lorsqu'elle retournait des résultats. Les besoins en dynamicité des applications puis en généricité a conduit à l'utilisation de programmation structurées puis à introduire les objets (méthodes et langage de programmation). Les besoins en autonomie, induis par la répartition et la distribution des ressources a conduit à introduire la notion d'agent (§2.1.1). Très peu de temps après, la coopération a amené les informaticiens à créer des systèmes à agents. La prise en compte d'objectif globaux et l'utilisation de l'émergence a conduit quant à elle à introduire le paradigme multi-agents.

Dans un système multi-agents, les agents offrent de la généricité de par leur découplage modèle/architecture/implantation (§2.1.2). Ce découplage permetra d'envisager plus simplement l'implantation sur des supports physiques variés.

La vision du collectif plutôt que de l'individu permet d'atteindre des objectifs globaux du système multi-agents (§2.2.2) dont il faudra garantir l'intégrité du fonctionnement global, notement dans le cas de système multi-agents ouvert (§2.2.1).

Conclusion

Nous avons défini, dans un premier chapitre, la notion de systèmes complexes physiques ouverts. Cette classe de systèmes se caractérise, d'une part, par la nature matérielle de ses composants, d'autre part, par la richesse des interactions mises en jeu. Les composants sont en effet en interaction entre eux, mais aussi avec l'environnement matériel du système. Ils peuvent le percevoir et agir sur lui, tout comme l'environnement agit sur eux. Nous avons identifié en particulier quatre catégories de systèmes complexes physiques : les systèmes de traitement (d'information et de signaux), les systèmes de communication, les systèmes de contrôle/commande et les systèmes interactifs.

Le paradigme multi-agents de part son inhérente décentralisation, sa prise en compte du global au niveau local, la richesse des interactions qu'elle peut opérationnaliser et la prise en compte du niveau social des composants, est efficace pour s'intéresser aux systèmes complexes. En effet, l'approche et la richesse des modèles développés dans ce domaine permet de s'intéresser aux systèmes complexes. Ces modèles sont, par exemple, liés à l'environnement, aux diverses interactions qui permettent entre autres la communication et la coopération, les organisations et aux agents.

Les systèmes multi-agents se sont attachés jusqu'ici essentiellement à traiter les problèmes complexes ouverts. Nous proposons dans la suite du manuscrit, de nous intéresser à la modélisation des systèmes complexes physiques ouverts à l'aide de systèmes multi-agents.

> Nous appellerons donc système multi-agents physique ouvert, ou système multi-agents embarqués, un système multi-agents qui modélise un système complexes physique ouvert. Les agents de ce système seront donc des systèmes mixtes logiciel/matériel et évolueront dans des environnements physiques (le monde réel).

Cette modélisation génère des besoins méthodologiques nouveaux (comment concevoir de tels systèmes?). De très nombreux travaux [Iglesias et al., 1998, Tveit, 2001, Picard, 2004] se penchent sur la construction de méthodes multi-agents mais aucune n'intègre réellement la prise en compte d'aspects physiques. Il ne s'agit donc pas alors d'améliorer des méthodes existantes, mais de se poser la question du cycle de développement le plus adapté pour une méthode de développement de système multi-agents embarqués?".

Les systèmes multi-agents embarqués se traduisent généralement par des systèmes d'entités physiques autonomes utilisant des communications sans fil. Il est donc indispensable, au delà des aspects méthodologiques, d'étudier ces besoins architecturaux spécifiques (quelle architecture pour assurer une communication fiable entre les agents?) afin de les intégrer à l'approche et de proposer les outils nécessaires.

deuxième partie

DIAMOND : UNE APPROCHE POUR LA CONCEPTION DES SYSTEMES COMPLEXES PHYSIQUES OUVERTS

Introduction

Les motivations de ce travail soulèvent de nombreuses questions notamment au niveau méthodologique : Quel cycle de vie pour construire des systèmes complexes physiques ouverts? Comment spécifier un système multi-agents mixte logiciel/matériel? Comment harmoniser au mieux, dans ce cycle de vie, les spécifications du logiciel et les spécifications matérielles? La spécificité des systèmes multi-agents physiques va t'elle demander de procéder à une analyse des besoins particulière? Comment intégrer au mieux les contraintes locales des éléments physiques aux fonctionnalités globales du système? Comment concevoir les parties logicielles, matérielles et les intégrer en un tout fonctionnel? Hormis pour les effecteurs et les capteurs, comment choisir ce qui doit être logiciel ou matériel? Comment tester un système multi-agents physique? Comment débugger un système multi-agents physique? Quelle démarche qualité pour la conception de tels systèmes?

C'est donc à ce premier aspect de la problématique que nous nous attachons dans cette partie.

Nous commencerons par rappeler les fondements des procédés traditionnels de production de logiciels. Ce chapitre a pour vocation de préciser un vocabulaire et insiste particulièrement sur les cycles de vie. Le choix d'un cycle de vie est effectivement important car il conditionne toute démarche. Nous présentons aussi succinctement les quatre familles de techniques de spécification et abordons la notion de qualité logicielle.

Nous nous intéressons, dans le chapitre suivant, à la conception des systèmes hybrides matériels/logiciels. Nous introduisons en particulier une nouvelle famille de méthode qui tente d'unifier la conception du logiciel et du matériel dans un seul et unique cycle de vie : les méthodes dites co-design ("concurent hardware/software"). Nous présentons, sous forme d'un tableau, la synthèse d'une revue de méthodes de co-design qui ont été éprouvées et discutons les techniques de spécification utilisées et leurs cibles.

Dans un dernier chapitre, nous développons notre contribution méthodologique, une méthode complète d'analyse et de conception, que nous appelons DIAMOND (Decentralized Iterative Approach for Multiagent Open Networks Design). Les activités de notre méthode sont agencées en un cycle de vie en spirale et organisées en quatre phases : le recueil des besoins (et leur structuration), la phase d'analyse, la phase de conception générique logicielle/matérielle ainsi que la phase d'implantation. En fin de chapitre, nous proposons une discussion comparative de DIAMOND avec les autres méthodes multi-agents non spécifiques.

Chapitre 3

Les procédés traditionnels de production de logiciels.

L'observation recueille les faits; la réflexion les combine; l'expérience vérifie le résultat de la combinaison.

Denis Diderot

Ce chapitre présente les notions de base sur la production de logiciel. Nous nous appuierons sur ces notions pour introduire le co-design dans le prochain chapitre et pour développer notre méthode.

3.1 Généralités

Le génie logiciel regroupe les différents éléments nécessaires au bon déroulement de la production des logiciels. A l'origine son objectif est de rendre les logiciels à la fois plus fiables et plus rapides à réaliser. Pour cela il englobe à la fois les aspects techniques de la conception des logiciels mais aussi les aspects humains : commanditaire, équipe de développement, utilisateur etc. Par la suite, le génie logiciel eut pour objectif l'amoindrissement des coûts de production et, plus récemment, la diminution des difficultés d'évolution du logiciel.

Le procédé de production d'un logiciel, que l'on appelle aussi processus de développement, donne une grande importance à l'analyse des besoins du client, à la conception et à la validation. Le développement du logiciel consiste généralement à établir une suite de descriptions de plus en plus proches d'un programme final et de sa documentation (approche par raffinements successifs). Ce processus doit connaître une réelle traçabilité : elle se traduit par la production de documents intermédiaires qui permettent de vérifier le bon déroulement du développement du logiciel.

Un logiciel, de sa naissance à sa mort, passe par un ensemble d'étapes qui concernent des points aussi divers que la faisabilité, la spécification, la production, le contrôle, les tests, le contrôle de production, l'utilisation du logiciel, sa vente et sa maintenance. Ces différentes étapes font intervenir plusieurs *processus* que l'on peut décomposer en *activités*.

Nous appellerons *étape* un ensemble de *processus* organisés dans le temps.

Nous appellerons *processus,* ou *phase,* un ensemble d'*activités* organisées dans le temps.

Nous appellerons *activité,* une action primitive entreprise dans le développement d'un produit logiciel.

L'agencement de ces étapes constitue le cycle de vie. Les relations entre les différentes activités, entre les différents processus, entre les différentes phases, entre activités et processus, et entre les processus et phases dépendent du *modèle de cycle de vie* utilisé.

Nous appellerons *cycle de vie* un ensemble ordonné dans le temps d'*étapes* de développement d'un produit logiciel.

Nous appellerons donc *méthode* une *démarche* ordonnée et raisonnée mettant en oeuvre des *concepts* et des *outils* permettant la production de modèles.

Une méthode regroupe donc :

1. Une *démarche* sous entendue méthodologique qui consiste en un ensemble ordonné de phases et d'étapes qui sont elles-mêmes décrites en termes d'activités de production. La démarche a pour but de guider le travail pour arriver à un résultat autrement dit à un livrable. La démarche permet donc de suivre le projet et fournit une assurance qualité. A une démarche est associé un cycle de vie. Nous insistons sur les démarches en §3.3 et sur les cycles de vie en §3.4,
2. Des *concepts* qui permettent de créer des modèles suffisamment explicites pour que l'on en ait une représentation unique. Un modèle est une abstraction d'une entité réelle définie en [Boughlam, 1988] comme étant *une image de la réalité, expression d'un problème que nous cherchons à appréhender, représentée en des termes que nous pensons comprendre*. Il est utilisé afin de simplifier la compréhension de cette dernière dans le but, généralement, de pouvoir la (re)construire. Généralement les concepts mettent en oeuvre des notations (voir §3.5).
3. Des *outils* qui permettent la production de modèles et autres diagrammes propres, lisibles, facilement modifiables, la vérification de leurs syntaxes, leurs cohérences et de leurs complétudes. Parmi les outils on compte les simulateurs qui permettent une vérification et une validation des modèles d'analyse.

Nous appelerons *outils* un logiciel qui permet de produire de réaliser des fonctions élémentaires comme la production d'un modèle, sa vérification etc.

Nous appelerons *environnement* un ensemble cohérent d'outils couvrant l'ensemble du cycle de vie et permettant le développement d'applications.

3.2 Les processus

Un processus peut regrouper plusieurs activités ou se résumer à une activité. Parmi les différents familles d'activités que compte le développement d'un logiciel on peut identifier [Alissali, 1998] :

– L'*analyse des besoins* a pour objectif de cerner et de mieux comprendre les besoins qui ont né-

cessité la création du logiciel, de répondre aux questions "Que doit-on faire et qui utilisera le produit ?". Il est nécessaire de produire un document qui décrit le domaine d'application, l'état actuel de l'environnement du futur système, le rôle du système, les ressources disponibles et requises, les contraintes d'utilisation, les performances attendues etc. Pour cela il faut établir un véritable dialogue entre les experts du domaine et les informaticiens qui réaliseront l'application. Il est à noter que ce processus est lié à l'étude de faisabilité et à la planification.

– La *spécification globale* ou *conception préliminaire* est une description de haut niveau (en modules) du futur produit informatique qui a pour objectif d'asseoir la confiance en la finalité et la faisabilité du produit, et, en second lieu, servir de base pour l'estimation de la quantité de travail à fournir pour le réaliser. Le point de départ de ce processus est le résultat de l'analyse des besoins. En effet, en y associant différentes considérations techniques et la faisabilité informatique, on pourra produire une description de ce que doit faire le système. Il est à noter que l'on s'intéresse à ce que devra faire l'application sans pour autant se demander comment doivent être faites les choses.
– La *conception détaillée* a pour objectif d'enrichir la description du logiciel de détails d'implémentation afin d'aboutir à une description très proche d'un programme. Chacun des modules énumérés dans le dossier de conception préliminaire doivent être décrit en détail : interfaces, fonctions et services proposées. Chacun de ces modules doit avoir un plan de test unitaire qui donnera aux réalisateurs la liste des tests à effectuer ou des types de scénarios de test à créer afin de vérifier que le module répond aux spécifications. De ce processus doit aussi apparaître une estimation précise de la charge de travail induite par la réalisation de chacun des modules.
– L'*implémentation* a pour but la réalisation, à partir de la conception détaillée, des modules décrit dans le document de conception détaillée. On peut considérer qu'un module a été réalisé quand il a été créé (programmé), testé et utilisé avec succès par un autre module ou par un processus de test au niveau système. Le test des modules comprend les tests au niveau unitaire, les tests de non-régression définis lors de la conception détaillée, ainsi que les tests de performance et de charge et l'analyse de couverture du code.
– La *gestion de configurations* et l'*intégration* prennent en charge respectivement l'évolution ainsi que la mise à jour des modules qui composent l'application tout au long du processus de développement et de la réalisation d'un ou de plusieurs systèmes exécutables à partir des différents composants. L'intégration peut être réalisée de façon incrémentale, en parallèle avec la réalisation de différents modules.
– La *validation* et la *vérification* ont pour but respectivement de vérifier que l'on a décrit le système que souhaitait réellement l'utilisateur (on valide les spécifications, les manuels et le prototypage rapide) et de vérifier que le développement est correct par rapport à la spécification globale (on s'assure que les descriptions successives et que le logiciel lui-même satisfont la spécification globale ; elle inclut des inspections de spécifications et de programmes ainsi que de la preuve et les différents tests). En d'autres termes la validation s'assure que le système met correctement en application les spécifications produites lors de la phase d'analyse tandis que la vérification s'occupe de la réalisation des tests pour s'assurer de la conformité du système produit par rapport à la conception.
– Le *maquettage* ou *prototypage rapide* consiste à développer rapidement une ébauche du futur système. Son rôle peut être exploratoire si il est utilisé pour préciser les besoins des utilisateurs, ou expérimental si il intervient lors de la conception pour aider à expérimenter différents choix.
– La *maintenance* et l'*assistance* s'occupent respectivement de la correction des défauts du logiciel

rencontrés soit pendant la phase de test soit après la diffusion du logiciel et de l'aide à fournir à l'utilisateur en plus des différentes documentations. Il est important qu'il y ait une traçabilité des défauts, qu'ils soient enregistrés dans un système de suivi.

3.3 Les démarches de conception

Les méthodes d'analyse et de conception fournissent des notations standards et des conseils pratiques qui permettent d'aboutir à des conceptions "raisonnables". Il existe différentes manières pour classer ces méthodes, dont :

- la distinction fonctionnelle (dirigée par le traitement). Dans la stratégie fonctionnelle un système est vu comme un ensemble d'unités en interaction, ayant chacune une fonction clairement définie. Les fonctions disposent d'un état local, mais le système a un état partagé, qui est centralisé et accessible par l'ensemble des fonctions. Les stratégies orientées objet considèrent qu'un système est un ensemble d'objets interagissant. Chaque objet dispose d'un ensemble d'attributs décrivant son état et l'état du système est décrit (de façon décentralisée) par l'état de l'ensemble.
- la distinction composition/décomposition qui met en opposition d'une part les méthodes ascendantes qui consistent à construire un logiciel par composition à partir de modules existants et, d'autre part, les méthodes descendantes qui décomposent récursivement un système jusqu'à arriver à des modules programmables "simplement".

Il existe plusieurs types d'approches basées sur différents types d'analyse.

3.3.1 Les approches fonctionnelles

L'objectif de ces approches est de découper la complexité d'un tout en éléments simples afin de mettre en évidence les fonctions à assurer et proposent une approche hiérarchique descendante et modulaire. Elles trouvent leur origine dans les langages procéduraux. Leur inconvénient est la mauvaise prise en compte de la dynamique des systèmes et de leur évolution. Cette analyse permet, en effet, d'identifier clairement le fonctionnement d'un mécanisme précis mais ne permet pas la compréhension de processus complexes.

Ces approches utilisent intensivement les raffinements successifs pour produire des spécification dont l'essentiel est sous forme de notation graphique en diagrammes de flots de données. Le plus haut niveau représente l'ensemble du problème (sous forme d'activité, de données ou de processus, selon la méthode). Chaque niveau est ensuite décomposé en respectant les entrées/sorties du niveau supérieur. La décomposition se poursuit jusqu'à arriver à des composants maîtrisables (voir figure 3.1).

Parmi les méthodes les plus répandues, qui reposent sur cette approche, on dénombre la méthode d'Analyse Structurée SADT (System Analysis and Design Technic [Marca and McGowan, 1987]) dont le plus haut niveau est appelé diagramme de contexte. Une boîte de diagramme de flots de données représente un processus et doit être décomposée. Chaque processus (ou traitement) non décomposé est décrit par une " mini-spécification " ; un dictionnaire précise la définition des données, processus et zones de stockage. Par exemple, la méthode SADT permet de produire un modèle du logiciel sous forme d'une suite cohérente et hiérarchisée de diagrammes obtenues par raffinements successifs.

L'analyse structurée n'étant pas suffisante pour exprimer les contraintes de temps et de synchronisation, des extensions ont été apportées à cet effet notamment dans la méthode SART (Structured Analysis

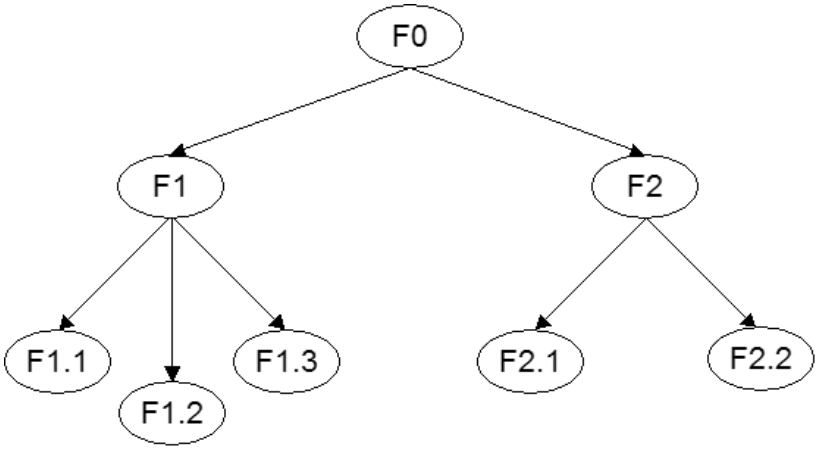


FIG. 3.1 – *Approche fonctionnelle*

for Real-Time systems [Ward, 1985]):

- ajout des diagrammes de flots de contrôle et spécifications de contrôle : information d'activation/désactivation des processus ;
- utilisation des diagrammes états/transitions.

D'autres méthodes références de ce domaine sont, par exemple, JSD [McNeile, 1986], MASCOT [Jackson and Simpson, 1975] et RAPID/USE [Wasserman et al., 1985]. Toutes ces méthodes proposent de modéliser le logiciel avec toutes ou partie des vues suivantes :

- flux de données : modélisation des transformations des données.
- entité-association (relation) : structure logique des données.
- vue structurée : documentation des composantes du système ainsi que leurs relations.

3.3.2 L'approche systémique

Elle complète la démarche cartésienne en se concentrant sur les liaisons entre les différents éléments constituant du système. L'analyse systémique consiste à regarder la réalité à travers des modèles proposés par la théorie des systèmes. Ces modèles sont généralement centrées sur les organisations. Merise est un bon représentant ce cette famille de méthodes. L'inconvénient majeur de cette analyse est qu'elle n'est pas objective : elle dépend beaucoup des convictions de la personne qui modélise le problème.

La figure 3.2 représente la dichotomie données/traitements de cette approche. Le modèle conceptuel de données (MCD) précise les informations sans prendre en compte les contraintes techniques et économique. Le modèle logique de données (MLD) décrit les données en fonction des conditions d'utilisation par les traitements. Le modèle physique de données (MPD) décrit la base de données. Le modèle conceptuel de traitement (MCT) décrit l'activité du domaine sans préciser les ressources et les organisations. Le modèle organisationnel de traitements (MOT) décrit le fonctionnement du domaines en intégrant les contraintes liées a l'environnement. Le modèle opérationnel des traitement (MPT) décrit l'architecture technique des programmes.

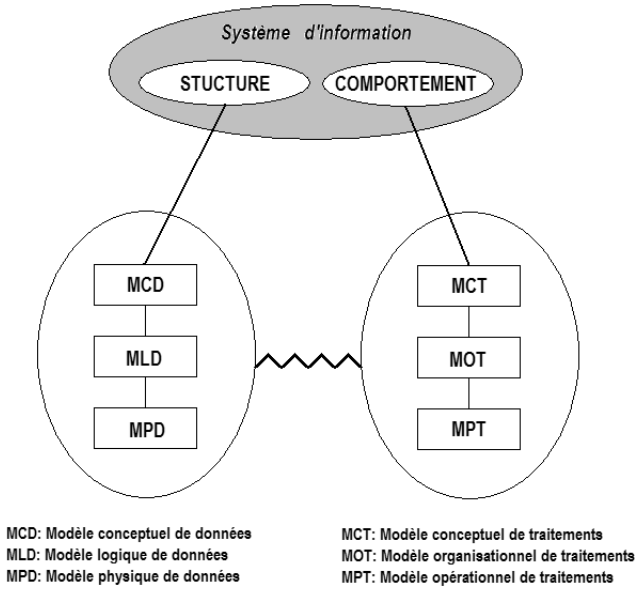


FIG. 3.2 – *Approche systémique: dichotomie données/traitements*

3.3.3 L'approche objet

L'approche objet permet d'analyser un système en mettant l'accent sur ce qu'il fait comme dans les méthodes fonctionnelles. Cependant, se faisant, elle tient compte de ce qu'est le système en terme de données et de composants (les objets). Les fonctions et les données sont encapsulées dans les objets comme l'illustre la figure 3.3. Cette approche s'intéresse particulièrement à la composition du système (en terme d'objet) se concentrant sur ce que manipule le système plutôt que ce qu'il fait. Le RUP (Rational Unified Process, [Kruchten and Kroll, 2003]), qui est le processus de développement associé à UML, est un bon représentant de ce type d'approche.

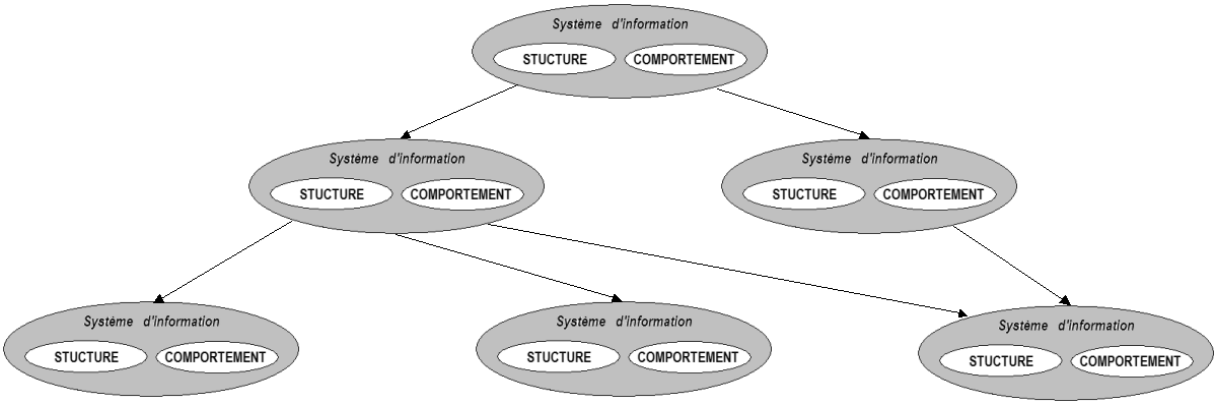


FIG. 3.3 – *Approche objet*

3.4 Les cycles de vie

De la naissance de l'idée jusqu'à livraison à l'utilisateur, le logiciel évolue par étapes : le cycle de vie regroupe ces étapes clairement identifiables et les agence dans le temps. Il permet de définir le vocabulaire employé et de planifier les activités. Les modèles de cycle de vie s'appliquent non seulement au produit mais aussi aux étapes (voir figure 3.4).

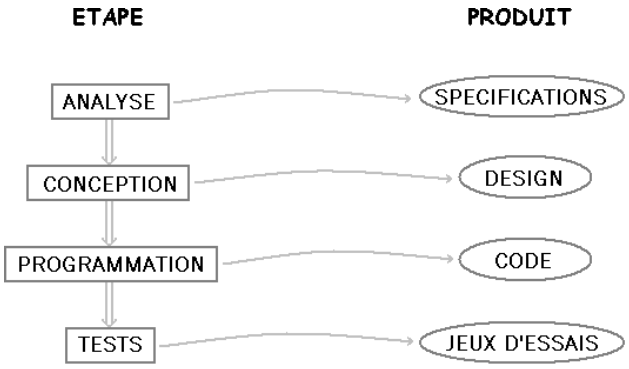


FIG. 3.4 – *Cycle de vie "minimal" et étape/produit*

Cette partie énonce les grands types de modèle de cycle de vie sans entrer dans le détail des documents à produire et des liens étapes/activités.

3.4.1 Le code-and-fix

Ce modèle ne présente pas de grand intérêt si ce n'est historique : il est le premier modèle de cycle de vie à avoir été identifié. Il n'est constitué que de deux phases que l'on réitère! La première consiste à produire du code et la seconde à tester le logiciel ainsi produit. La réalisation d'une application nécessitait d'itérer plusieurs fois. Le logiciel produit devenait très difficile à comprendre et surtout à maintenir.

3.4.2 Le modèle de la cascade

Dans ce modèle [Boehm, 1976] les activités sont représentées dans des processus séparés (voir figure 3.5).

Un principe très simple qui consiste à produire certains documents dès qu'une phase se termine est adopté. Les résultats sont définis sur la base des interactions entre étapes et activités et ils sont soumis à une revue approfondie : on ne passe à la phase suivante que s'ils sont jugés satisfaisants. On peut remarquer que certaines phases de ce modèle portent le nom d'activités. Cela est dû au fait que l'activité joue un rôle important pour cette phase. Certaines activités peuvent être présentes tout au long du processus : c'est le cas, par exemple, du contrôle technique et de la gestion de la configuration.

Le modèle en cascade originel ne permettait pas de "retour en arrière" comme on peut le voir sur notre représentation graphique du modèle. En effet, ce n'est qu'après certaines évolutions que cette possibilité a été ajoutée : le principe est qu'une étape ne peut remettre en cause que l'étape précédente. Ce modèle, qui a connu de nombreux développements, permet depuis peu de procéder à de la validation/vérification à chaque étape.

Ce modèle a été inspiré de la conception/production de matériel dans l'industrie et il est actuellement un des plus utilisés. Ainsi de nombreux environnements l'utilisent. Ces avantages de ce modèle sont

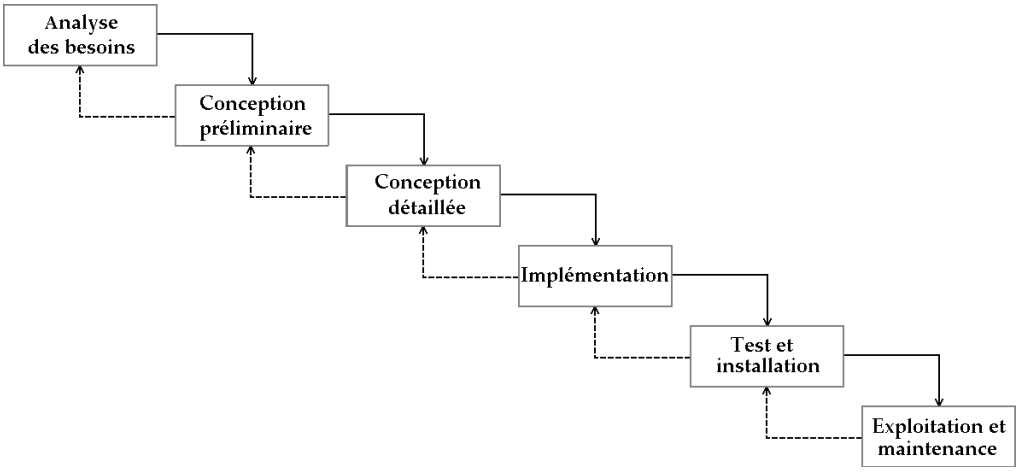


FIG. 3.5 – *Modèle en cascade*

de deux natures. D'un point de vue technique, le processus est clair et systématique. Il existe un grand nombre de méthodes pour toutes les étapes. Les informations sont très bien organisées, transmissibles et réutilisables. D'un point de vue gestion de projet, il fournit un cadre de référence pour la planification et le contrôle ainsi qu'une bonne visibilité des progrès et des résultats.

Il existe trois principaux inconvénients à ce modèle de cycle. Tout d'abord, les possibilités de contrôle sont parfois exploitées de manière abusive ce qui entraîne ce que d'aucun appelle des lourdeurs administratives. Le modèle, malgré les différentes améliorations qui y ont été apportées, supporte mal les retours en arrière. Cela se ressent tant du point de vue technique que du point de vue de la gestion de projet. Enfin, avant de commencer la conception il est nécessaire d'avoir des spécifications écrites et complètes (difficile pour les systèmes complexes ou en constante évolution).

Le génie logiciel travaille actuellement pour :

– Augmenter la part de l'effort consacré aux étapes d'analyse et de conception,
– Diminuer la part de l'effort consacré à la programmation,
– Diminuer le nombre d'erreurs à détecter et à corriger pendant les tests,
– Rationaliser l'étape de test.

3.4.3 Le modèle en b

Le modèle en b [Birrel and Ould, 1985] est une variante du modèle en cascade. Ce modèle a été élaboré en partant du constat que les grands systèmes nécessitaient beaucoup de maintenance. Cette dernière couvre environ les deux-tiers de modèle de cycle de vie. Le principe de base est la ré-utilisation de scénarios de tests (tests de non-régression, procédures de mise en exploitation) établis pendant la phase de développement. C'est en cela que réside d'ailleurs son principal intérêt : on prévoit, lors de la phase projet, la réutilisation des tests et des procédures de mise en exploitation.

3.4.4 Le modèle en V

Ce modèle est une variante du modèle en cascade qui insiste sur la symétrie et la relation entre les phases de début et de fin de cycle de vie. Le principe de ce modèle est donc qu'avec toute décomposition doit être décrite la recomposition, et que toute description d'un composant est accompagnée de tests

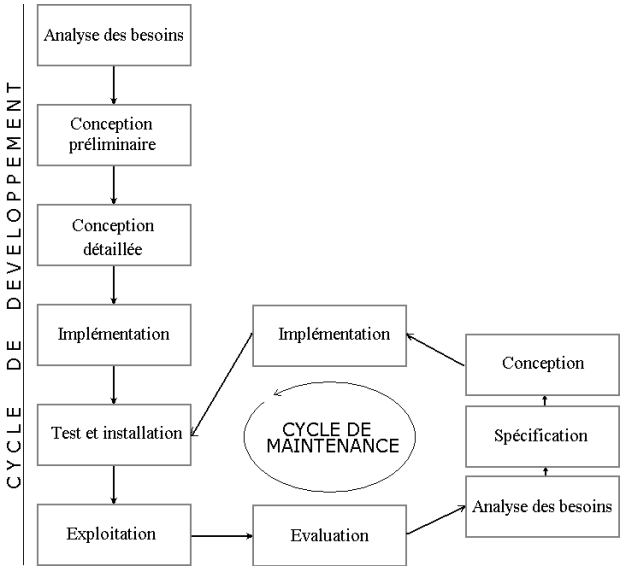


FIG. 3.6 – *Modèle en b*

qui permettront de s'assurer qu'il correspond à sa description. Ceci rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel), et permet ainsi d'éviter d'énoncer une propriété qu'il est impossible de vérifier objectivement après la réalisation.

Les deux sortes de dépendances que l'on distingue dans ce modèle (voir figure 3.7) sont l'*enchaînement* et l'*itération* et se déroulent essentiellement de gauche à droite. Le second type de dépendance est *la préparation des phases ultérieures* : ainsi une fois la conception architecturale finie, le protocole et les jeux de test de l'intégration doivent être complètement décrits.

Avec ce modèle, les activités de chacune des phases peuvent être classées en cinq catégories : l'assurance qualité, la production, le contrôle technique, la gestion et le contrôle de qualité.

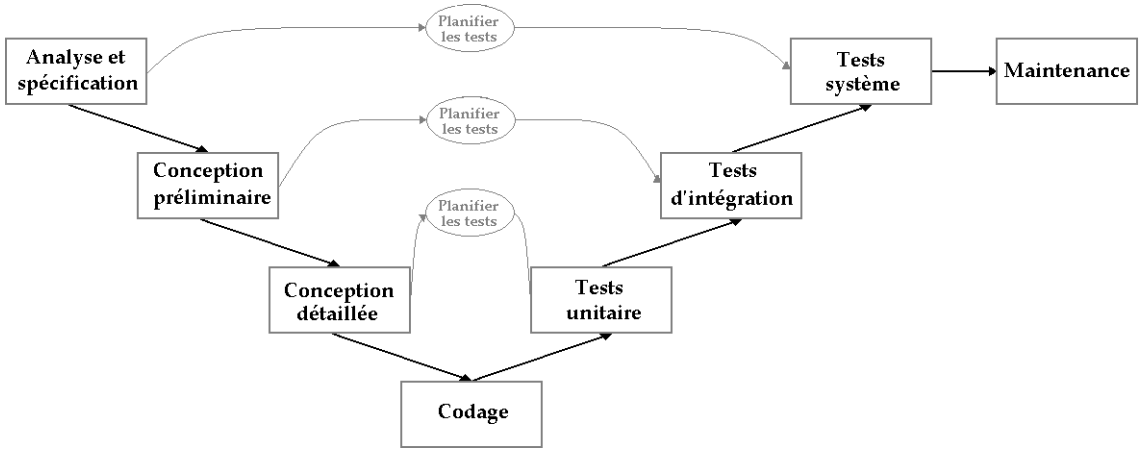


FIG. 3.7 – *Modèle en V*

3.4.5 Le modèle par développement évolutif ou prototypage

Avec ce type de modèle (voir figure 3.8, [Yeh, 1990]), un prototype initial est amélioré jusqu'à l'obtention d'un système définitif. Il est généralement utilisé pour comprendre les besoins de l'utilisateur : son but est de s'assurer de la faisabilité d'une application et de vérifier que les exigences de l'utilisateur sont prises en compte. Le prototype est reconstruit en tenant compte du feedback de l'utilisateur. Ce modèle est dit évolutif dans le sens où plusieurs prototypes sont développés (avec un minimum de fonctionnalités) et que le seul dont l'évolution continuera est celui retenu par l'usager. Actuellement, de nouvelles versions sont développées en utilisant le modèle en cascade.

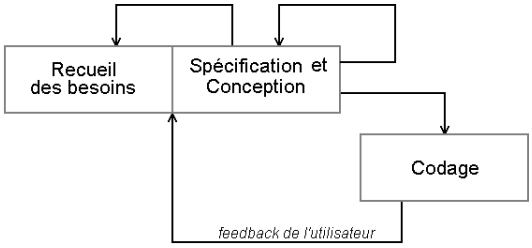


FIG. 3.8 – *Modèle par développement évolutif*

3.4.6 Le modèle par incrément

Ce modèle (voir figure 3.9) contrairement à la plupart des autres n'adopte pas une approche de décomposition en composants du logiciel avec une intégration en fin du processus. En effet, la particularité de ce modèle est que l'on procède par incréments : on développe un noyau de logiciel et on y intègre les différents incréments que l'on conçoit, chacun d'eux étant développé selon un modèle différent (généralement le modèle en cascade).

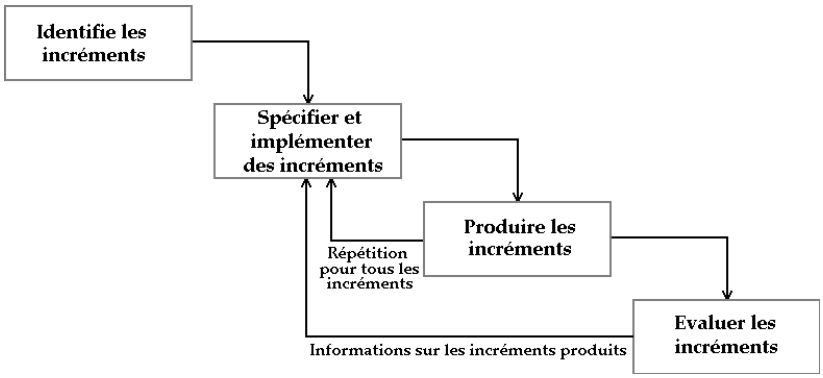


FIG. 3.9 – *Modèle incrémental*

Les avantages sont que chaque développement est moins complexe, que les intégrations sont progressives, qu'il est possible de livrer et de mettre en service le logiciel après chaque incrément. On constate aussi la présence d'un meilleur lissage du temps et de l'effort de développement car il est possible que les différentes phases se recouvrent.

Cependant des inconvénients subsistent. Il se peut que lors du processus on remette en cause les incréments précédents voire le noyau. Il se peut aussi qu'on ne puisse pas intégrer de nouveaux incréments.

Il est donc nécessaire que les noyaux, les incréments ainsi que leurs interactions soient décrites globalement en début du projet et, qu'une fois cette tâche effectuée, qu'on définisse des priorités associées à une répartition de leur développement dans le temps. Pour un bon déroulement du processus, il est nécessaire que les incréments soient aussi indépendants que possible tant d'un point de vue fonctionnel que temporel (calendrier du développement).

3.4.7 Le modèle en spirale

Ce modèle [Boehm, 1988] basé sur un processus évolutif insiste sur l'activité d'analyse des risques à chaque étape : selon l'appréciation du résultat de cette activité on pourra revenir ou non à la phase précédente (voir figure 3.10).

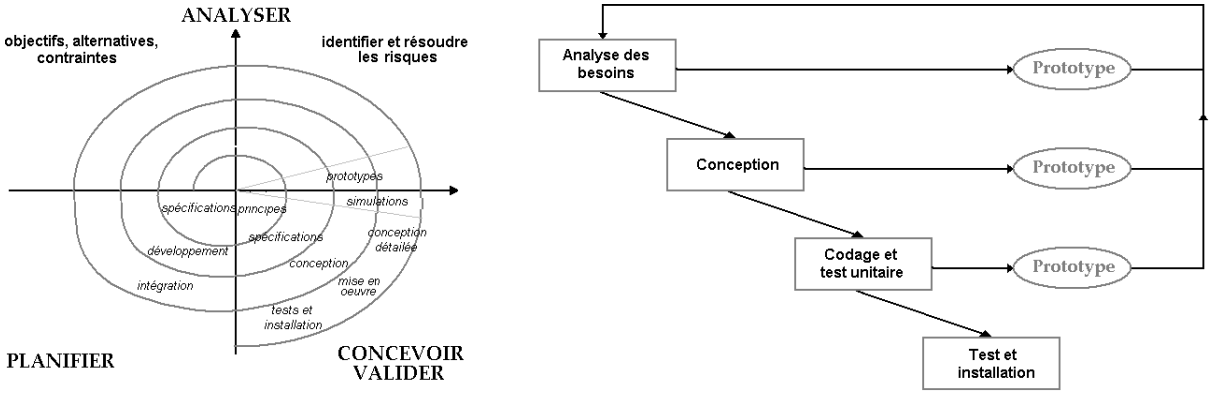


FIG. 3.10 – *Modèle en Spirale*

D'une manière générale on peut dire qu'il combine le prototypage et le modèle séquentiel : avant d'arriver à la version finale on passe par plusieurs versions intermédiaires évolutives. En fait, le coté "prototypage" est encore plus poussé car des prototypes sont développés à la fin de chaque activité. Chaque cycle de la spirale, se déroule en quatre phases :

1. La détermination, à partir des résultats des cycles précédents ou de l'analyse préliminaire des besoins, des objectifs du cycle, des alternatives pour les atteindre, et des contraintes ;
2. L'analyse des risques, l'évaluation des alternatives et, éventuellement le maquettage
3. Le développement et la vérification de la solution retenue. Il est à noter qu'un modèle "classique" (cascade ou en V) peut être utilisé ici,
4. La revue des résultats et la planification du cycle suivant.

L'analyse préliminaire est affinée au cours des premiers cycles. Le modèle utilise des maquettes exploratoires pour guider la phase de conception du cycle suivant. Le dernier cycle se termine par un processus de développement classique.

Ce modèle développé en 1988 a été moins expérimenté car il est beaucoup plus récent que les modèles en cascade ou en V. Sa mise en oeuvre demande de grandes compétences et devrait être limitée aux grands projets innovants à cause de l'importance qu'il accorde à l'analyse des risques. Néanmoins, ce dernier concept peut être appliqué aux autres modèles.

Un des inconvénients majeurs de ce modèle est qu'il nécessite que l'utilisateur soit régulièrement présent. Les risques majeurs du développement du logiciel avec ce cycle de vie sont la défaillance du personnel, le calendrier et le budget irréalistes, le développement de fonctions inappropriées, le développement d'interfaces utilisateurs inappropriées et l'apparition de problèmes de performance en vue des

exigences démesurées par rapport à la technologie : d'une manière générale ce modèle tend à arriver à un produit surdimensionné.

3.4.8 Le modèle en Y

Ce modèle [Larvet, 1994] a pour objectif une maîtrise des coûts et délais grâce à une paralèllisation des tâches d'analyse et de conception. Comme on peut le voir en figure 3.11, la conception consiste en une analyse de l'architecture et aboutit à la définition d'archétypes qui facilitent l'écriture ultérieure de code.

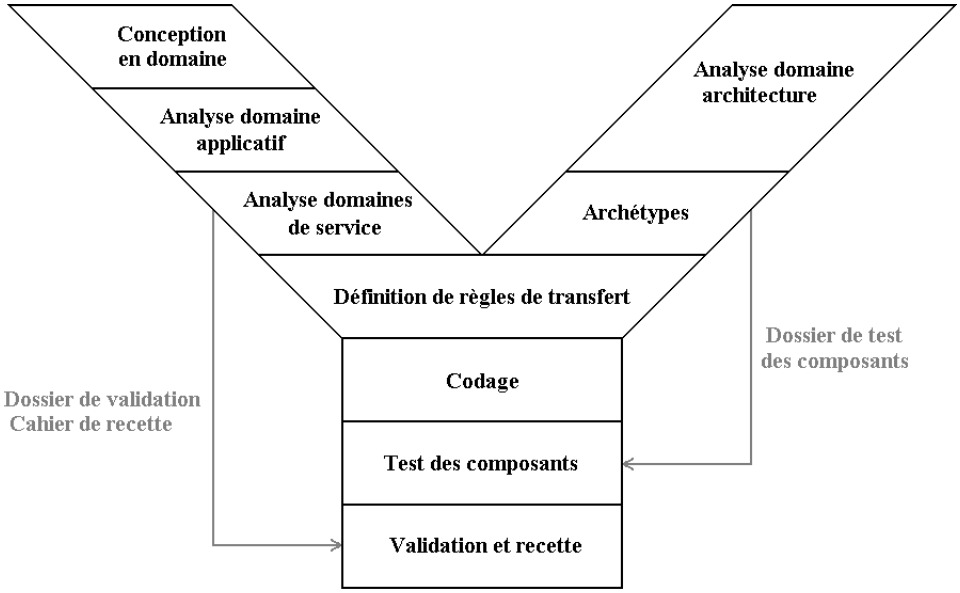


FIG. 3.11 – *Modèle en Y*

Le principal inconvénient de ce modèle réside en la difficulté de mise en oeuvre dans le cadre de grands projets et dans le développement de systèmes complexes. La validation nécessite de faire valider, par le client, le livrable étape par étape. Cette contrainte ajoutée aux contraintes de budgets, de ressources et de planning font que la paralèllisation souhaitée initialement devient très vite un agencement séquentiel.

D'autres modèles apparaissent dans la littérature sous la dénomination de modèle en Y : il s'agit généralement d'une composition du modèle en V pour synchroniser les tests et les spécifications et du modèle incrémental pour sa décomposition en sous-ensembles.

3.4.9 Le modèle opérationnel

Dans ce type de modèle [Agresti, 1986], la production d'une spécification mathématique formelle et sa transformation sont faites selon des méthodes mathématiques. La figure 3.12 illustre ce modèle.

L'abréviation CASE utilisée sur la figure signifie "Computer Aided Software Engineering" c'est-à-dire conception de logiciels assistée par ordinateur.

Ce modèle bien que gérable est quasiment impraticable dans la plupart des cas.

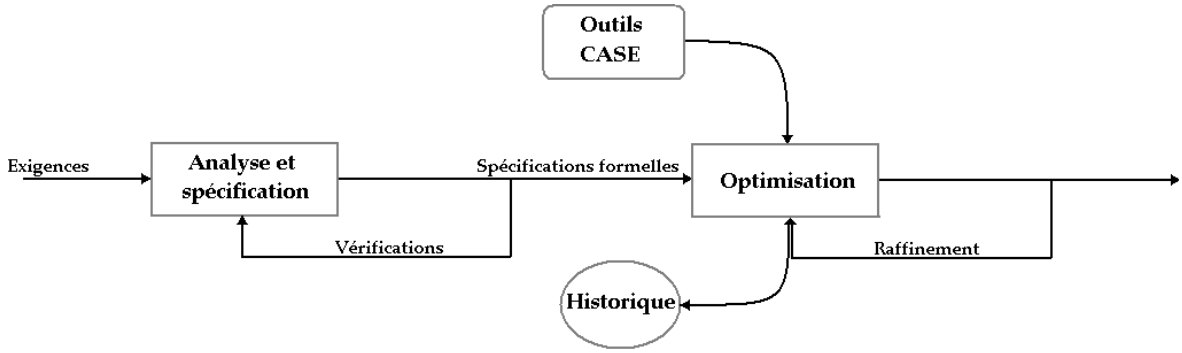


FIG. 3.12 – *Modèle transformationnel*

3.4.10 L'extreme programming

L'*extreme programming* [Beck, 1998, Bénard et al., 2002] est un modèle très récent qui veut exploiter au mieux le travail en équipe pour satisfaire le client qui demande le logiciel. Cette méthode est dite agile c'est-à-dire qu'elle permet de concevoir des logiciels en impliquant au maximum le demandeur afin de permettre une grande réactivité à ses demandes. Son cycle de vie (voir figure 3.13 fait appel à quatre types d'activités: la planification, la conception, le codage et le test.

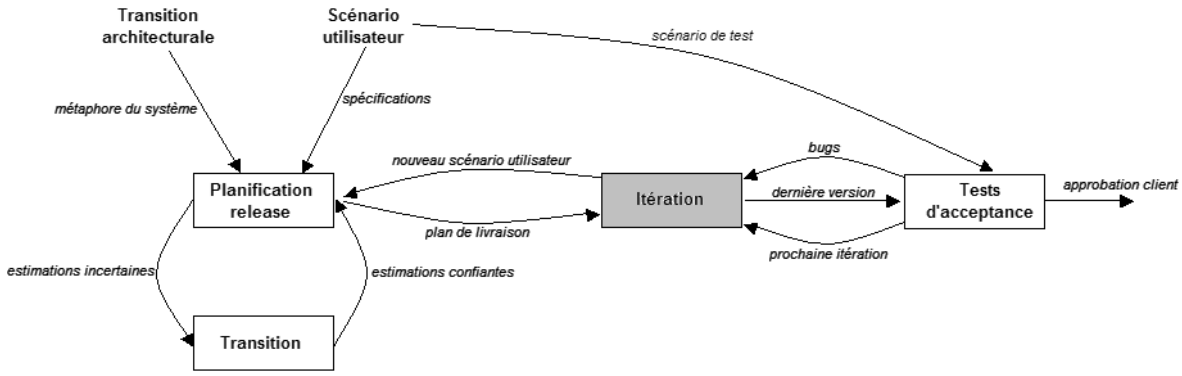


FIG. 3.13 – *L'extreme programming*

L'objectif de la planification est d'établir un planning des livraisons. A partir de scénarios d'utilisation écrits par les utilisateurs qui se substituent au traditionnel cahier des charges, on planifie les dates de livraison (produits intermédiaires et définitifs) et l'affectation des ressources humaines et matérielles.

Durant la phase de conception l'objectif est d'être minimaliste et simpliste afin de concevoir au plus vite et pour le meilleur prix. Pour cela on explore les solutions potentielles, on choisit une métaphore du système et on procède à une conception orientée objet.

La phase de codage nécessite d'avoir le client toujours disponible et de le faire participer aux tests fonctionnels (il n'y a pas de cahier des charges). Le processus de développement consiste en une itération : on crée un test pour un petit aspect du problème à traiter, on crée un bout de code qui passe le test et on itére jusqu'au moment où il n'y a plus rien à tester.

La phase de test est importante car tout bout de code doit avoir ses tests unitaires. Il convient en fait d'écrire un test qui validera l'implémentation avant même d'implémenter une fonctionnalité. Cependant, si un bug est identifié un test sera ajouté afin de s'assurer qu'il ne se reproduise pas par la suite.

L'avantage de ce modèle est de pouvoir faire face à des changements de besoins, par exemple, lorsque le client n'a pas une idée encore précise et définitive du produit qu'il désire. Ses deux inconvénients majeurs sont l'importance des ressources humaines qu'il faut allouer au projet et la présence du client qui doit être toujours disponible.

3.4.11 Le cycle RAD

Le cycle de vie RAD[1] (voir figure 3.14) est employé pour impliquer de manière forte l'utilisateur ce qui permet d'assurer une certaine qualité du produit final. La contre partie est la nécessité de mettre en oeuvre des équipes d'experts afin de proposer dans des temps réduits des solutions répondant aux demandes de l'utilisateur.

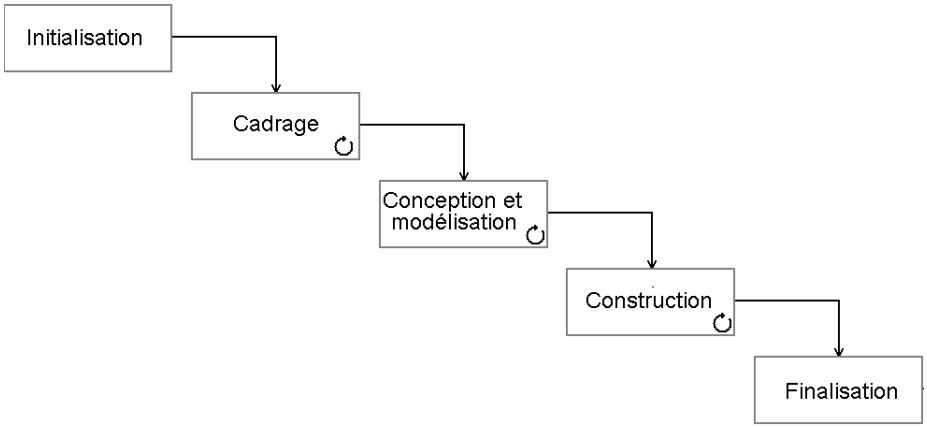


FIG. 3.14 – *Le cycle du RAD*

Le cycle repose sur cinq phases:

1. L'*initialisation* qui permet de définir le périmètre général du projet, de déterminer les thèmes abordés par le projet et d'en déduire les acteurs pertinents,
2. Le *cadrage* qui permet l'analyse et l'expression des exigences,
3. La *conception et modélisation* qui implique l'utilisateur dans l'affinage et la validation des modèles mais aussi l'ergonomie de l'application,
4. La *construction* qui s'effectue module par module et en plusieurs itérations,
5. La *finalisation* qui permet d'officialiser la livraison à partir des recettes partielles obtenues précédement.

Les inconvénients de ce modèle sont liés au haut niveau exigé pour les membres de l'équipe logicielle et à la participation de l'utilisateur à l'organisation du système informatique (son rôle ne se limite pas aux interfaces). L'impression de facilité peut mener l'utilisateur à des exigences difficiles à remplir, ou des contestations sur les coûts.

3.5 Les notations

Nous allons présenter par la suite différentes notations existantes qui permettent de spécifier le logiciel. Elle vont des "plus naturelles" aux plus formelles.

1. Rapid Application Development, http://rad.fr

3.5.1 Les énoncés informels

Cette catégorie de notation se base sur des descriptions faites en langage naturel. Bien que pouvant être standardisées, elles sont bien souvent propres à une entreprise voire un projet ou une équipe de développeurs. L'inconvénient majeur de ce type de techniques est dû à l'ambiguïté du langage. Les risques qui peuvent donc apparaître, en plus de l'ambiguïté, sont les risques d'incohérence, de non complétude, de difficulté d'organisation et de redondance d'information.

3.5.2 Les présentations formatées

Les présentations formatées sont des notations dont l'objectif est de pallier l'ambiguïté du langage naturel. Il existe de nombreux types de présentations formatées. On peut clairement identifier :

- Les *dictionnaires de données* ou *glossaires*, dans lesquels est défini le vocabulaire du domaine d'application , spécifient l'ensemble des données utilisées en analyse et en conception. Il est possible, afin de supprimer encore plus d'ambiguïté, d'utiliser des notations syntaxiques strictes de forme Backus-Naur.
- Les *tables de décision*, qui permettent la définition des systèmes finis, donnent les correspondances entre les valeurs d'entrée et les valeurs de sortie d'un processus.
- Les *tables états-transitions* sont des tables regroupant les états (auxquels on peut associer des actions) et, pour chaque état, les évènements qui provoquent le passage à un autre.
- Le *modèle entité-association* dans lequel les objets du domaine concerné (entités) sont identifiés par des attributs et sont reliés par des liens (associations) dont on peut préciser les limites du nombre d'occurrences (cardinalité).
- L'ensemble des notations utilisées dans UML (diagramme de cas d'utilisation, de séquence, de collaboration, d'états-transitions, d'activités, de classes, d'objets, de composants, de déploiement [Roques and Vallée, 2003]).

3.5.3 Les notations semi-formelles

Ces notations sont relativement simples et permettent ainsi à différentes parties d'un projet de communiquer facilement et efficacement d'autant plus qu'on les associe généralement à des énoncés informels. Les représentations classiques de cette catégorie de notations sont :

- Les *diagrammes états-transitions* dont l'objectif est, comme pour les tables états-transitions, de représenter les différents états du système et les évènements qui déclenchent les changements d'états.
- Les *diagrammes de flots de données* intègrent les flots de données et des processus de transformation. Ils montrent comment chaque processus transforme les données qu'il reçoit en flots de sortie. Il est souvent accompagné d'un *Diagramme De Contexte*, que l'on peut considérer comme le diagramme de flots de données de niveau 0, qui représente les interactions entre le système et l'environnement extérieur.

– Les *réseaux de Petri* qui sont des outils mathématiques permettant de modéliser le comportement des systèmes dynamiques à événements discrets et les *grafcet* eux-mêmes inspirés des réseaux de Petri, qui sont des outils de spécification qui conviennent bien pour les systèmes automatisés séquentiels.

3.5.4 Les notations formelles

Cette famille de notations qui repose sur les mathématiques permet, lors d'une étape de spécification, d'éliminer les ambiguïtés du langage naturel et les mauvaises interprétations qui en découlent. Ces outils permettent de décrire ce que le système doit faire mais pas comment il le fait. Les méthodes et langages de la spécification formelle possèdent tous trois composantes :

1. Une *syntaxe* qui précise la notation utilisée pour procéder à la spécification,
2. Une *sémantique* qui donne une définition non ambiguë de cette syntaxe,
3. Un *ensemble de relations* qui définissent les règles qui donnent les propriétés des objets mathématiques satisfaisant la spécification.

Des représentants des notations formelles sont, par exemple, LOTOS [ISO/IEC, 1988], le langage B [Abrial, 1996], le langage Z [Spivey, 1988] et le π-calcul [Milner, 1999]. Certaines spécifications non formelles peuvent se voir adjoindre des langages formels pour l'expression des contraintes. On peut citer par exemple OCL [Bottoni et al., 2000] dans le cas d'UML.

3.6 La qualité du logiciel

L'ISO (International Organization for Standardization) définissait en 1994 la qualité comme l'ensemble des traits et des caractéristiques d'un produit ou d'un service qui lui confèrent l'aptitude à satisfaire des besoins exprimés ou implicites (ISO 8402). Dans sa version la plus récente, la définition donnée est "aptitude d'un ensemble de caractéristiques intrinsèques s'un produit, d'un système ou d'un processus à satisfaire les exigences des clients et autres parties intéressées" (ISO 9000).

La qualité traite donc de l'aptitude d'un produit ou d'un service à satisfaire les besoins des utilisateurs et selon divers critères tels que la fonctionnalité, le délai et le coût. Les facteurs les plus importants [Call, 1994], qui peuvent d'ailleurs être incompatibles, sont :

– La *conformité* ou *validité* qui estime la capacité d'un logiciel à remplir les fonctions spécifiées lors de la définition du cahier des charges, à satisfaire aux spécifications et à remplir ses missions dans les situations opérationnelles définies,
– La *robustesse* qui détermine l'aptitude à s'adapter à des perturbations, à accomplir sans défaillance l'ensemble des fonctionnalités spécifiées, dans un environnement opérationnel de référence et pour une durée d'utilisation donnée
– Sa *testabilité* ou *vérifiabilité* qui mesure la facilité avec laquelle on peut établir les procédures de test,
– La *sécurité* ou *intégrité* qui estime l'aptitude d'un logiciel à surveiller, recenser, protéger et contrôler les accès aux codes ou aux données afin de se protéger contre des accès non autorisés ou malveillant,
– La *compatibilité*, sous-entendues d'autres applications logicielles,

- L'*extensibilité* qui caractérise la facilité avec laquelle s'applique la maintenance curative et la modification de fonctionnalités existantes,
- L'*adaptabilité* afin de minimiser l'effort nécessaire pour le modifier par suite d'évolution des spécifications
- La *réutilisabilité* qui estime la capacité d'un logiciel à être réutilisé dans de nouvelles applications (que ce soit partiellement ou intégralement),
- Sa *portabilité* vers différents environnements matériels et logiciels,
- L'*interopérabilité* qui traite de sa capacité à s'interconnecter à d'autres systèmes.
- Son *efficacité* qui mesure la bonne utilisation des ressources matérielles ressources strictement nécessaires à l'accomplissement des fonctions prévues (critère important du co-design),
- La *maniabilité* afin de faciliter l'emploi du logiciel par un utilisateur que ce soit en marche normale ou non, de minimiser l'effort nécessaire pour l'apprentissage, la mise en oeuvre des entrées et l'exploitation des résultats.
- La *maintenabilité* afin de minimiser l'effort pour localiser et corriger les fautes.

Ces facteurs sont classés en deux groupes selon le critère de la visibilité : les facteurs internes, qui ne sont visibles que par les développeurs, et les facteurs externes visibles par les utilisateurs du logiciel.

Dans le processus de conception du logiciel, la non qualité se manifeste par des coûts de correction élevé. De plus, ce n'est que lorsque le logiciel sera exploité que ces défauts apparaîtront.

3.7 Conclusion

Ce chapitre présente les généralités sur la production traditionnelle de logiciels pour, entre autres, préciser le vocabulaire qui va être utilisé par la suite. Cette présentation insiste sur les activités élémentaires des méthodes, les cycles de vie et les différents formalismes pour la spécification de logiciels. Il introduit aussi la notion de qualité logicielle.

Nous avons présenté le modèle en cascade pour lequel les activités sont représentées dans des processus séparés et pour lequel un livrable est produit après chaque étape (condition pour passer au processus suivant). Le *modèle en b* est un modèle en cascade pour lequel on insiste sur la maintenance. Le *modèle en V* est une variante du modèle en cascade pour lequel un lien relationnel est fait entre les phases de début de de fin du cycle. Le développement *évolutif* (prototypage) se base sur un prototype initial pour l'évoluer jusqu'à obtenir le système définitif. Le *modèle incrémentiel* permet une implantation élaborée par incrément successif. Le *modèle en spirale* considère le niveau de risque à chaque étape et permet d'itérer sur un ensemble de processus. Le *modèle opérationnel* (transformationnel) pour lequel la production d'une spécification mathématique formelle et sa transformation est faite selon des méthodes mathématiques. L'*extreme programming* est un cycle de vie agile qui exploite des scénarios d'utilisation pour concevoir l'application et qui implique grandement l'utilisateur. Le cycle de vie de la méthode *RAD*, qui nécessite une équipe logicielle experte, a pour objectif une rapide mise sur le marché du produit.

Parmi les différentes techniques pour spécifier un logiciel, on identifie quatre familles. Les énoncés informels sont pratiques car facilement compréhensibles par le demandeur mais laissent libre court à de trop nombreuses interprétations (conséquence de la seule utilisation du langage naturel). Les présentations formatées tentent de pallier l'ambiguïté issue de l'utilisation du langage naturel en la contraignant. Les notations semi-formelles qui sont un bon compromis entre la facilité de compréhension par un tiers

et le traitement des ambiguïtés. Les notations formelles qui sont efficaces pour lever les ambiguïtés des langages naturels mais ne sont utilisables que par des experts.

Les différents critères de la qualité logicielle sont la conformité, la robustesse, la testabilité, la comptabilité, l'extensibilité, l'adaptivité, la réutilisation, la portabilité, l'interopérabilité, l'efficacité, la maniabilité et sa maintenabilité. Ces termes ont été décris. Une démarche qualité passe notamment par une traçabilité du processus d'élaboration du système et donc de sa documentation.

Chapitre 4

Conception de systèmes mixtes logiciel/matériel

[Parlant de la conception conjointe logicielle/matérielle]
Comment prendre deux disciplines très différentes non seulement par les techniques employées, mais aussi culturellement et les assembler en un tout intégré, fonctionnel et respectant un certain nombre de caractéristiques ?

Erik Stoy

Ce chapitre a pour objectif de présenter la conception des systèmes mixtes logiciel/matériel car les systèmes complexes qui nous intéressent sont eux-mêmes constitués à la fois de logiciels et de matériels. Pour cela nous proposons une définition de ce que l'on appelle système embarqué et insistons ensuite sur le co-design en opposition avec les méthodes traditionnelles de conception de systèmes mixtes.

4.1 Les systèmes embarqués

De plus en plus, dans le domaine de l'informatique, on parle de systèmes embarqués, systèmes diffus, systèmes enfouis, systèmes autonomes, systèmes pervasifs etc. Il est important de mener une réflexion sur les caractéristiques de ces systèmes, sur les critères qui les différencient des systèmes logiciels traditionnels.

4.1.1 Définition

Il est difficile de trouver une définition précise de "systèmes embarqués" [Vahid and Givargis, 2002]. On peut cependant s'appuyer sur des définitions des systèmes embarqués qui restent tout de même relativement vagues.

Dans [Vahid and Givargis, 2002], un système embarqué est défini comme un système informatique qui n'est autre qu'un ordinateur, un ordinateur portable ou une station de travail. Suite à cet énoncé, les

auteurs énumèrent plusieurs applications possibles.

Dans [Kadionik, 2004], la définition proposée est "un système électronique et informatique autonome ne possédant pas d'entrées/sorties standards comme un clavier ou un écran d'ordinateur". Il précise aussi que "le système matériel et l'application sont intimement liés et noyés dans le matériel et ne sont pas aussi facilement discernables que dans un environnement de travail classique de type PC".

Dans [Laurgeau et al., 2002], l'appellation embarquée est présentée comme un qualificatif à "tout système informatique qui est appelé à fonctionner sur une plate-forme mobile (véhicule terrestre, voiture, camion, engin agricole, militaire etc.)". Il est aussi précisé que leurs spécificités résident dans des contraintes d'autonomie énergétique, la résistance aux vibrations et le cas échéant à d'autres problèmes liés à la masse ou au volume transporté, à la dissipation thermique, aux contraintes électromagnétiques.

Dans [Koudil, 2004], il est précisé qu'il s'agit d'un système contenant du matériel et du logiciel complètement intégré dans l'environnement qu'il contrôle. Il est généralement autonome, exécute une tâche précise dédiée à une application spécifique, remplace souvent des composants électromécaniques, n'a généralement pas de périphérique standard et possède une interface IHM qui peut être aussi simple qu'une diode électro-luminescente qui clignote ou aussi complexe qu'un système de vision de nuit en Temps Réel.

D'une manière générale toutes ces définitions sont connotées: on devine dans ces définitions les applications auxquelles s'intéressent les auteurs. Les auteurs de la première définition, très simpliste, insiste sur le fait que leur définition est imparfaite et consacre une bonne partie de leur premier chapitre aux applications. La seconde définition proposée laisse penser que l'auteur, spécialiste de *Embedded Linux*, s'intéresse aux applications similaires aux applications traditionnelles mais embarquées sur des systèmes de type *PC industriel*[1]. La troisième définition lie le caractère embarqué à la mobilité des équipements mis en jeu dans les applications mais aussi aux contraintes imposées par l'environnement applicatif. La quatrième définition insiste sur le coté matériel/logiciel du système et laisse pressentir des applications de contrôle en milieu industriel.

Une définition plus ambitieuse a été donnée par le travail de synthèse effectué par le groupe de travail dans le cadre d'un RNTL (groupe de travail B1 : systèmes embarqués) [Joloboff et al., 2000]. Il est appelé système embarqué " un appareillage remplissant une mission spécifique en utilisant un ou plusieurs microprocesseurs. Ces microprocesseurs embarqués, et souvent invisibles de l'utilisateur, effectuent des opérations faisant partie intégrante de la mission de l'appareillage en question. L'appareillage n'est pas vu par l'utilisateur comme un ordinateur exécutant des applications informatiques comme pour un ordinateur ou un serveur.". Les auteurs de cette définition précisent qu'elle est un peu vague car ils pensent qu'il est nécessaire de faire la distinction entre systèmes personnels et système collectifs.

La vision que nous avons des systèmes embarqués est en fait assez différente. Pour l'illustrer, nous commencerons par critiquer et commenter cette dernière définition. Dans un premier temps, il est mis en avant que le coeur des systèmes embarqués sont les microprocesseurs. Cette caractéristique nous parait très limitative. En effet, de plus en plus, les systèmes embarqués reposent en partie sur des composants électriquement programmables (Electrical Programmable Logic Device). Dans ces composants électroniques, contrairement aux microprocesseurs qui exécutent un programme stocké en mémoire, c'est, par

1. Les cartes "PC industriel" sont des cartes mères autonomes répondant à des critères ergonomiques précis (dimensions réduites, faible consommation d'énergie etc.)

exemple, la synthèse combinatoire qui est codée en dur dans le composant (on "grille des fusibles") : il n'y a pas de mémoire!!!
Dans un second temps, la définition précise que ces microprocesseurs, participent à une mission de l'appareillage en question (pour les systèmes multi-processeurs) mais qu'il n'est pas vu par l'utilisateur comme étant un ordinateur... Mais qu'est ce qu'un ordinateur? Un ordinateur est un système souvent multi-processeurs : on y trouve le microprocesseur principal qui exécute les programmes utilisateurs, des processeurs dédiés (processeurs graphiques et autres contrôleurs). On y exécute des programmes pour fournir des services, remplir certaines missions. L'ordinateur est-il pour autant un système embarqué?
Enfin, l'aparté des auteurs précise que le coté collectif y est essentiel : un système embarqué dans une automobile qui a pour unique tâche le contrôle du freinage n'est cependant pas, par essence même, un système embarqué? A contrario, un ordinateur collaborant avec d'autres sur un réseau devient-il un système embarqué?

> On le voit bien, il sera impossible de donner une définition qui sera acceptée par tous. Celle qui conviendrait le mieux, à notre sens, serait d'assimiler la caractéristique "embarquée" comme étant un ensemble de contraintes fortes (d'autonomie, financières, techniques et énergétiques). En effet, en plus des contraintes propres à tous les systèmes informatiques, on ajouterait par exemple selon les cas:
>
> – des contraintes de temps d'exécution de type temps réel et bien que les processeurs soient de puissance toute relative face à ceux qui équipent les ordinateurs de bureau (ils sont généralement cadencés à des fréquences d'environ 8MHz à 40MHz), ainsi que de capacités mémoires limitées (les cartes sont généralement équipées de 32ko à 128ko de mémoires (pour le code et données).
> – Les systèmes embarqués sont souvent autonomes d'un point de vue énergétique et doivent donc être le plus économe possible.
> – Enfin la sûreté de fonctionnement est capitale : les systèmes embarqués doivent être tolérants aux pannes et ne jamais être dans un état bloqué. Pour des raisons de sécurité c'est au matériel qu'il convient généralement d'assumer ce rôle[2].

4.1.2 Un domaine en essor

L'engouement pour les systèmes embarqués n'est pas une mode passagère : le marché est là pour le démontrer.

Le rapport réalisé par le groupe de travail "Systèmes Embarqués" à la demande du Secrétariat d'Etat à l'Industrie [Joloboff et al., 2000] considère le marché de l'embarqué comme "en pleine explosion" et la suppose comparable à celle du web avec le développement de nouvelles applications de plus en plus complexes : téléphone portable et bientôt multimédia, contrôle et gestion sophistiquées des systèmes de commande et de propulsion notamment en automobile, aide et assistance à la conduite des transports terrestres et aériens, gestion des systèmes de sécurité, robotique, domotique, systèmes automatisés autour des capteurs intelligents, pacemakers, prothèses automatisées, agendas électroniques, application des cartes à puce... L'informatique embarquée touche en effet de nombreux domaines : en 1999 seul 5% des processeurs vendus sont à destination du marché des ordinateurs contre 95% pour le marché de l'embarqué [Koudil, 2004]. L'impact dans certains secteurs est tel que le magazine *Automobile* de mars 2002, qui considère que 80% des nouveautés du secteur sont liées à l'électronique, affirme que "les

constructeurs changent de métier".

Les systèmes embarqués ne sont pourtant pas un domaine récent de l'informatique ou de l'électronique. Cependant, ils connaissent bel et bien un accroissement important. Les besoins qui sont apparus et qui sont à l'origine de cette explosion sont les besoins en mobilité des hommes qui ont entraîné une mobilité des équipements. D'autres critères peuvent être les besoins en sécurité et en confort qui ont rendu tous les capteurs et actionneurs intelligents et communicants et la qualité qui a rendu l'instrumentation omniprésente dans tous les moyens de production. Il est quasiment impossible de faire une liste exhaustive de ces besoins d'autant plus que certains d'entre eux naissent de ces mêmes besoins : on peut citer par exemple la sécurité dans les transmissions qui est un domaine en plein essor grâce aux besoins en mobilité.

4.1.3 Systèmes embarqués : logiciel et matériel.

L'avancée des technologies en informatique et en électronique numérique a entraîné la mutation d'un grand nombre de systèmes autrefois matériels vers le domaine plus logiciel.
Nous avons vu en §1.3 un aperçu des catégories de systèmes physiques. Nous allons tenter de proposer une classification de ces familles en fonction de leur orientation plutôt logicielle ou matérielle dont nous proposons une synthèse en figure 4.1.

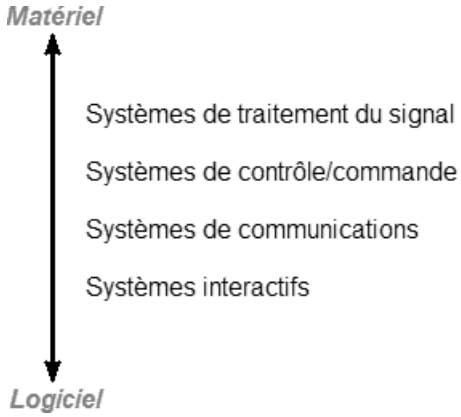


FIG. 4.1 – *Familles de systèmes embarqués*

Les systèmes de traitement d'informations contiennent deux familles distinctes :

- Les systèmes de traitement de signal qui sont à l'origine des systèmes de traitements analogiques (traitement du la voix, traitement de signaux vidéo PAL/SECAM/NTSC etc.) et donc plutôt matériel. Cependant, l'avancée des technologies en semi-conducteur a entraîné un glissement vers le numérique et donc le logiciel. Ces composants spécialisés sont par exemple les DSP (Digital Signal Processing).
- Les systèmes de traitement de données logicielles que représentent, par exemple, les bases de données. De plus en plus les systèmes matériels gèrent des collections importantes de données pour assurer par exemple la traçabilité. Dans ces systèmes, ce sont des logiciels qui assurent cette gestion.

Les systèmes de contrôle/commande qui permettent l'évolution de procédés physiques et décident d'action. A l'origine exclusivement physique (régulateur mécanique, régulateur électrique etc.), ce domaine est devenu lui aussi de plus en plus logiciel.

Les systèmes de communications qui gèrent les flots de messages. Autrefois essentiellement matériels, la puissance et la flexibilité qu'offrent les logiciels a entraîné un glissement vers le logiciel tout comme pour les systèmes de traitement d'information.

Les systèmes interactifs qui assurent le dialogue homme-machine. Dans de tels systèmes le logiciel est prédominant. Il permet de consulter une base de données, de s'adapter aux utilisateurs etc.

4.1.4 Les machines virtuelles

Il est important d'aborder les machines virtuelles lorsqu'on s'intéresse aux systèmes embarqués. Il a longtemps été question qu'elles révolutionnent le monde du système embarqué. En effet, l'utilisation de machines virtuelles permet de pallier le problème de la portabilité : le langage et le programme que l'on crée ne sont pas dépendants de la plate-forme cible. Qu'en est-il vraiment de leur utilisation en informatique embarquée ? Il est assez difficile de répondre à cette question avec précision mais on peut déjà dire qu'elle n'a pas eu le succès escompté. Bien que leur intérêt soit incontestable, leur utilisation en informatique embarquée reste essentiellement limitée aux PDA et autres téléphones mobiles même si certains parient sur le fait que les ressources disponibles sur les systèmes embarqués s'accroîtront toujours [Baker and Ong, 2002]. Il y a peu de systèmes de contrôle ou de système de supervision qui utilisent les langages basés sur des machines virtuelles. Dans l'article *JAVA et l'informatique industrielle* [Chiozzotto and Lenail, 1998] écrit par des cadres Sun Microsystems, Java[3] n'est mis en avant essentiellement pour la dimension applicative et la dynamique d'évolution qu'il apporte à Internet et aux réseaux d'une manière générale. En effet, les machines virtuelles embarquées se limitent généralement aux terminaux internet, aux organiseurs et autres téléphones (KVM [Sun Microsystems, 2000], Jeode[4], ChaiVM[5], PERC[6]).

Les concepteurs et développeurs de systèmes embarqués logiciel/matériel viennent à la fois du domaine de l'électronique et de l'informatique. Ils aiment les systèmes déterministes, compacts et performants: les machines virtuelles satisfaire à ces critères [Clohessy, 2001, Quinnell, 2002] pour entrer réellement dans le monde de l'embarqué.

Des travaux logiciels [Ericsson, 2003, AFNOR, 2003, Chen and Talwar, 2003, Talpin et al., 2004] mais aussi matériels [Newman, 2004, Xu and Xin, 2005, Yiyu et al., 2005]. ont lieu sur ce sujet mais, pour la plupart, ils sont toujours à destination des équipements personnels type PDA.

3. Dans la suite, nous prendrons comme exemple le langage java qui est la référence des langages portables utilisant une machine virtuelle.

4. http://www.insignia.com

5. http://www.hp.com/emso/products/chaivm.html

6. http://www.newmonics.com/WebRoot/perc.info.html

4.2 Le co-design

Nous appellerons *système mixte* un système constitué d'une partie matérielle et d'une partie logicielle.

Les systèmes mixtes sont traités depuis longtemps et souvent d'une unique manière : dès les différents besoins identifiés, on sépare ce qui deviendra la partie logicielle et la partie matérielle du système. Cette séparation était nécessaire car les compétences à mettre en oeuvre étaient différentes (informatique, électronique numérique et analogique). Ainsi une équipe "logiciel" s'occupera de la première des parties tandis qu'une équipe "matériel" concevra l'équipement électronique sur lequel, en fin du processus de production, le logiciel sera intégré (voir figure 4.2).

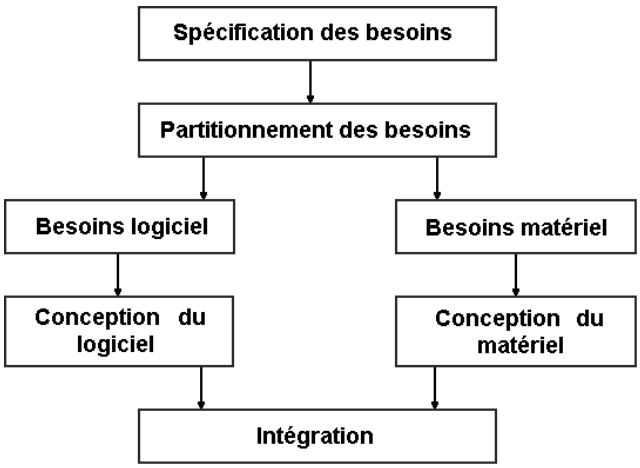


FIG. 4.2 – *Approche traditionnelle de développement d'un système mixte*

Cette méthode présente de nombreux problèmes dont voici les plus importants [Koudil, 2002]:

- Les spécialistes du logiciel connaissent généralement mal les fonctionnalités proposées par le matériel, l'exploite donc mal et recrée même parfois ce qui existe déjà,
- Les spécialistes du matériel apprécient mal les besoins des gens du logiciel ce qui conduit souvent à un mauvais dimensionnement de cette partie du système,
- De la phase d'intégration des différentes parties du système peuvent émerger des problèmes qui nécessitent parfois d'importantes modifications d'une ou l'autre des parties,
- Un ajustement ou un changement des spécifications tardif peut remettre en question une des parties du système.

Dans [Skazinski, 2003], il est rapporté que 71.5% des projets d'informatique embarquée n'atteignent pas 30% des performances attendues durant la phase de pré-conception. Les méthodes actuelles sont donc en échec! Une enquête parue dans [Venture Development Corp., 2003] nous informe que, d'après un sondage réalisé auprès de 400 entreprises travaillant dans le domaine de l'embarqué, les retards dans les projets proviennent généralement de changements de spécifications ou de leurs inadéquations.

4.2.1 Définition

La démarche vue précédemment présente de nombreuses limites que le co-design tente de repousser. Les méthodes co-design proposent en fait un processus de conception unifiée des deux différentes parties

que l'on appelle conception conjointe matérielle/logicielle[7].

> Nous appelerons *co-design* une méthode unique de conception, couvrant l'intégralité du cycle de vie du système mixte, en unifiant la conception des parties logicielles et matérielles.

Il a été constaté lors de différent travaux [Kalavade and Lee, 1993, Wolf, 1994] que pour que la conception d'un système mixte soit non seulement fonctionnelle mais soit aussi optimale d'un point de vue d'encombrement du code ou de la mémoire nécessaire, en terme de contraintes temps réel, de surface et de coût, il faut que le matériel et le logiciel soient conçus ensemble : le point clé des méthodes co-design est donc de retarder au plus tard cette séparation entre le matériel et le logiciel [Kalavade, 1995]. Elles ont donc pour objectif de conduire au meilleur compromis matériel/logiciel à travers le processus de conception [Kumar et al., 1993] car, si pour certaines tâches le choix entre l'implémentation matérielle ou logicielle d'un algorithme est facile (comme pour le traitement de paquets de grandes tailles à grande vitesse), ce n'est pas toujours le cas [Woo et al., 1994].

Les méthodes co-design intègrent donc dans un même environnement la conception du matériel et du logiciel.

La grande question du co-design [Stoy, 1995] est en fait *"Comment prendre deux disciplines très différentes non seulement par les techniques employées, mais aussi culturellement et les assembler en un tout intégré, fonctionnel et respectant un certain nombre de caractéristiques ?"*

4.2.2 Motivations du co-design

Différents avantages des méthodes co-design par rapport aux méthodes traditionnelles ont été identifiés par [Stoy, 1995] repris par [Koudil, 2002] :

- Une flexibilité de conception qui rend le processus permissif au changement de spécification pendant le processus de développement,
- Un délai de mise sur le marché réduit grâce au fait qu'une méthode de conception unique permette d'accélérer le processus de conception,
- Une exploration efficace de l'espace de conception[8],
- Une phase d'intégration et de test simplifiées car la conception conjointe prévient des différents problèmes intervenant lors de ces différentes phases dans les méthodes traditionnelles,
- Une conception à moindre coût car la phase d'exploration a permis de s'intéresser à de nombreux compromis logiciel/matériel,
- Un prototypage rapide.

4.2.3 Les étapes du co-design

Les étapes mises en jeu dans le co-design sont différentes des étapes que l'on rencontre dans les méthodes de conception traditionnelles. Le graphique de synthèse établi par [Koudil, 2002] montre l'enchaînement de celles-ci (voir figure 4.3).

Il est important de mettre en évidence la comparaison du processus de co-design (figure 4.3)) et du processus de conception traditionnel (figure 4.2). Dans le cycle de vie traditionnel on a, dès le cahier des charges effectué, procédé au partitionnement en établissant deux cahiers des charges distincts. C'est

7. Les anglophones utilisent indifféremment les notations *Hw/Sw design* ou *Hw/Sw co-design* ou *co-design*

8. On appelle espace de conception l'ensemble des solutions logicielles, matérielles ainsi que tous les compromis pour accomplir la tâche incombant au système

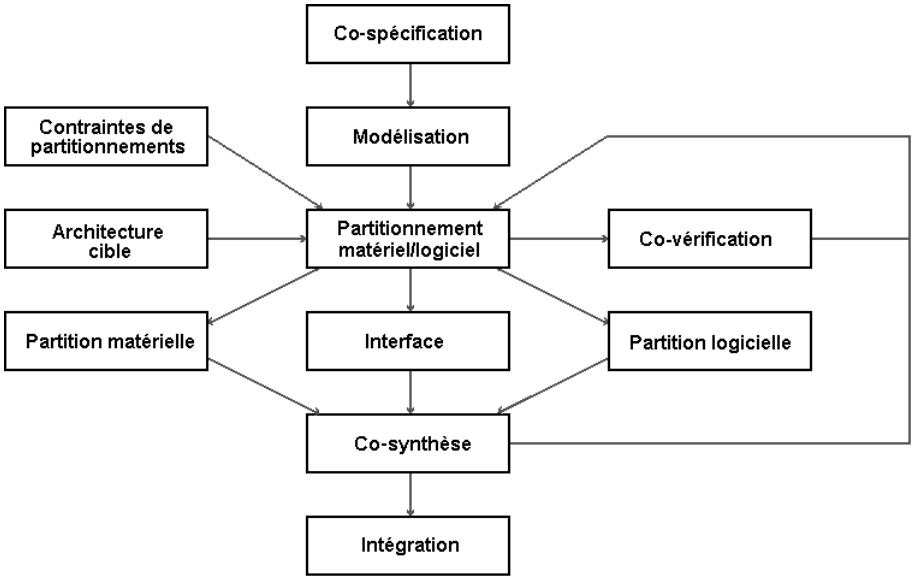


FIG. 4.3 – *Approche co-design "type" pour le développement d'un système mixte*

à ce niveau qu'a été faite une grande partie des choix matériels et logiciels. L'essentiel du travail de conception et de développement peut alors être fait (conception logicielle et matérielle en parallèle).

Avec une méthode de co-design, à partir du cahier des charges on va décrire le fonctionnement du système sans discriminer le logiciel et le matériel. Le partitionnement peut se faire de manière automatique pour optimiser certains critères: plusieurs configurations logicielles/matérielles peuvent être testées, ce qui ne serait pas si simple avec un cycle traditionnel (cela impliquerait le développement de plusieurs solutions en parallèle). De plus, un changement de spécification ne remettra pas autant de travail en cause que dans le cas d'un développement traditionnel pour lesquels, au fur et à mesure de la conception, des contraintes sont définitivement adoptées. Il en est de même pour l'ajout de fonctionnalités.

Voici une description succincte de ces différentes étapes :

Etape de co-spécification Tout comme dans les méthodes traditionnelles de conception, la première étape de la méthode consiste à s'intéresser aux besoins des utilisateurs et donc à leur spécification. Dans le contexte des systèmes mixtes, il faut nous attacher à utiliser un langage assez puissant pour spécifier le logiciel, le matériel ainsi que toutes les contraintes[9] imposées aux systèmes.

Trois différents points de vue se confrontent alors :

- Celui qui consiste à utiliser un formalisme existant (et un seul) pour spécifier l'intégralité du système mixte. Dans ce cas, ce sont les langages de description matérielle qui sont le plus souvent utilisés entre autres parce que de nombreux outils de synthèse logique et de haut niveau existent [Olukotun et al., 1994].
- Celui qui consiste à créer de nouveaux formalismes dédiés à la co-spécification comme Promela [Tripakis and Courcoubetis, 1996], Ruby [Jones and Sheeran, 1990] etc.
- Celui qui consiste à faire collaborer différents formalismes issus du logiciel classique *(Fortran,C)* éventuellement spécialisé *(Hardware C [Ku and De Micheli, 1990], Cx [Ernst et al., 1993], ...)*,

9. Ces contraintes peuvent être quelque fois très difficiles à formaliser (ex: contraintes de sécurité etc.)

des langages de description des systèmes distribués *(SDL (Specification and Description Langage [Turner, 1992]), Estelle [Turner, 1992], Lotos [ISO/IEC, 1988], CCS (Calculus of Communicating Systems [Milner, 1985]), CSP (Communicating Sequential Process [Hoare, 2004]) ...*, des langages synchrones *(Esterel [Berry and Gonthier, 1992], Lustre [Caspi et al., 1987], Lucid [Ashcroft and Wadge, 1977])* et parallèles *(OCCAM [Pountain, 1987])* voire, depuis peu, des approches orientées objet (C++, Java).

Le VHDL [10] [Aumiaux, 2000] et Verilog [E. Sternheim, 1993] sont les langages de description de matériel les plus populaires et sont donc devenus, de ce fait, les outils de spécification de système mixtes les plus répandus.

Etape de modélisation De très nombreux formalismes sont utilisés pour la modélisation des systèmes mixtes *(FSM (Finite State Machine), DFD (Data Flow Diagrams), RDP (Réseaux de Petri) ...)*. Le formalisme qui apparaît comme étant le plus utilisé pour la modélisation des systèmes matériels sont les machines à états finis. Ces dernières étant aussi très utilisés en logiciel, la modélisation des systèmes mixtes repose souvent sur elles. Le co-design utilise donc très majoritairement les machines à états finis.

Etape de partitionnement logiciel/matériel Le rôle de cette étape est la création d'une architecture composée d'une partie matérielle et d'une partie logicielle à partir des spécifications de la partie système relevant de l'activité du co-design. Les paramètres de cette transformation sont par exemple:

- Les *contraintes statiques* qui sont le coût financier, la consommation énergétique, la surface de silicium maximale, la taille du code et de la mémoire,
- Les *contraintes dynamiques* qui sont essentiellement les contraintes temporelles,
- La *sûreté de fonctionnement* qui peut par exemple entraîner une redondance de composants matériels et/ou logiciel voire l'utilisation de mécanisme proche des techniques de réplication utilisées en logiciel,
- La *testabilité* qui va ajouter des composants matériel supplémentaire ou l'ajout d'instructions,
- La *réutilisation*.

Cette étape peut être manuelle, lorsque le partitionnement est réalisé par l'utilisateur, interactif lorsqu'il est assisté par des outils, automatique lorsque qu'il est traité intégralement par des outils. Généralement ces derniers utilisent des approches heuristiques. Il est à noter que s'agissant d'un problème NP complet, le nombre de critères doit être limité lors de recherches automatiques.

Etape de co-synthèse Cette étape permet la co-synthèse de la communication entre les différentes parties indépendantes du système partitionné. Les sous-systèmes mis en jeu ont des besoins en communication : il est nécessaire de permettre celle-ci. Les principales approches de synthèse d'interface utilisent des protocoles de communication, des systèmes d'exploitation, des primitives ou des bibliothèques de communication.

Etape de co-vérification La co-vérification utilise la co-simulation qui permet une simulation simultanée des parties logicielles et des parties matérielles sans en oublier leurs interactions. Cette co-simulation permet d'analyser les propriétés dynamiques du système. Elle permet de co-vérifier le système c'est-à-dire de garantir :

- Les performances et fonctionnalités du système à concevoir sont conformes aux co-spécifications.

10. Very High Description Langage

– Que des déviations du fonctionnement (contraintes temporelles, contraintes ergonomiques etc.) n'aient pas été introduites par les différentes étapes du processus de co-design.

Par ailleurs, il est possible de procéder à de la co-validation. Elle se fait à l'aide d'outils formels. Elle seule permet de vérifier que le système réalisé respecte toutes les contraintes fixées : elle est cependant très difficile à mettre en oeuvre.

4.2.4 Revue des environnements co-design

Il existe de nombreux environnements de co-design [Ismail, 1996, Berrebi, 1997, Valderrama, 1998, Zurawski, 2005] dont un aperçu non exhaustif est présenté dans le tableau 4.1. Pour chacune des méthodes, le tableau présente l'outil de spécification, le type d'application visé ainsi que les cibles potentielles pour la réalisation.

TAB. 4.1 – *Apercu des différents environnements de co-design*

Outils	Spécification	Type d'application	Architecture cible
$Castel$ [Theißinger et al., 1994]	C	Traitement du signal	ASIP
$Codes$ [Buchenrieder et al., 1993]	SDL, StateCharts	Système de communication	Multiprocesseurs, FPGA ,ASIC
$Cosmos$ [Ismail, 1996]	SDL	Système de contrôle, communication etc.	Multi-processeurs,ASIC, FPGA
$Cosyma$ [Ernst et al., 1993]	Cx	Systèmes complexes	CPU, ASIC
$CoWare$ [De Man et al., 1995]	POPE	Traitement du signal	Multi-processeurs, ASIC
$Lycos$ [Grode et al., 1998]	C		CPU + ASIC
$MCSE$ [Calvez et al., 1997]	Formalisme personnel	Système de contrôle et de communication	Monoprocesseur, DSP, ASIP
$Ptolemy$ [Eker et al., 2003]	Blocs interconnecté multi-langage	Traitement du signal et systèmes de communications	Mono-processeur
$Rapid$ [Rethman and Wilsey, 1993]	VHDL	Système de communication et de contrôle	Au moins un processeur (pour le logiciel)
$RASSP$ [Sedmak and Evans, 1995]	VHDL	Traitement du signal	CPU + DSP
SAW [Thomas et al., 1988]	CSP	Optimisation de fonctions logicielles	CPU + (FPGA,ASIC)
$SpecSyn$ [Gajski et al., 1998]	SpecCharts	Système de contrôle et de communication	Mono-processeurs et ASIC
$SynDex$ [Sorel, 1994, Niang et al., 2004]	SIGNAL	Traitement du signal	Multi-processeurs
$Tosca$ [Allara et al., 2000]	SpeedChart	Systèmes de communication	Mono-processeur avec co-processeur
$Vulcan$ [Gupta and De Micheli, 1993]	HardwareC	Système temps réel	CPU, ASIC

Formalismes de spécification. Un des intérêts de cette revue est de permettre l'identification de formalismes de spécification. On remarque que les outils démarrent avec des modèles de spécification de bas niveau par rapport au niveau système afin de réduire la distance entre le modèle de spécification et le prototype.

Certaines approches ont étendu des langages pour supporter des concepts matériels et la communication comme Cosyma (Cx est une extension du langage C qui intègre des contraintes temporelles et le

parallélisme) et Vulcan (HardwareC est une extension au langage C pour intégrer le parallélisme et des aspects pour la description matérielle). Lycos et Castle utilisent le langage C.

Certains travaux sur le codesign utilisent des langages de spécification issus des langages de spécification de systèmes distribués tels que SDL (traduction de modèle SDL en VHDL [Glunz et al., 1993]), LOTOS (processus LOTOS traduit en automate à états finis VHDL [Carreras et al., 1995]) ou ESTELLE (traduction de ESTELLE en VHDL [Wytrebwicz and Budkowski, 1995]).

StateCharts [Harel, 1987] est un langage synchrone ayant un formalisme visuel et permettant de décrire des machines à états finis ainsi que leur contrôle dans le temps.

Parmi les formalismes propriétaire, on compte SpecCharts [Narayan et al., 1992] qui est un langage de spécification au niveau système semblable à StateCharts avec des instructions issues du VHDL (description comportementale). Speedchart [Belhadj, 1994] est un langage de description graphique et textuel qui utilise le formalisme StateCharts. POPE [De Man et al., 1995] est un formalisme qui sépare strictement le comportement fonctionnel et la communication. Le modèle repose sur des processus qui décrit le comportement des ports qui permet aux composants de communiquer, les protocoles qui définissent la sémantique de la communication et des canaux qui permettent de connecter des ports. Ptolemy est un des premiers framework à utiliser des blocs interconnectés [Buck et al., 1994] pour décrire le fonctionnement d'une application. Ces blocs permettaient de cacher des objets écrits en C++, ce qui était à l'époque très novateur.

Type d'applications. Comme on peut le voir dans le tableau 4.1, les différents outils de co-design ne visent généralement qu'un champ d'application (système de contrôle, système de traitement, système de communication etc.). Deux paramètres en sont à l'origine. Tout d'abord, ces outils sont généralement associés à des méthodes, et les méthodes sont créées par des équipes qui travaillent sur des applications spécifiques. Ce sont donc les applications qui ont dirigé les méthodes. Le deuxième point est que les méthodes demandent souvent beaucoup de bibliothèques (de composants, de fonctions). Ces bibliothèques, lourdes à développer, sont créées pour ces mêmes types d'application initialement visées.

Architectures cibles. Un processeur (noté CPU) exécute, à chaque top d'horloge, une action correspondant à une instruction ou une partie d'instruction. Ces instructions sont des opérations élémentaires que le processeur peut accomplir. Ces instructions, stockées dans une mémoire réservée au code (généralement de type ROM), nécessitent de pouvoir manipuler des données stockées temporairement dans une mémoire (généralement de type RAM). Ces processeurs peuvent avoir plusieurs architectures dont les plus connues sont les architectures CISC (Complex Instruction Set Computer) dans lesquels des instructions complexes (difficilement faisables avec les instructions de bases) sont câblées en dur dans le processeur et RISC (Reduced Instruction Set Computer) optimisé pour des cycles d'horloge courts mais qui ne tolèrent qu'un jeu d'instruction réduit.

Les ASIP (Application Specific Instruction Set Processor) sont des processeurs programmables spécifiques qui sont développés pour des applications données (optimisation de leurs architectures internes, de leurs jeux d'instructions).

Les ASIC (Application Specific Integrated Circuit) sont des circuits spécifiques qui sont construits pour des applications spécifiques. Ces composants sont généralement utilisés pour accélérer certaines des tâches que doit faire un système à processeur (ils peuvent cependant être utilisé d'une manière autonome).

Les DSP (Digital Signal Processor) sont des processeurs de signal numérique et sont optimisés pour les calculs type traitement de signal. En terme de structure, un DSP est un processeur, ou calculateur, dont l'architecture est optimisée pour effectuer des calculs complexes en un coup d'horloge, mais aussi pour accéder très facilement à un grand nombre d'entrées-sorties.

Les FPGA (Field Programmable Gate Array) sont des circuits intégrés qui peuvent être reprogrammés. Ils sont plus lents que les ASIC et consomment plus d'énergie. Leur temps de développement est cependant plus court et leurs coût inférieur pour de petites séries. Il est souvent possible de transformer un FPGA en une version ASIC plus rapide et consommant moins.

Les PAL (Programmable Array Logic), les PLD (Programmable Logic Device), les EPLD (Erasable Programmable Logic Device) et les EEPLD (Electrically Erasable Programmable Logic Device) sont des réseaux logiques programmables. Leur programmation consiste à établir des connexions en imposant un courant supérieur aux courants de fonctionnement normaux (claquage de fusibles ou de jonctions). Les circuits logiques programmables incluent un grand nombre de solutions, toutes basées sur des variantes de l'architecture des portes ET, OU et l'utilisation de bascules pour la mémorisation.

4.3 Conclusion

Les objectifs de ce chapitre étaient de définir le vocabulaire que nous utiliserons dans la méthode, de définir que ce que l'on entend par système embarqué et de présenter le co-design.

Lorsqu'on s'intéresse à la production de systèmes mixtes logiciel/matériel, il semble important d'unifier en un même cycle de vie le développement des deux parties. L'objectif est d'éviter les écueils habituels de leur conception et de profiter des avantages d'une telle harmonisation des cycles de vie des deux parties (vu en 4.2.2).

L'inconvénient des méthodes traditionnelles de co-design est leur trop grande spécialisation : une méthode est faite pour un type de problème (codec, module de transmission, module de traitement de signal etc.) et qu'elles sont axés sur l'implémentation. Pour notre type de problème il faut que la démarche co-design commence dès l'analyse: le co-design ne doit pas se limiter à la construction de l'agent.

Il y a de nombreux formalismes : le choix d'un formalisme de spécification dépend généralement du critère que l'on souhaite privilégier. Les formalismes les plus faciles à prendre en main reposent sur des notations graphiques. Cependant, la diversité des problèmes que l'on doit pouvoir envisager tend à penser qu'il ne faut pas autoriser la collaboration de plusieurs formalismes.

Une méthode peut ne pas imposer un formalisme particulier mais se comporter comme un guide. Cependant, au niveau de l'outil, le choix d'un formalisme est important.

L'idée de blocs interconnectés (Ptolemy) est très séduisante pour décrire le fonctionnement d'une entité d'un système. L'inconvénient est que seule la richesse des bibliothèques disponibles rend une telle approche réellement consistante. D'autre part, les machines à états finis sont un outil efficace pour modéliser un fonctionnement et permettre sa traduction en code logiciel ou en une description de matériel.

Chapitre 5

La démarche de la méthode DIAMOND

Ce n'est pas assez de faire des pas qui doivent un jour conduire au but, chaque pas doit être lui-même un but en même temps qu'il nous porte en avant.

Johann Wolfgang von Goethe

Ce chapitre expose notre méthode pour la réalisation de systèmes complexes physiques ouverts. Dans un avant-propos, nous présentons les rôles des différents intervenants de la méthode et nous présentons rapidement les normes documentaires que nous adoptons. Le cycle de vie en spirale est présenté puis le détail des processus et des activités est effectué. En fin de chapitre, une discussion de notre méthode en comparaison avec d'autres est proposée. Une synthèse des activités de la méthode figure en annexe C.

5.1 Introduction à la méthode

5.1.1 Avant propos

Les acteurs intervenants dans le développement du système. Nous allons dans la suite de cette section parler de différents acteurs humains : il est important de préciser leur rôle dans le processus de développement. Le nom des rôles et les fonctions associées sont explicités dans le tableau 5.1. Concrètement, une personne peut assumer plusieurs de ces rôles.

TAB. 5.1 – *Les acteurs humains de notre méthode de développement*

Acteur	Fonction
Demandeur	Le demandeur est la personne qui soulève le besoin en logiciel. Pour les applications industrielles il s'agit généralement du client.
Utilisateur	Le ou les utilisateurs finaux de l'application.
Expert	Certaines applications nécessitent une ou plusieurs personnes ayant l'expertise d'un domaine : l'expert est la personne qui la possède.
Analyste	Personne chargée de procéder à l'analyse du problème. Il sert généralement d'interface entre le demandeur et l'équipe de conception/développement
Concepteur	Personne qui conçoit l'application.

Normes documentaires. La documentation est un point clé d'une démarche qualité logicielle. Dans notre méthode, chaque activité fait l'objet d'un document : ils sont importants afin que toute la démarche intellectuelle puisse être retracée et qu'elle soit compréhensible. Dans un système multi-agents, comme pour de nombreuses applications, le code seul ne permet pas d'assurer une maintenance : il faut comprendre la psychologie des analystes et concepteurs.

Dans l'exposé de notre méthode, nous n'aborderons que les documents qui sont produits par les analystes et concepteurs en respectant les normes documentaires en vigueur (ISO 9000-3). Ainsi chaque document doit avoir :

– Un en-tête de page qui est un cartouche qui permet au lecteur connaître le document auquel il a à faire,
– Un corps de document qui représente la zone utilisable par le rédacteur,
– Un pied de page pour les informations complémentaires.

Pour identifier sans ambiguïté un document, nous dotons (comme il est d'usage en informatique) nos documents d'un titre spécifique, une date de création, une date de dernière modification, un numéro de version, un nom d'auteur (le responsable). Le pied de page sera laissé à l'appréciation de l'utilisateur de la méthode.

Pour les numéros de version, nous utiliserons la notation normalisée de l'ISO. Tout document peut être dans trois états différents (il peut y avoir des cycles) :

– provisoire : le document est en cours d'élaboration (version 0.0)
– à valider : le document est complet, il est en cours de validation (version 0.1)
– validé/approuvé: le document est conforme, il est diffusé pour action (version 1.0)

Afin d'améliorer la lisibilité des documents produits, nous proposons un arbre des documents produits en figure 5.1.

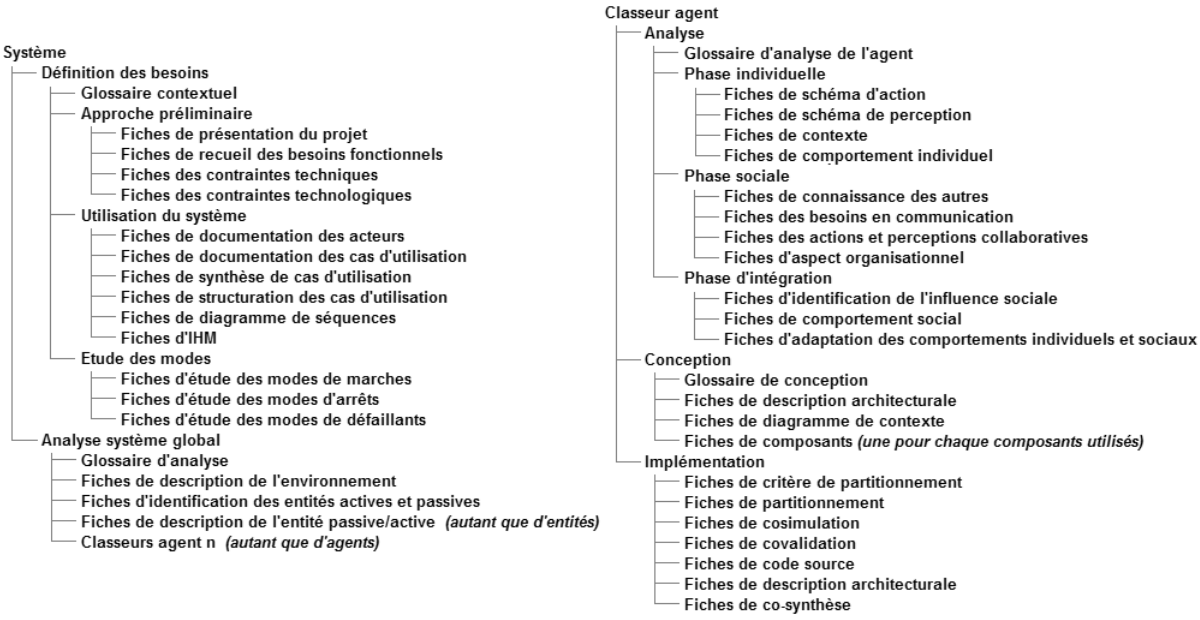


FIG. 5.1 – *Arbre des documents produits dans la méthode DIAMOND*

Organisation de ce chapitre. Après un positionnement et une présentation rapide de la méthode (en §5.1.2) et du cas d'étude (en §5.1.3) qui nous permet de l'illustrer simplement, nous insisterons, dans des sections séparées, sur ses différentes phases. Tout comme chaque section représente une phase (de §5.2 à §5.5), chaque sous-section représente un processus. Pour chacun des processus, les activités seront détaillées. La méthode exposée, une section présente une synthèse détaillée qui récapitule les objectifs de chaque étape, les documents nécessaires et produits. Enfin, une discussion comparative avec d'autres méthodes multi-agents est proposée (§5.6).

5.1.2 Présentation générale de méthode

Pour présenter au mieux la démarche générale de notre méthode, nous allons tout d'abord positionner notre approche par rapport aux autres méthodes multi-agents dans le contexte du développement de systèmes complexes physiques. Nous présenterons alors notre méthode en nous appuyant sur le cycle de vie associé à notre méthode. Nous introduisons le cas d'étude qui nous permettra d'illustrer et d'insister sur certains points de notre contribution. En fin de chapitre, nous proposons une discussion comparative de notre méthode avec des méthodes multi-agents.

5.1.2.1 Positionnement

Nous présentons dans le tableau 5.2 une analyse des cycles de vies selon des critères qui nous semblent pertinent pour une méthode co-design de systèmes multi-agents.

	Possibilité de remise en cause des besoins	Raffinement	Incrémental	Possibilité de remise en cause du produit	Gestion des risques
	OUI	OUI	OUI	NON	OUI
Code and Fix	non	**oui**	non	oui	non
Cascade	non	non	non	**non**	non
b	non	non	non	oui	non
V	non	non	non	**non**	**oui**
Incrément	**oui**	**oui**	**oui**	oui	non
Evolutif	**oui**	**oui**	**oui**	oui	non
Spirale	**oui**	**oui**	**oui**	**non**	**oui**
Y	non	non	non	**non**	**oui**
Opérationnel	non	**oui**	non	oui	non
RAD	**oui**	**oui**	non	**non**	non
XP	**oui**	**oui**	**oui**	oui	**oui**

TAB. 5.2 – *Bilan des caractéristiques des cycles de vie*

Nous avons retenu quelques critères qui nous semblent discriminants, tirés de notre expérience et de l'étude menée au chapitre précédent :

- La possibilité de remettre en cause les besoins. La classe des problèmes complexes est difficilement spécifiable. Les spécifications évoluent généralement avec la compréhension du problème qu'ont l'utilisateur mais aussi l'analyste.
- Le raffinement qui permettra une exploration efficace de l'espace de conception, c'est-à-dire de trouver un bon compromis logiciel/matériel.

– Le caractère incrémental qui milite pour la généricité.
– La possibilité de remise en cause du produit, que l'on ne souhaite pas, afin que le système mixte soit directement issu de la conception. Cela permet d'éviter des divergences au niveau des spécifications et une meilleure maintenance du système. Ce critère est d'autant plus important que la part des parties matérielles est importante dans le produit.
– La gestion des risques que nécessite la criticité des applications mais aussi la nature logicielle et matérielle du système à concevoir.

Nous avons retenu le modèle de cycle de vie en spirale. Il est à noter que les critiques les plus fréquentes de ce modèle traitent du surdimensionnement des produits. Nous pensons, qu'entre autres, la démarche qualité permettra une maîtrise d'ouvrage complète (coût, délai).

Peu de travaux abordent des systèmes multi-agents comprenant des parties physiques. Cependant, les applications nouvelles appellent à couvrir ce champ d'application (systèmes multi-agents sur PDA [Carabelea et al., 2003, Carabelea and Boissier, 2004] et applications industrielles des systèmes multi-agents [Van Dyke Parunak, 2000]). Même si nous ne sommes qu'à son début, le développement des systèmes multi-agents physiques semble tendre à s'inscrire dans le prolongement du développement traditionnel des logiciels embarqués : il aurait donc tendance à restreindre le développement de l'agent à la création du système logiciel embarqué sur cette plate-forme comme l'illustre la figure 5.2. Cette figure reprend le cycle traditionnel de développement d'un système mixte. Elle fait apparaître notre point de vue qui est que les méthodes multi-agents ne se préoccupent guère des aspects physiques de l'agent pour se consacrer aux aspects logiciels. Dans notre approche, l'agent n'est pas que la partie décisionnelle des entités du système multi-agents mais l'entité complète au sens physique du terme. Le co-design évite le partitionnement précoce dans le cycle de vie : notre approche couvre donc l'intégralité du développement du système ce qu'aucune autre méthode ne fait à ce jour.

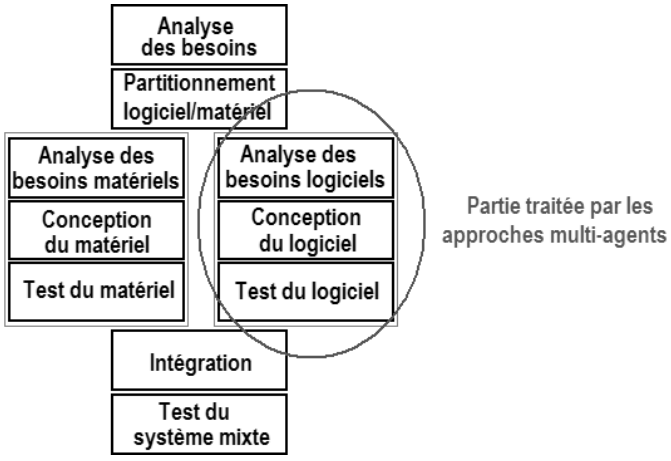


FIG. 5.2 – *Utilisation traditionnelle d'une approche multi-agents pour les systèmes physiques*

5.1.2.2 Notre démarche

Notre démarche pour la construction de systèmes complexes physiques est agencée en quatre différentes étapes. Comme l'illustre la figure 5.3, ces phases sont associées en un cycle de vie en spirale. Il

permet des itérations au niveau de l'étape de d'analyse et de la conception générique comme le spécifie [Boehm, 1988] et l'illustre la figure 5.3.

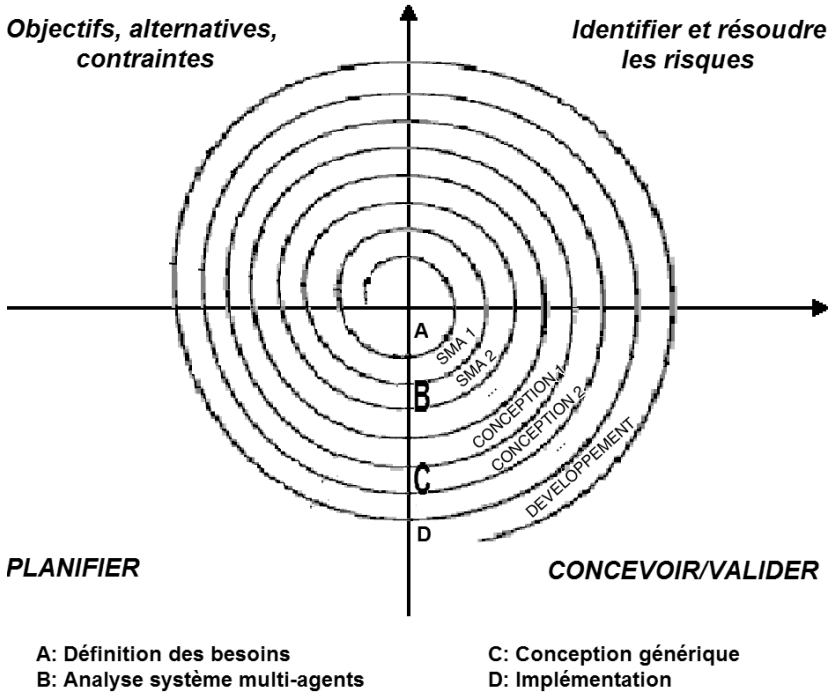


FIG. 5.3 – *Modèle en spirale de DIAMOND*

Comme l'illustre la figure 5.4, la première étape consiste en la définition des besoins (exprimer le problème à traiter), l'établissement du cahier des charges : cette partie est détaillée en §5.2. Si l'approche multi-agents est adaptée au problème préalablement spécifié, la seconde étape va permettre l'analyse de l'application à concevoir. Cette étape est détaillée en §5.3. La troisième étape de notre méthode consiste en une conception générique du système : à ce niveau du cycle de vie, on spécifie sans distinction ce qui deviendra logiciel ou matériel. Cette étape qui utilise les composants comme unité opératoire est détaillée en §5.4. Elle a donc pour objectif la construction des entités élémentaires de notre système (les agents) en respect avec l'analyse effectuée précédemment. La quatrième étape de notre cycle de vie, détaillée en section 5.5, consiste à implémenter l'application construite dans la étape précédente : le choix d'une partition logicielle/matérielle,

La présentation de la méthode DIAMOND adoptera une version tronquée du modèle en spirale tel qu'il est défini en §3.4.7. En effet, notre travail n'a pas porté sur la phase d'évaluation des risques et la planification. Pour obtenir un modèle complet , il faudra intégrer une démarche qualité. Cette démarche mettra en oeuvre des mesures notamment au niveau des prototypes afin de quantifier la qualité de la solution proposée lors de l'itération (via des simulations). Elle permettra ainsi, entre autres, d'en évaluer les faiblesses et les déviations par rapport aux spécifications.

5.1.3 Cas d'étude utilisé pour illustration

Pour illustrer les différentes étapes, processus et activités de notre méthode, nous prendrons un exemple classique d'utilisation de systèmes multi-agents physiques : l'exemple d'une équipe de robots

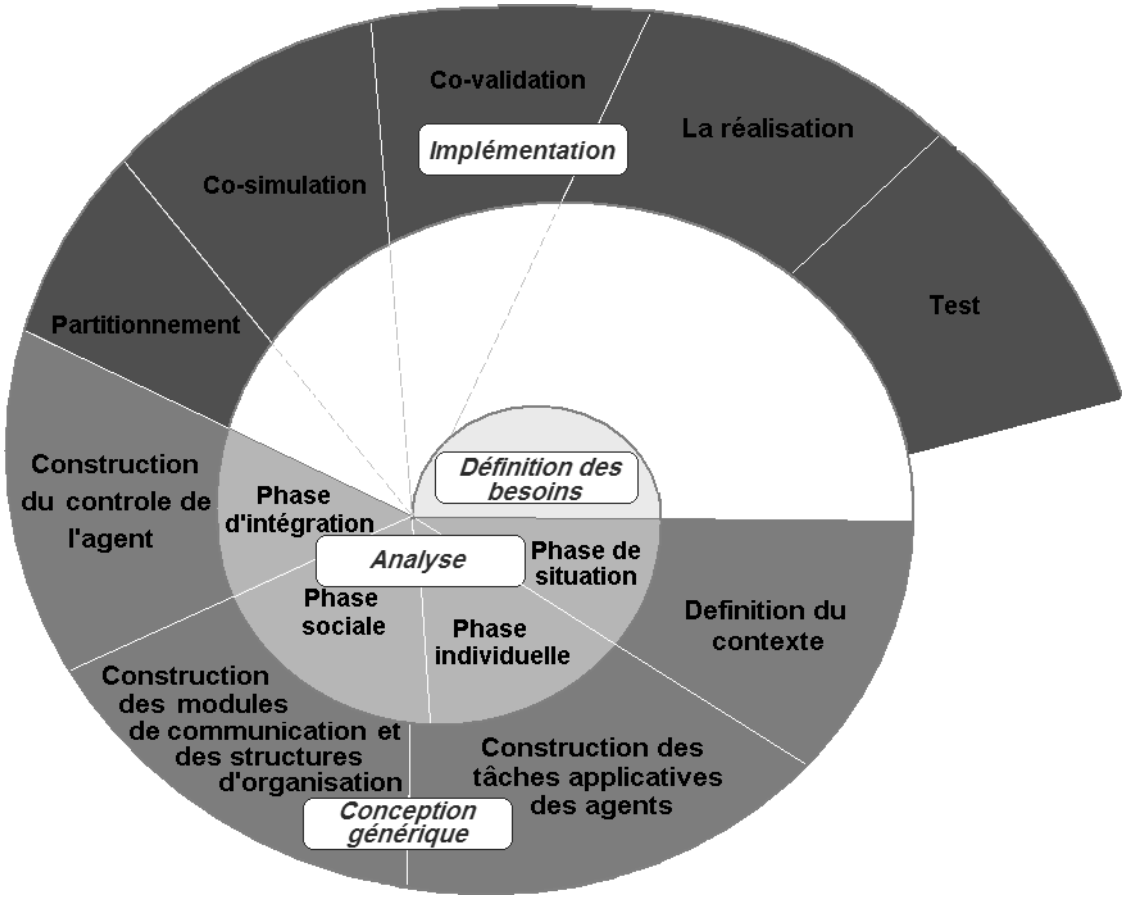


FIG. 5.4 – *Un cycle de vie pour le co-design de systèmes multi-agents*

footballeurs[1]. L'objectif de notre cas d'étude étant d'illustrer simplement la méthode, nous adopterons un cahier des charges volontairement simplifié. Les conditions expérimentales que nous adoptons sont inspirées de [Huang et al., 2001].

Des robots évoluent dans un terrain de football comme l'illustre la figure 5.5. Une caméra permet de connaître la position de chacun des robots ainsi que de la balle. Ces positions sont diffusées périodiquement à tous les membres des équipes.

Un arbitre peut arrêter le déroulement du match si une faute est commise (collision de deux robots ou sortie de la balle). S'l y eu collision, c'est le robot du camp dans lequel est la balle qui joue. Dans le cas d'une sortie de balle, c'est le robot de l'équipe non fautive qui récupère la balle et joue. Si un robot n'a plus de batterie ou dysfonctionne, le match est arrêté et le robot retiré du terrain; tous les robots doivent être alors immobiles.

Au début d'un match les robots doivent être situés dans leur camp. Le match terminé les robots qui perdent retournent immédiatement dans leur vestiaire tandis que les robots victorieux restent sur le terrain ; ils sont généralement rejoints par leur concepteur (après leur mise hors-service).

L'équipe qui a marqué le plus de buts en 90 minutes a gagné.

1. C.f. le site de la RoboCup World Championship http://www.robocup.org/

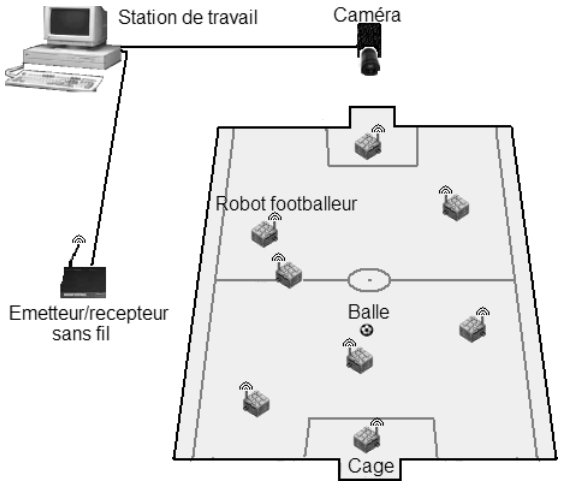


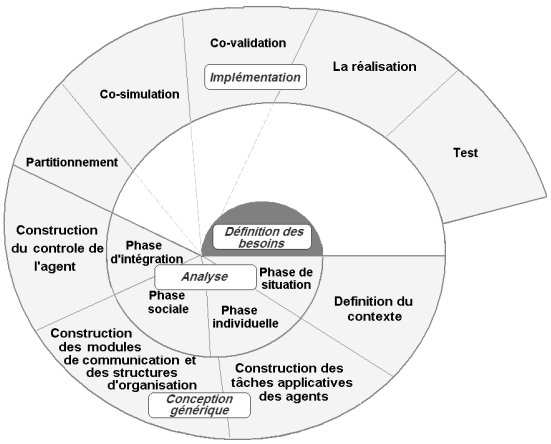
FIG. 5.5 – *Illustration de l'application équipe de robots footballeurs.*

5.2 La définition des besoins (étape A)

Toute méthode complète d'analyse et de conception s'intéresse tout d'abord au recueil des besoins des utilisateurs. Cette étape a pour objectif d'exprimer un problème.

Cette étape est la première du cycle de vie de l'application mixte logicielle/matérielle. Elle a pour objectifs de déterminer avec précision les attentes du demandeur et des utilisateurs du futur système ainsi que d'exprimer aux concepteurs le problème. Pour cela, on établit un document qui permettra aux analystes de comprendre les fonctionnalités à la fois globales du système et les tâches plus spécifiques qu'il doit réaliser.

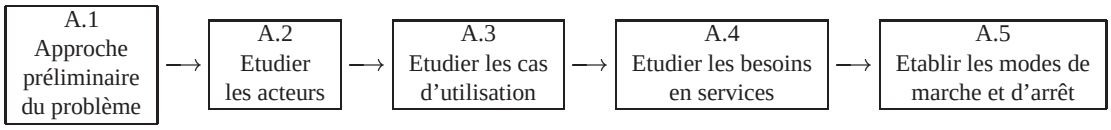
Cette étape est constituée de plusieurs activités qui permettront d'établir un cahier des charges fonctionnel suffisamment complet afin de déterminer, en premier lieu, si l'approche multi-agents convient au problème. Si oui, la suite de la méthode pourra être appliquée. Dans le cas contraire, il doit permettre de choisir la solution méthodologique/technologique à utiliser.



Tout d'abord, il est nécessaire à l'analyste de préciser de manière informelle le problème, de s'en imprégner, d'en avoir une vue globale même imprécise (A.1). On identifie ensuite les différents intervenants

du système (A.2). Le rôle de ces acteurs, qu'ils soient humain ou non , doit être spécifié. On identifie alors le rôle global du système et il faut donc identifier les différents cas d'utilisation de l'application (A.3), les décrire précisément et les hiérarchiser. Tout cela permettra de construire le *diagramme de cas d'utilisation*. Les besoins en services des différents acteurs doivent être aussi précisés (A.4). Vu que le système est en partie physique il est important d'identifier ses différents modes de marche et d'arrêt (A.5) afin de spécifier les contraintes que l'on peut avoir selon les modes de fonctionnement comme pour les arrêts d'urgence.

Cette partie est très informelle bien qu'utilisant des présentations formatées (cf §3.5.2). Elle nécessite donc beaucoup de rigueur et beaucoup de recul de la part des personnes qui conçoivent les différents documents. Tout document doit être approuvé par toutes les parties prenantes du projet et si nécessaire des besoins consensuels seront établis.



5.2.1 Approche préliminaire du problème (A.1)

Nous allons nous intéresser, dans cette activité, à toutes les opérations utiles pour comprendre les besoins des utilisateurs et décrire de manière informelle le problème que souhaite résoudre le demandeur. Pour identifier les attentes des différentes parties prenantes (le demandeur et les utilisateurs), plusieurs techniques peuvent être mises en oeuvre : les interviews, l'observation du processus à automatiser (ainsi que les opérateurs), l'utilisation d'experts, le maquettage (confronter un utilisateur à une maquette afin de mettre en exergue des manquements), l'identification des flux de travail etc. Il peut être important de déterminer l'ontologie du domaine ou plus simplement d'établir un glossaire qui permettra de décrire les principaux concepts et le vocabulaire utilisé dans la description des besoins et des cas d'utilisation. A l'issu de ce processus, il est important de s'assurer de l'adéquation entre la problématique et l'utilisation du paradigme multi-agents [Picard, 2004]. Même si c'est essentiellement l'expérience de l'analyste qui peut le vérifier, on peut identifier et vérifier que les caractéristiques de la problématique font pressentir cet intérêt (voir [Boissier et al., 2004]).

Documentation liée à ce processus A.1 . A ce niveau de l'étude on établit plusieurs fiches :

- une *fiche de présentation du projet* qui résume, présente le contexte du projet (demandeur, problématique etc.),
- une *fiche de recueil des besoins fonctionnels* (figure D.2) qui est une synthèse des besoins exprimés dans le cahier des charges,
- une *fiche des contraintes techniques* qui permet d'insister sur des points particuliers tels que la sécurité,
- une *fiche des contraintes technologiques* pour le cas où le client impose certains matériels existants ou émet des préférences,
- un *glossaire contextuel* (figure D.1) qui permet de préciser la définition du vocabulaire voire d'aider, si nécessaire, à la construction d'une ontologie du domaine.

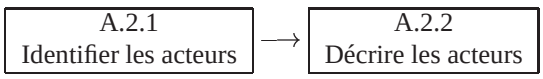
Ces fiches pourront être complétées dans les autres activités de la phase de définition des besoins.

Application au cas d'étude : Premier contact avec le cahier des charges et glossaire

On va prendre connaissance du cahier des charges et éclaircir toutes les zones d'ombre en interagissant avec le demandeur (ou dans le cas d'une compétition de ce type la personne qui établit les règles). Dans un exemple aussi simple que le notre, on peut se limiter à une ré-écriture différente du cahier des charges et expliciter les termes mis en jeu.

- Robot : Tous les équipements du robot (partie commande, capteurs et actionneurs) doivent entrer dans un cylindre de diamètre 20 cm et d'une hauteur de 25 cm. Un tag est apposé au dessus de chacun d'eux afin de permettre leur identification par la caméra.
- Arbitre : Ici c'est un logiciel qui a plusieurs fonctions. En début de match il assigne le rôle de gardien à un robot et un terrain à chaque équipe. Il veille à ce que la balle ne traverse pas les lignes qui délimitent le terrain. Il doit aussi vérifier qu'avant chaque remise en jeu (après un but) tous les robots soient dans leur partie du terrain. Il permet aussi à un opérateur humain d'effectuer un changement de robot dans une limite de deux par match. Enfin l'arbitre valide chaque but et déclare le vainqueur au bout de quatre-vingt-dix minutes de jeux.
- Point: Un point est marqué quand la balle pénètre dans les caisses (voir 5.5). L'arbitre valide le point et chacune des équipes retourne dans son camp.
- Terrain : Les limites du terrain sont blanches (seuls éléments de cette couleur sur le terrain) afin de faciliter leur reconnaissance par la caméra.
- Surface de réparation : Surface dans laquelle seul le gardien peut entrer.
- Cage : zone dans laquelle doit entrer la balle pour que le but soit pris en compte. Un point sera ainsi obtenu.
- Gardien : Le gardien est le seul robot qui a le droit de s'interposer entre la balle et les caisses dans la surface de réparation. Aucun autre robot ne peut pénétrer dans cette surface pour arrêter la balle. Tout robot doit pouvoir assumer ce rôle désigné par l'arbitre en début de rencontre.

5.2.2 Etudier des acteurs (A.2)



Identifier les acteurs (A.2.1). Les acteurs représentent en premier lieu les différentes personnes qui interagissent avec le système en cours de modélisation. A une même personne peuvent être associés plusieurs acteurs selon les rôles fonctionnels qu'ils jouent sur le système. Un acteur peut cependant être une machine extérieure au système mais qui interagit avec ce dernier. De la même manière que pour une personne, à une entité peut être associée plusieurs acteurs d'un système.

Décrire les acteurs (A.2.2). Il est nécessaire pour ne pas laisser de multiples interprétations des rôles joués par les acteurs, de décrire leurs fonctions, les fonctionnalités qu'ils demandent au système et des

fonctionnalités que le système leur demande.

Documentation liée à ce processus A.2 . En s'appuyant notamment sur les *fiches de présentation du projet* et les *fiches de recueil des besoins fonctionnels* on détermine les acteurs. Chacun des acteurs devra être documenté via une *fiche de documentation des acteurs* (figure D.3) qui insiste sur la définition du rôle des acteurs.

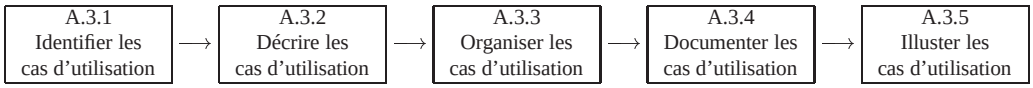
Application au cas d'étude : Qui interagit avec le système? Pourquoi?

Les acteurs concernés sont :

1. L'*arbitre* qui :
   - gère des paramètres du match en désignant un gardien et en assignant un camp à une équipe.
   - gère le temps en signalant quand le match commence et finit.
   - veille au respect des règles de jeu en prenant garde de sanctionner les collisions de robots et les sorties.
   - veille à ce que les robots soient arrêtés quand un opérateur intervient dans l'aire de jeu.
2. Le *manager* qui est chargé d'évacuer les robots en panne d'énergie,
3. Le *ballon* qui se déplace sous l'action des robots.
4. L'*équipe adverse* qui agit sur le ballon et occupe une place sur le terrain.
5. La *système caméra* qui envoie les positions de chaque robot et du ballon.

5.2.3 Etudier les cas d'utilisation (A.3)

L'objectif de cette étape est la caractérisation des différents cas d'utilisation du système c'est-à-dire ses grands objectifs. Elle est organisée en quatre étapes.



Identifier les cas d'utilisation (A.3.1). Les cas d'utilisation représentent les objectifs majeurs du système à modéliser : elle représente un ensemble de séquence d'actions. Il ne s'agit en aucun cas de tâches (granularité trop basse) mais bien des services rendus par le système voire un comportement attendu par le système.
A la différence des diagrammes de cas d'utilisation, tels qu'ils sont utilisés dans UML, le système ne se résume pas à un système logiciel. Aussi, les acteurs sont forcément extérieurs au système. Dans un diagramme UML un effecteur utilisé par le système serait à modéliser comme étant un acteur secondaire, ce qui n'est pas le cas ici. En effet, UML focalise sur le système logiciel alors que notre objet d'étude est le système mixte qui est donc vu comme un tout.

Décrire les cas d'utilisation (A.3.2). Chaque cas d'utilisation doit être décrit précisément. Pour cela, on pourra avoir une vue centrée acteur (est-il l'acteur principal c'est-à-dire celui qui déclenche le

cas ou non?) et aux messages émis ou reçus par les acteurs qui collaborent aux présents cas (un tableau de synthèse sera créé). Enfin, d'une manière plus traditionnelle on peut avoir une vue centrée cas qui permet de connaître les conditions qui provoquent le déclenchement et l'arrêt du cas ainsi que sa chronologie (un tableau de synthèse sera lui aussi créé).

Organiser les cas d'utilisation (A.3.3). Les différents cas d'utilisation peuvent être organisés selon deux façons complémentaires. On peut tout d'abord établir les éventuelles relations qui lient les cas (inclusion, extension ou généralisation). Enfin, on peut regrouper ces cas en packages "fonctionnels".

Documenter les cas d'utilisation (A.3.4). La description textuelle (A.3.2) est efficace pour communiquer avec les utilisateurs et s'entendre sur la terminologie du domaine. Cependant, il est nécessaire d'avoir recours à d'autres outils pour montrer comment les enchaînements se succèdent ou permettre une maintenance efficace de l'application. Pour cela on dispose des *diagrammes d'activité* qui permettent de consolider les enchaînements de la fiche descriptive textuelle du cas. L'alternative est le *diagramme d'état* qui se prête mieux à la modélisation événementielle. Cependant il est plus difficile à appréhender par l'utilisateur.

Illustrer les cas d'utilisation (A.3.5). Les cas d'utilisation, une fois documentés (A.3.4), doivent être illustrés avec des scénarios particuliers comme les scénarios nominaux qui caractérisent une marche normale de l'application. Pour ce faire on dispose des *diagrammes de scénario* facilement compréhensibles par les utilisateurs et les *diagrammes de collaboration* qui sont moins compréhensibles par les utilisateurs.

Documentation liée à ce processus A.3 . A partir de la *fiche de présentation du projet*, de la *fiche de recueil des besoins fonctionnels*, de la *fiche des contraintes techniques* et des *fiches de documentation des acteurs*, on détermine les cas d'utilisation. Chaque cas est documenté via une *fiche de documentation de cas d'utilisation* (figure D.4). qui indique les acteurs qui interagissent avec le cas étudié. Une synthèse des cas apparaîtra dans la *fiche de synthèse des cas d'utilisation* (figure D.5). Ces cas d'utilisation organisés via la *fiche de structuration des cas d'utilisation* (figure D.6). Chacun des packages, qui regroupent les cas d'utilisation par thèmes, devra être illustrer par un diagramme de cas d'utilisation. Sur cette fiche apparaîtra le nom du cas, son but, son résumé et les acteurs qui interagissent avec ce cas, les pré-conditions, enchaînements nominaux et alternatifs et exceptions. Ces fiches sont tirées de [Roques and Vallée, 2003].

Application au cas d'étude : Quels sont les différents cas d'utilisation du système?

Les différents cas d'utilisation du système sont le *paramétrage* (choix d'un gardien et d'un camp) et le *jeu*.

Dans le cas d'utilisation *paramétrage*, l'arbitre commence par vérifier que chaque robot/joueur est dans son camp. Ensuite l'arbitre désigne aléatoirement un gardien. Ce dernier ira se placer dans la surface de réparation. La balle est donnée à un des robots d'une des équipes tirées au sort. Un des membres de l'équipe ira se positionner au centre du terrain. Cette opération effectuée, le coup d'envoi est donné.

Le cas d'utilisation *jeu*, commence une fois que le coup d'envoi a été donné. Chaque équipe tente de marquer des points. Quand une balle entre dans une cage il y a but. Les équipes retournent alors dans leur camp. L'équipe qui a encaissé le but se saisit de la balle et va au centre. Si une balle traverse la ligne de sortie le dernier à l'avoir touchée est fautif : l'autre camp rejoue la balle à l'endroit indiqué par l'arbitre. A la mi-temps, l'arbitre ordonne un changement de camp.

Dès le paramétrage fini, on passe à la phase de jeu : le cas d'utilisation paramétrage déclenche le cas d'utilisation jeu (voir paragraphe 5.6).

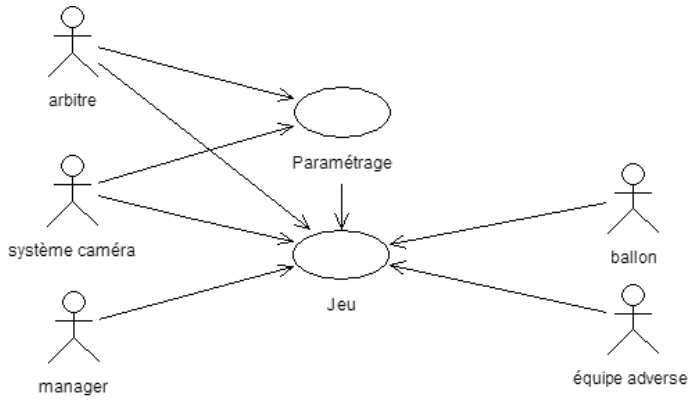


FIG. 5.6 – *Cas d'étude - Diagramme de cas d'utilisation*

5.2.4 Etudier les besoins en services (A.4)

L'objectif de cette étape est de préparer la spécification des besoins en services de chacun des acteurs. Les diagrammes de séquence, issus d'UML, permettent aux analystes de concevoir des scénarios d'utilisation du système.

Ils permettent de représenter simplement les échanges de messages entre le système et les acteurs qui interagissent avec lui.

Certains acteurs du diagramme des cas d'utilisation peuvent ne pas figurer sur les diagrammes de séquence (comme traditionnellement dans les applications purement logicielles). Cela tient du fait que des interactions purement physiques peuvent avoir lieu, comme par exemple, un choc.

Documentation liée à ce processus A.4 . Ce processus s'appuie sur les fiches liées aux acteurs et aux cas d'utilisation. Des *fiches de diagramme de séquence* permettront de mettre en évidence les besoins en services (les interactions) entre les acteurs et les différents cas d'utilisation. Ainsi les fiches correspondantes seront utilisées.

Des *fiches d'IHM* peuvent éventuellement être réalisées. Elles relèvent bien de la détermination des besoins fonctionnels car leur objectif et de présenter au demandeur l'ensemble des interactions qu'il peut faire. Ces interfaces ne sont pas forcement logicielles (copie d'écran d'interface graphique attendue par exemple) mais peuvent être matérielles (dessin d'un panneau de commande avec interrupteurs et voyants).

Application au cas d'étude : Etude des besoins en services

1. Diagramme de séquence associé au cas d'utilisation "paramétrage" (figure 5.7)

Il est important de bien avoir à l'esprit les limites du système. Ce que l'on doit construire est une équipe de robots et non le système permettant la mise en compétition avec une autre. Ainsi pour ce qui concerne ce cas d'utilisation l'acquisition des positions des différents robots par l'arbitre est par exemple hors sujet.

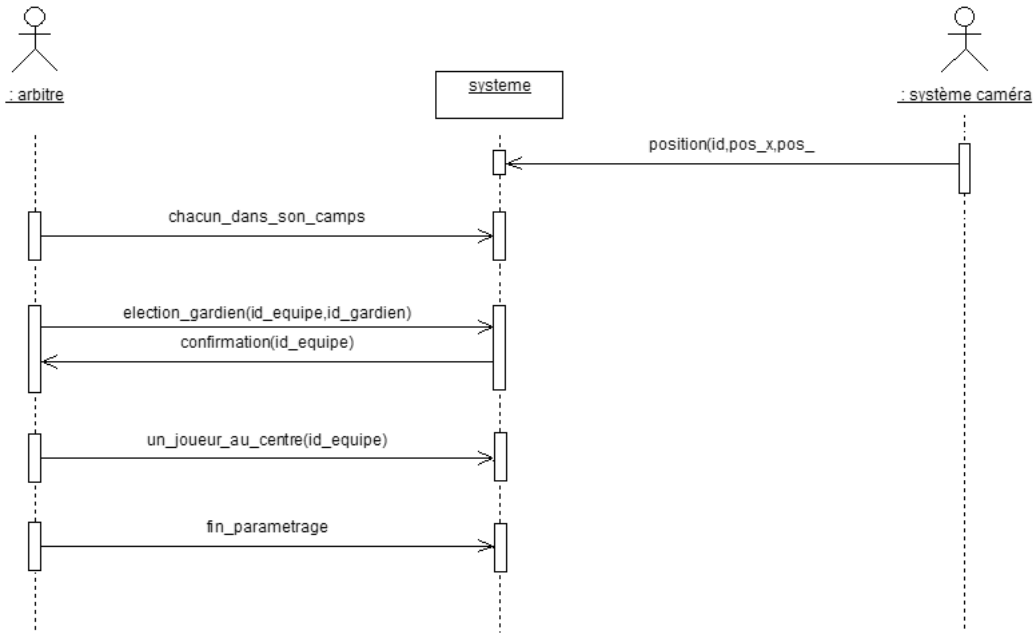


FIG. 5.7 – *Cas d'étude - Diagramme de séquence du cas paramétrage*

2. Diagramme de séquence associé au cas d'utilisation "jeu" (figure 5.8)

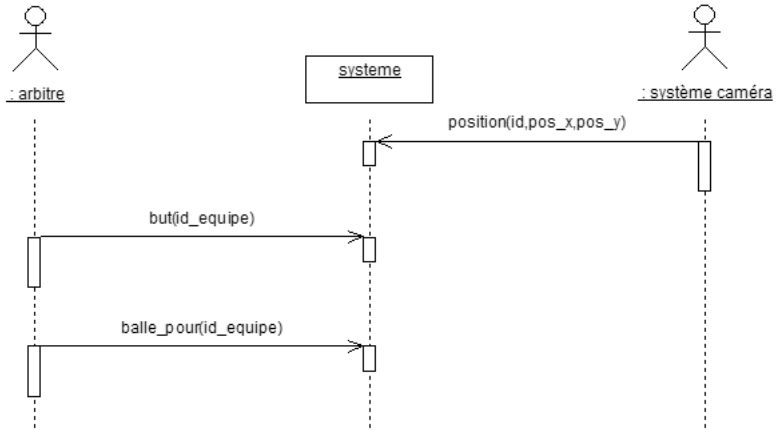


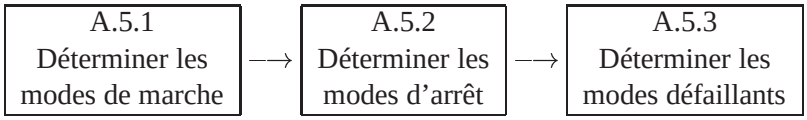
FIG. 5.8 – *Cas d'étude - Diagramme de séquence du cas jeu*

Les acteurs "ballon" et "équipe adverse" apparaissent sur le diagramme de cas d'utilisation et non

sur le diagramme de séquence.

5.2.5 Etablir les modes de marche et d'arrêt (A.5)

Les applications envisagées avec notre méthode étant à la fois logicielles et matérielles, il est nécessaire de prendre en considération les modes de marche et d'arrêt. Cette activité est inspirée du GEMMA (Guide des Modes de Marche et d'Arrêt) qui est un guide permettant de structurer graphiquement le fonctionnement des systèmes automatisés de production (SAP) [Moreno and Peulot, 1997]. En effet, le GEMMA a pour vocation de regrouper par thème les états d'un SAP qui serait vu comme une machine à états finis. A ce titre, le GEMMA est utilisé en collaboration avec les GRAFCET. Nous en avons fait un outil textuel pour spécifier des modes de fonctionnement important du système. Des modes ont été supprimés car ils étaient trop spécialisés pour les SAP tandis que d'autres ont été étendus ou ajoutés.



Cette partie de l'étude est très importante car elle va permettre de structurer le fonctionnement global du système. Si généralement on souhaite que le système fonctionne en autonomie, il est nécessaire de connaître précisément tous les autres comportements : comment doit être le système avant l'arrêt ? comment doit être le système pour être étalonné/réglé ? comment prendre en compte les arrêts d'urgence? Même si le problème sera par la suite résolu de manière décentralisée, cette organisation des modes de fonctionnement présente l'avantage d'être facilement interprétable par le demandeur et l'utilisateur. De plus, bien qu'à intelligence décentralisée le système doit se soumettre à une législation qui impose, à raison, la présence d'arrêts d'urgence pour les systèmes physiques pouvant nuire à l'intégrité d'une personne ou d'un bien. Ainsi on établit des procédures de marches, de défaillances et d'arrêts. On peut aussi spécifier les transitions entre ces modes. Pour chacun de ces modes, un document doit préciser une description de l'état attendu du système (partie informelle) et formuler les propriétés qui permettront de caractériser l'état du système. On va donc déterminer les propriétés qui, selon qu'elles soient ou non respectées, feront que l'on sera dans un état de marche normal, défaillant ou d'arrêt.

Cette définition des modes pourra en plus, par la suite :

- mettre en exergue des modes de fonctionnement dégradés du système,
- spécifier les premiers éléments nécessaires à la tolérance aux pannes,
- permettre l'identification de situations de coopération ou non coopération,
- permettre de définir des états de reconnaissance utiles pour l'analyse d'un processus auto-organisationnel d'une application,
- prendre en compte la sécurité de l'intégrité physique de l'utilisateur éventuellement plongé dans le système physique.

Déterminer les modes de marche (A.5.1). Les modes de marches définissent les états de fonctionnement du système mais insistent aussi sur les états d'étalonnage ou de réglage du système. A ce niveau de l'étude on n'envisage pas forcément des réglages automatiques : on souhaite obtenir du demandeur

des informations très importantes qu'il ne penserait pas forcément à fournir. Selon l'application, il se peut que des modes soient inutiles voir sans aucun sens.

Les différents modes de marche que l'on peut identifier sont présentés ci-dessous :

- Marche normale : Nous allons nous intéresser dans ce mode au fonctionnement normal du système à concevoir. Ce mode a surtout un intérêt pour la sûreté de fonctionnement. On définira par exemple les propriétés que doit vérifier le système qui prouve son fonctionnement normal.
- Marche de préparation : La spécification de ce mode permet de préciser les contraintes à prendre en compte pour amener à une marche normale. Au niveau global cela se traduit généralement par des conditions sur, par exemple, les ressources disponibles à l'intérieur du système. Au niveau local, cela se traduit par des contraintes de fonctionnement pour les composants du système : un robot ne peut être disponible que si son niveau d'énergie est supérieur à un seuil, l'utilisation d'un four peut nécessiter une préchauffe d'une durée donnée etc. Comme on le pressent, l'étude de ce mode sera surtout utile pour les composants physiques du système.
- Marche de clôture : La spécification de ce mode permet de définir l'état que doit atteindre le système avant un arrêt prolongé. On va donc s'intéresser aux actions à effectuer pour pouvoir procéder à un arrêt du système : des robots mobiles peuvent avoir à se rendre dans un espace de stationnement etc.
- Marche de vérification en marche : Dans cet état on va s'intéresser à la possibilité/nécessité de vérifier les différentes primitives des composants du système ou de fonctionnalités globales du système. De telles procédures peuvent être nécessaires pour vérifier le bon fonctionnement ou isoler un problème.
- Marche de vérification en hors-marche : Dans cet état on va s'intéresser aux procédures de vérifications beaucoup plus complètes mais qui nécessitent que le système suspende temporairement son fonctionnement normal. Ces procédures ne concernent pas seulement la vérification mais les modes de suspension.
- Marche d'étalonnage : La spécification de ce mode permet de définir le protocole de réglage ou d'étalonnage d'un système.

Déterminer les modes d'arrêt (A.5.2). L'étude des modes d'arrêt va permettre tout d'abord d'identifier les états d'arrêt qui peuvent survenir pour des raisons extérieures au système. On définit aussi les éventuelles procédures qui les accompagnent. Ces états permettent d'amener le système à un arrêt normal ou de préparer le système à une procédure de re-initialisation. Selon l'application, il se peut que des modes soient inutiles voire sans aucun sens.

Les différents modes d'arrêt que l'on peut identifier sont :

- Repos : Il s'agit de caractériser ici l'état dans lequel se trouve le système lorsqu'il est au repos c'est-à-dire en attente de tâches à effectuer.
- Arrêt demandé en marche normale : La spécification de ce mode d'arrêt permet de s'intéresser à la suspension du fonctionnement normal sans pour autant procéder à une interruption brutale du système. Cela peut se limiter à identifier les états durant lesquels on peut arrêter le système et donc s'intéresser aux procédures pour l'y conduire. Ce mode est par exemple intéressant pour des systèmes utilisant des ressources finies. En effet, si un système peut éventuellement nécessiter qu'on interrompe son fonctionnement normal afin de le ré-alimenter. Par souci de sécurité il se peut que cette opération ne soit possible que dans des états particuliers du système.
- Arrêt demandé dans un état déterminé : La spécification de ce mode d'arrêt permet de conduire le système à un arrêt différent du précédent afin que, par exemple, une équipe de maintenance intervienne sur le système.

– Préparation pour remise en route après défaillance : La spécification de ce mode permet de ramener le système après une défaillance dans un état qui lui permettra de remettre en route le système. Cela peut se faire de manière automatique ou, éventuellement, nécessiter que des opérateurs interviennent pour corriger le système (initialiser un élément etc.).

Déterminer les modes défaillants (A.5.3). L'étude des modes défaillants va permettre de spécifier les procédures de sécurité permettant de réagir lorsqu'une défaillance est rencontrée. Les différents modes défaillants qu'il est possible d'identifier sont:

– Marche ou arrêt en vue d'assurer la sécurité : Cet état permet de gérer le système lors d'un arrêt d'urgence. On prévoit dans cet état toutes les mesures visant à protéger l'utilisateur et le système, les cycles de dégagements et les précautions pour limiter les conséquences d'une défaillance.
– Diagnostic et/ou traitement de défaillance : Ce mode traite de ce qui permet à la maintenance de diagnostiquer l'origine de la défaillance et d'envisager le traitement approprié qui permettra le redémarrage du système après traitement de la défaillance. Dans ce mode, le système est arrêté.
– Marche dégradée : Cet état permet de permet de passer outre une défaillance non résolue du système.

Documentation liée à ce processus A.5 . A partir de toutes les informations cumulées, et donc toutes les fiches, on renseigne trois fiches : une *fiche d'étude des modes de marche* (figure D.7), une *fiche d'étude des modes d'arrêt* et une *fiche d'étude des modes défaillants*. Ces fiches doivent être validées par le demandeur et les utilisateurs du système à concevoir. En effet, un des intérêts de ces fiches est qu'elles sont compréhensibles par des non-informaticiens.

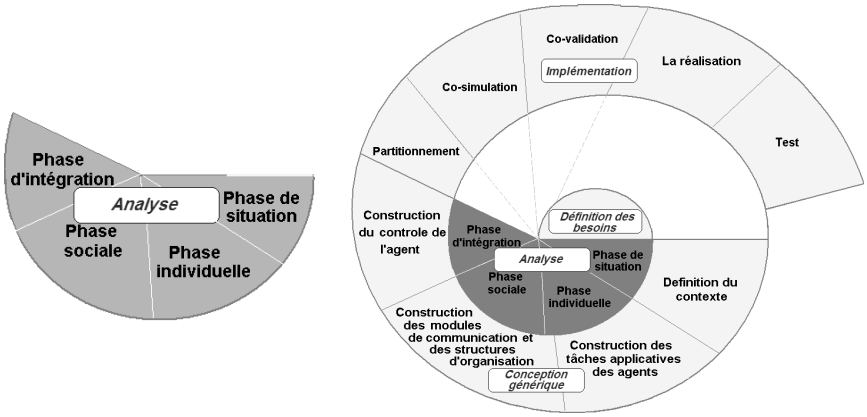
Application au cas d'étude : Structurer le fonctionnement global du système

Pour notre application, après discussion avec le demandeur, les modes identifiés sont :

1. Mode d'arrêt : Deux modes sont pertinents. Dans cette application les autres modes ne sont pas exploités.
   – Repos : Dans un état de repos, les robots sont immobiles.
   – Arrêt demandé en marche normale : Si on doit le ré-alimenter un robot doit se positionner dans un coin de son camp (coté cage) en attendant la mi-temps que tous les robots soient immobiles.
2. Modes de marche : le mode de vérification en marche normale n'a aucun sens car pendant un match on ne peut pas procéder à une vérification. On peut par contre procéder à des vérifications hors marche normale.
   – Marche normale : dans ce mode tous les robots doivent répondre aux requêtes de l'arbitre, aucun robot ne signale de problème, il n'y a pas d'arrêt d'urgence.
   – Marche de préparation : Durant la phase de préparation les robots sont positionnés sur le terrain. Dans ce mode les robots ne doivent ni bouger ni utiliser leurs actionneurs. Ce mode prend fin quand l'arbitre déclare la période de paramétrage commencée.
   – Marche de clôture : avant un arrêt prolongé les robots doivent retourner dans le camp et se rapprocher des coins pour que leur extraction soit facilitée.
   – Marche de test : on peut vouloir étalonner la puissance maximale du tir. En effet, par souci de sécurité les vitesses maximales des robots et de la puissance de tir sont limitées. De plus la vitesse d'éjection de la balle n'est pas la même si on fait une passe ou un tir.

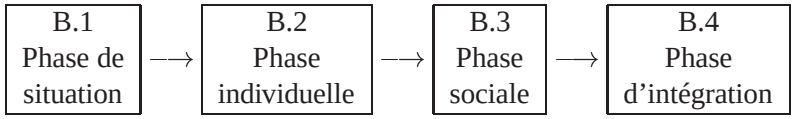
3. Modes défaillants : seul la gestion des arrêts d'urgence est pertinente dans notre application.
   – Marche ou arrêt en vue d'assurer la sécurité : Si un arrêt d'urgence est activé, les robots n'ont plus le droit de bouger.

5.3 L'étape d'analyse (étape B)



Vu le caractère mixte logiciel/matériel, il nous faut dans cette démarche, étudier une spécialisation de la définition des différents axes de la décomposition AEIO pour les étendre à la description du "matériel" (il est à noter qu'à ce niveau de la méthode, aucune frontière logicielle/matérielle fixe n'existe pour ce qui concerne la spécification du contrôle). Une des premières questions à élucider est celle qui concerne l'agencement des axes. D'une manière générale, notre expérience nous conduit à penser que l'ordre d'approche des axes dans la décomposition multi-agents dépend des problèmes et des domaines abordés [Occello, 2003]. Lorsque l'on travaille sur un système physique il nous semble en fait assez naturel de commencer par s'intéresser d'abord à l'**E**nvironnement dans lequel les entités évoluent, puis à l'axe **A**gent, aux **I**nteractions, et enfin l'**O**rganisation.
L'itération multi-agents de la spirale regroupe quatre différentes activités que nous abordons par la suite :

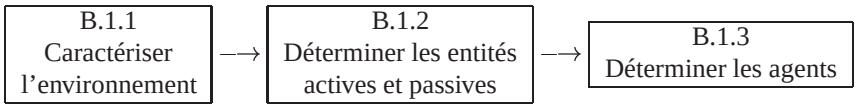
- une phase de situation pour définir les limites du système multi-agents, de l'environnement (B.1),
- une phase individuelle pour s'intéresser aux agents de façon interne (B.2),
- une phase sociale pour les interactions et l'organisation (B.3),
- une phase d'intégration pour la socialisation des agents c'est-à-dire l'intégration des influences sociales dans les agents (B.4).



5.3.1 Phase de situation (B.1)

Cette activité consiste à définir les limites de l'application, à caractériser les agents et l'environnement.

Dans notre contexte, c'est l'analyse de l'environnement qui est l'objet de la phase de situation, qui pose les bases de la décomposition. Nous allons donc nous intéresser dans un premier temps à déterminer les frontières de l'application et caractériser quels éléments de l'environnement il est pertinent de prendre en compte dans la future application.



Caractériser l'environnement (B.1.1). On va s'intéresser aux caractéristiques de l'environnement (vu en §2.2.3.2) du système qui seront autant d'informations précieuses pour le concepteur. Elles portent principalement sur le contexte de conception.

- L'*accessibilité* - Dans le monde réel, on ne peut pas tout connaître de l'environnement mais, suivant les applications, on peut mesurer via des capteurs des paramètres qui suffiront à déterminer un état de reconnaissance suffisant pour l'application envisagée.
  On détermine tout d'abord le degré d'accessibilité de l'environnement. Autrement dit que peut-on percevoir des différents éléments de l'environnement qui nous intéresse? Quelle incertitude sur ces mesures? Quel impact?
  Si ces mesures suffisent à reconnaître un état de l'environnement (accessible), on va ainsi préciser les primitives de perception nécessaires aux agents. Il faudra penser à doter l'agent de moyens de mémorisation (afin qu'il puisse construire la représentation de l'environnement) et de raisonnement pour qu'il puisse raisonner sur cette représentation (conditionne l'aspect décisionnel de l'agent).
  Si l'environnement ne s'avère pas accessible, peut-on raisonnablement supposer sur l'information inaccessible de l'environnement à partir de la perception des différentes évolutions d'autres paramètres? Tant que l'environnement ne sera pas suffisamment accessible, il sera nécessaire de demander des précisions aux demandeurs et aux utilisateurs.
- Le *déterminisme* - Pour mesurer le degré de déterminisme de l'environnement il faut s'interroger sur ce que l'on peut physiquement contrôler de cet environnement. Les environnements réels sont très rarement déterministes. On peut donc, lorsqu'on s'intéresse à ce critère, commencer à s'interroger sur les effecteurs de l'agent.
- *Episodique* - Les environnements réels sont généralement épisodiques. Cette caractéristique aura une influence directe sur les buts des agents qui ont pour objectif le contrôle de cet environnement.
- *Statique* - Les environnements réels sont quasiment toujours dynamiques mais c'est au concepteur d'apprécier le degré de dynamique de la partie de l'environnement qui l'intéresse.
  Ce paramètre influe sur l'architecture des agents. Ainsi, pour des environnements réels, elle favorisera des architectures réactives et hydrides.
- *Discret* - Comme toujours, ce critère est laissé à l'appréciation du concepteur suivant l'application qu'il envisage. Un environnement réel est quasiment toujours continu.

Déterminer les entités actives et passives (B.1.2). Dans un système multi-agents physique, la présence ou non d'adversaire relationnel, joue sur les stratégies à mettre en oeuvre dans l'aspect décisionnel de l'agent. Dans le monde réel il faut penser aux possibilités d'interférences avec d'autres systèmes multi-agents.

Dans cette activité, on identifie les entités actives et passives qui composent le système. Une entité active possède de l'autonomie. Cette autonomie peut être, par exemple, vis-à-vis de son environnement et des autres entités. Elle peut ainsi régir des règles d'interactions entre les différentes entités. Une entité passive ne possède pas ce pouvoir de dire non que lui confère l'autonomie : elle répond quasiment uniformément aux sollicitations des autres entités. Généralement les entités passives sont des ressources du systèmes.

Les entités peuvent être contenir du logiciel (active: un dispatcheur de services, passive: un fichier) et du matériel (active: un robot, passive: une balle). Une entité peut aussi se présenter plus simplement sous la forme d'un jeu de contraintes qui module des interactions. Pour chacune de ces entités il est nécessaire de lui trouver un nom, de préciser son rôle et sa raison d'être. Il va être nécessaire d'identifier les interactions entre ces entités et l'environnement. A partir des interactions entre les utilisateurs et le système (identifiées en §5.2.4), via les *fiches de diagramme de séquence*, on va identifier les entités qui participent aux interactions entités/utilisateurs.

Déterminer les agents (B.1.3). A ce niveau de l'analyse, il va falloir choisir les entités que l'on modélisera par des agents. Pour cela, on s'appuie sur les descriptions faites lors de l'activité précédente. On va s'intéresser pour cela essentiellement aux entités actives (les entités passives seront essentiellement étudiées par la vue qu'en auront les entités actives). Selon la complexité des entités traitées, il sera possible de l'éclater en un ensemble d'agents : une entité active sera donc modélisable par des agents coopérants.

Cette activité fait appelle a l'expérience de l'analyste. Nous travaillons cependant pour assister l'analyste dans cette opération via la démarche qualité qui sera mise en place. En effet, nous pensons qu'en associant à chaque entité des mesures permettant d'évaluer la contribution à la qualité globale du système, il sera possible d'assister le choix des agents.

Documentation liée à ce processus B.1 . Dans cette partie de l'analyse on s'appuie sur les fiches résultants de l'approche préliminaire. Pour identifier les entités actives et passives on se servira aussi des fiches issues de l'étude des acteurs, des cas d'utilisation et de l'étude des modes de fonctionnement. Il en est de même pour l'identification des agents.

Pour ce qui concerne la production de documents, on va créer plusieurs fiches dans la partie "analyse" de la documentation :

- On commence aussi à définir le *glossaire d'analyse*. Dans chacun des autres processus de la phase d'analyse, des termes pourront être ajoutés. Un même terme peut apparaître dans des glossaires différents en ayant un définition différente. Il est, par exemple, fréquent qu'un concepteur donne à une entité logicielle le nom de l'entité physique dont il est une abstraction.
- Une *fiche de description de l'environnement* qui rend compte de l'appréciation de chacune des caractéristiques de l'environnement et les argumente. Il y figure un inventaire des perceptions élémentaires et des opérations supportables par l'environnement (et qui peuvent être intéressantes pour l'application) qui permettra de réfléchir aux primitives d'actions/perceptions, aux actions plus évoluées et éventuels schémas d'interaction que l'on peut construire à partir des primitives disponibles.
- Une *fiche d'identification des entités actives et passives* sera créée à cet effet. On y trouvera la classification des entités ainsi que les hypothèses et l'argumentation qui a permis ce choix.

On crée dès lors dans la partie "conception" de la documentation un classeur par entité mais pour chaque entité active devenant agent on créer un classeur : il rassemblera plusieurs documents. Ce classeur porte le nom de l'agent et contient une description de sa raison d'être.

Application au cas d'étude : Phase de situation

L'environnement est :

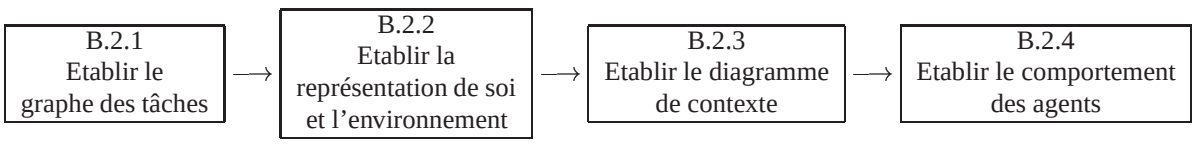
- Accessible. On peut connaître sa position géographique, la position de la balle, de chaque robot de son équipe et de l'équipe adverse. On connaît les dimensions du terrain et le camp de chacune des équipes est communiqué en début de chaque partie. Le temps est mesurable et si l'on peut mémoriser les positions de chacun à des dates différentes, on pourra estimer les déplacements, directions des robots et des trajectoires. La trajectoire de la balle obéissant à des lois physiques, on peut aussi la déterminer. Nous faisons abstraction des autres paramètres mesurables non intéressants pour l'application envisagée (température etc.).
- Plutôt non déterministe. Même si les agents coopèrent et qu'il n'y a pas de dysfonctionnement des actions, on ne peut pas savoir les actions faites par les autres agents. Cependant des éléments de l'environnement sont prédictibles comme la trajectoire de la balle. Les actions faisables sur l'environnement sont le déplacement des robots et éventuellement de la balle. De plus, les règles étant connues à l'avance on peut aussi prédire les décisions de l'arbitre.
- Non épisodique car on supposera qu'aucune intervention de l'homme n'est possible. Les évolutions futures ne dépendent donc par des actions réalisées par les robots.
- Dynamique comme tout environnement réel car le temps ne peut pas être arrêté et ce quels que soient les agents. En effet, le temps n'est pas commandable, il n'est pas arrêtable.
- Continu bien que les actions faisables soient finies.

Les entités actives sont les robots-joueurs. La balle peut-être assimilée à une entité passive qui répond à une action (le tir) par un déplacement soumis aux lois de la physique. Elle fait partie de l'environnement.

Vu le comportement autonome des robots et de leurs besoins en coopération, ils seront des agents.

5.3.2 Phase Individuelle (B.2)

Il est nécessaire avant de créer "l'intelligence" de chaque agent de commencer par une phase mono-agent de construction des comportements : l'agent sera vue comme une entité "autiste". Il est donc nécessaire de s'intéresser aux aspects externes et internes des agents.



L'aspect interne consiste à définir ce qui est propre à l'agent, c'est-à-dire ce qu'il sait faire (primitives d'actions ou services, B.2.1) et ce qu'il connaît (sa représentation des A - E, B.2.2). Dans la plupart des

cas, les actions travaillent à partir des données disponibles dans la représentation de l'environnement détenue par l'agent. Il s'agit donc ensuite de spécifier cette représentation à partir des besoins exprimés lors de la spécification des actions. Afin de garantir la réalité de ces données, il est alors nécessaire de définir les capacités de perception utiles.

L'aspect externe concerne la définition des médias mettant en rapport l'agent avec le monde extérieur, c'est-à-dire ce qu'il peut percevoir (représentation du monde, B.2.2) et comment il le perçoit (diagramme de contexte, B.2.3). L'activité B.2.3 va mettre en exergue, pour chacun des agents, les besoins en capteurs, en effecteurs. Il faudra donc produire un tableau des perceptions/actions qui permettra d'identifier la nature de ces perceptions, les capteurs nécessaires pour accomplir cette perception, et les effecteurs afin de, éventuellement, modifier l'état de ce qui est perçu.

Dans A.3, nous avons établi une vue fonctionnelle du système : il nous faut désormais voir le même problème d'un point de vue plus matériel et centré sur l'agent. Les diagrammes de contexte sont un cas particulier de diagrammes de flots de données qui définissent les interfaces physiques d'un système. Ils définissent les composants de l'environnement d'un système tels qu'ils sont perçus du point de vue du système, en les identifiant comme une série de terminaisons rattachées aux interfaces du système.

Nous allons ainsi, grâce à cette activité, définir le contexte de l'agent physique c'est-à-dire caractériser l'environnement physique dans lequel évolue l'agent. Il est nécessaire de s'intéresser entre autres aux interactions possibles même non désirées entre le système et son environnement (actions et perceptions).

Etablir le graphe des tâches (B.2.1). L'instanciation du système de raisonnement commence, comme nous l'avons vu, par une phase monoagent de construction de plans d'action. On va dans cette activité s'intéresser à la définition des missions confiées aux agents et des actions qui leurs sont nécessaires. On ordonnance ensuite ces actions en plans en ne tenant pas compte des possibles influences externes liées au propre travail des autres agents. Un plan exprime le déroulement normal du travail que doit effectuer un agent pour un but donné, il est constitué d'une séquence d'actions. Les actions peuvent modifier l'environnement ou acquérir des informations. Il est possible de définir des actions que notre agent n'est pas capable d'effectuer et qu'il déléguera au moment de l'exécution [Sichman et al., 1994].

A ce niveau, nous avons identifié quatre catégories d'actions :

1. Les *actions primitives* qui sont les tâches non physiquement décomposables.
2. les *actions primitives paramétrées* qui sont des actions créées à partir d'une action primitive que l'on paramètre.
3. Les *actions composées* qui sont des agencements de primitives.
4. Les *actions situées* qui nécessitent une partie de la représentation du monde pour instancier les actions composées ou les primitives.

Le qualificatif "physiquement" est utilisé dans la définition des primitives pour écarter certaines ambiguïtés. Si on considère par exemple le cas d'un moteur qui permet faire de avancer un mobile selon deux vitesses, $avancer\_lentement$ et $avancer\_vite$ ne seront pas des actions primitives mais des actions primitives paramétrées. D'un point de vue logique cela peut paraître étonnant mais c'est la nature physique de l'application qui nous oblige à prendre ce point de vue. En effet, sans la primitive $avancer$ on ne peut pas créer $avancer\_lentement$ et $avancer\_vite$. La réciproque n'a aucun sens.

Etablir la représentation de soi et de l'environnement (B.2.2). On commencera par s'intéresser à la représentation qu'a l'agent de lui même (identifiant, niveau d'énergie etc.). Ces données de l'agent sont donc à identifier.

Les actions travaillent, dans la plupart des cas, à partir des données disponibles dans la représentation de l'environnement détenue par l'agent. Il s'agit donc de spécifier cette représentation à partir des

besoins exprimés lors de la spécification des actions. Afin de garantir la réalité de ces données, il est alors nécessaire de définir les capacités de perception utiles. Ces perceptions nécessitées par ces tâches peuvent être définies d'une manière analogue aux actions :

1. Les *perceptions primitives* qui sont les perceptions non décomposables.
2. Les *perceptions composées* qui sont construites à partir d'autres perceptions c'est-à-dire des éléments de la représentation du monde.

Les besoins en données de l'agent, qui ne sont pas satisfaits, devront être clairement définis et mis en avant. Lors de la phase sociale, il sera nécessaire de savoir si les autres agents peuvent répondre à ces besoins. Si ce ne sera pas le cas, il faudra contacter le demandeur et re-itérer sur le processus d'analyse.

Diagramme de Contexte (B.2.3) Cette activité peut avoir des répercussions sur le diagramme de cas d'utilisation. On établit[2] ainsi, sans tenir compte des différents solutions technologiques ou des contraintes de conception, le diagramme de Contexte qui va nous permettre d'identifier le système (ou certaines fonctionnalités principales) par rapport aux flux d'entrées et de sorties. Dans l'activité précédente, on a défini les actions et les perceptions de l'agent : à ce niveau on va les instancier en précisant les échanges d'informations entre les différents organes physiques qui exécuteront les actions/perceptions primitives. Aucun choix technologique n'est fait à ce niveau (ce sera fait durant la phase de conception). Le diagramme de contexte peut éventuellement être accompagné d'un diagramme de Flots de Données qui est un sous-processus du diagramme de contexte et qui va permettre d'analyser chaque élément du diagramme de contexte et plus particulièrement les terminateurs et les flots de données. Il sera possible aussi de définir le diagramme de Flots de Contrôle hiérarchisé qui résume le diagramme de contexte et le diagramme de flots de données en intégrant des nouvelles sorties et entrées ainsi que les fiches de spécification des différents processus de transformation ou de contrôle.

Etablir le comportement des agents (B.2.4).

A ce niveau, on a défini l'ensemble des actions nécessaires à la réalisation des missions confiées à l'agent par l'analyse du problème. On ordonnance ensuite ces actions en comportements ne tenant pas compte des possibles influences externes liées au propre travail des autres agents. Un comportement exprime le déroulement normal du travail que doit effectuer un agent pour un but donné, il est constitué d'une séquence d'actions.

Les actions peuvent modifier l'environnement ou acquérir des informations de celui-ci dans un processus de fonctionnement ordinaire, échanger des informations ou de tâches avec d'autres agents.

Documentation liée à ce processus B.2 . Dans cette phase, dans le classeur de chaque "type" d'agents. On créé une partie *phase individuelle* dans laquelle se trouve :

- une *fiche de contexte* dans laquelle va apparaître le diagramme de contexte de l'agent (en accord avec les actions/perception identifiées précédemment). Pour cela on s'appuie essentiellement sur la fiche description de l'environnement et sur les fiches liées aux cas d'utilisation. Aucun choix technologique n'étant fait à ce niveau : le diagramme sera à ce stade "incomplet".
- Une *fiche de schéma d'actions* et une *fiche de perception* est créée pour mettre en respectivement en les dépendances entre les actions (graphe des tâches) et les perceptions. On ne s'intéresse pas aux dépendances perceptions/actions car ce sont des comportements (réactif dans ce cas). On s'appuie essentiellement sur la fiche de contexte pour créer cette fiche.

- Une *fiche de comportement individuelle* exprime le comportement de l'agent (aucun formalisme n'est imposé pour l'analyse). Plusieurs fiches pourront être instanciées si plusieurs comportements

2. Cette activité est fortement inspirée de la méthode SART.

co-habitent. On s'appuie pour construire le comportement des agents sur les fiches liées aux acteurs et au cas d'utilisation. Il est important de consulter les fiches des modes de marches et d'arrêt car certains modes peuvent affecter des comportements individuels.

– Tout terme spécifique (apparaissant dans les diagrammes, les arbres etc.) devra apparaître dans le *glossaire d'analyse de l'agent*. Il figure dans le classeur mais n'est associé à aucune partie.

Application au cas d'étude : Phase individuelle *Rappel : phase mono-agent*

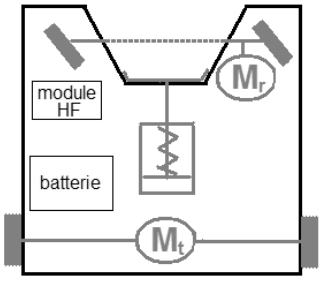


FIG. 5.9 – *Cas d'étude - Synoptique du robot footballeur*

Les actions faisables par l'agent robot sont données en figure 5.10.

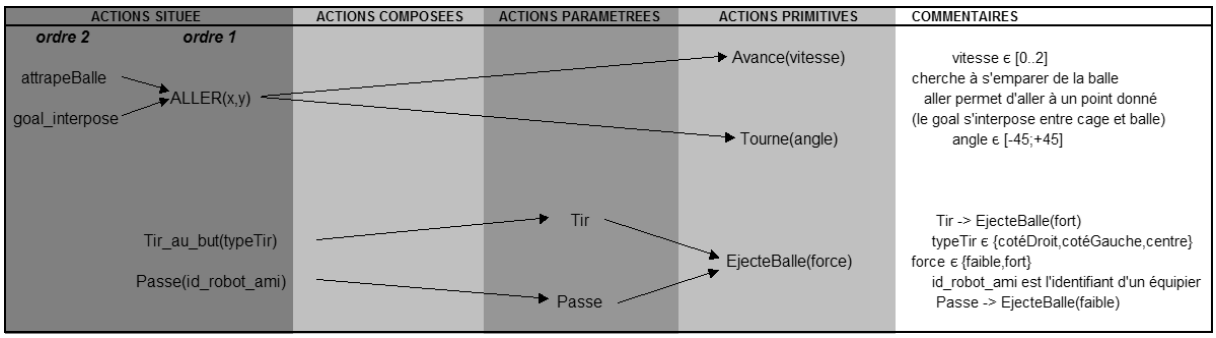


FIG. 5.10 – *Cas d'étude - Définition des actions faisables*

L'agent robot a un identifiant unique. Il peut percevoir sa position géographique et la position de la balle : cela se fait via des envois périodiques de messages (toutes les 0.5s) par voie hertzienne. Chaque agent doit donc avoir un *récepteur HF*.
La position des cages et des lignes de démarcation du terrain doivent donc être codées en dur dans les données de l'agent (voir figure 5.11).
L'agent robot peut se déplacer : nous prévoyons une vitesse lente et rapide (moteur d'avance). La direction peut-être modifiée en tournant sur soi-même *(*moteur rotation). Pour cela on précise l'angle de rotation. Les agents seront donc équipés de deux moteurs.
L'agent peut agir sur la balle en tirant (*piston éjecteur*). Nous estimons qu'il y aura une vitesse pour la passe et une pour le tir.
Selon les stratégies à mettre en oeuvre par la suite, il est possible qu'il soit nécessaire de mesurer le

temps : il faudrait donc doter les agents d'une *horloge*.
Un *bouton* permet de mettre en route et arrêter l'agent
Pour le cas d'étude ou les agents peuvent communiquer entre eux, les agents ont donc aussi la possibilité d'envoyer des messages (*émetteur HF*).

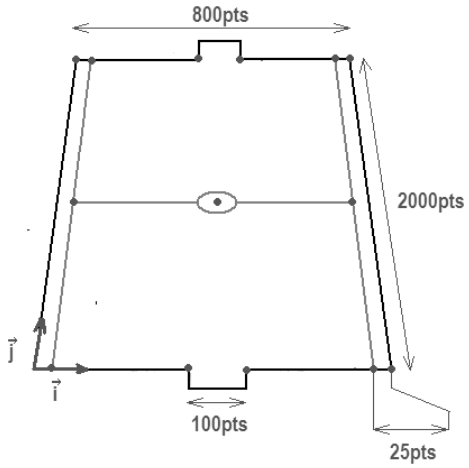


FIG. 5.11 – *Cas d'étude - Représentation du terrain*

La figure 5.12 fait apparaître le diagramme de contexte. Ce diagramme permet d'identifier les signaux de commandes et de données nécessaires à la partie décisionnelle de l'agent. Aucun choix technologique n'est à effectuer à ce niveau de l'analyse ; c'est donc dans une prochaine activité que l'on s'intéressera à la nature de ces signaux.

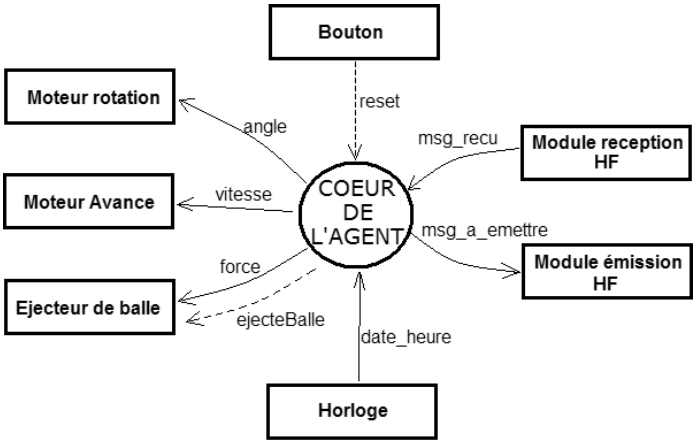


FIG. 5.12 – *Cas d'étude - Diagramme de contexte de l'agent robot footballeur*

Notre direction est donnée par l'angle d'inclinaison des roues : on la représentera par un vecteur unitaire.

Comportement

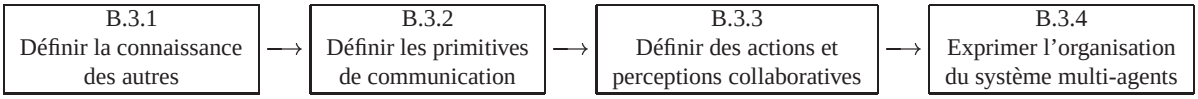
- Comportement d'un agent attaquant
  - Si la balle est à l'adversaire : on procède à un marquage individuel (on s'interpose entre la cage et le robot ennemi).
  - Si la balle est à un ami on se replie pour défendre (on minimalise le nombre de passes pour éviter la perte de balle par interception ou mauvaise direction du destinataire (pour que la passe soit bonne, il faut que la bouche du robot soit orientée vers le passeur))
  - Si la balle n'est à personne, si on est le plus proche de l'équipe on s'en rapproche.
- Comportement d'un agent gardien de but
  - On s'arrange toujours pour être entre la balle et les cages, sur une éventuelle trajectoire de tir.

5.3.3 Phase Sociale (B.3)

Les modes d'échanges sont formalisés de façon externe et commune aux agents à l' aide de protocoles d'interaction (qui peuvent éventuellement mettre en oeuvre des actes de langages). La gestion des conflits est optimisée par la prise en compte des relations entre les agents, c'est à dire la construction d'une organisation. Ces relations peuvent être construites par une analyse des dépendances entre agent en terme de tâches, de position sociale ou d'influence. L'expression d'une organisation doit être externe. Les agents ont la capacité d'exploiter cette connaissance organisationnelle. La modélisation de l'organisation se fait naturellement avec des relations de subordination qui expriment la priorité d'un d'agent sur un autre.

Dans le cas de délégation de tâches, on précise à quels autres agents ces actions peuvent être déléguées et donc de construire une représentation des autres et de leurs compétences.

A ce niveau du cycle de vie, nous allons nous intéresser aux actions qui peuvent initialiser des interactions avec d'autres agents et répondre à tous les besoins en interaction. Nous pouvons distinguer différents types de granularité dans les protocoles d'interactions. Cette granularité va donner naissance aux différentes activités de cette étape.



Définir la connaissance des autres (B.3.1). Dans un premier temps il est nécessaire de s'intéresser à la perception des autres (sous-entendu les autres agents). Ainsi pour chaque agent on indique ce que l'agent peut savoir à tout instant et comment il peut le savoir. Dans cette partie, on n'aborde pas la définition précise des protocoles d'interaction du type protocoles d'introduction : on se contente d'exprimer des conditions sur les données perceptibles.

Définir les primitives de communication (B.3.2). Il nous faut dès lors caractériser les primitives de communication. Cette caractérisation se fait selon quatre points et en respect du cahier des charges :

- Les *médias de communication* qu'offre l'environnement physique. Un médium est le support de transmission permettant le passage des informations d'un équipement à un autre (fibre optique, cuivre, air etc.).

- L'*adressage* qui doit être mis en place. On détermine si les agents doivent pouvoir avoir des conversations bipartites ou multipartites, diffuser à des groupes (multicast), inonder les autres agents (broadcast).
- La *persistance* pour savoir si les messages doivent être disponibles pour une certaines durée (utilisation éventuelle de blackboard ou autre).
- La *localité* des communications : on doit identifier s'il est possible ou non que les envois de messages nécessitent de s'appuyer sur son entourage.

Si le cahier des charges contient des contraintes techniques à ce sujet, il est important de les prendre en compte. Cependant, en aucun cas le choix d'un médium parmi différents types de média possibles n'est effectué à ce niveau : c'est au concepteur de choisir une solution technologique en fonction des contraintes technologiques spécifiées : seuls l'analyse des besoins en communication des agents est approfondie.

Définir des actions et perceptions collaboratives (B.3.3). A ce stade de l'analyse, on s'intéresse à la définition des services évolués intégrant des interactions avec les autres agents et aux éventuels contrats: on définit les protocoles d'interaction. On s'intéresse aussi aux perceptions que l'on peut obtenir grâce à d'autres agents. Les besoins en données qui n'ont pas été satisfaits dans la phase individuelle doivent en effet trouver un écho favorable dans cette activité sinon il sera nécessaire de solliciter à nouveau l'utilisateur.

Exprimer l'organisation du SMA (B.3.4). On s'intéresse ensuite à l'aspect social du système comme dans la décomposition AEIO des systèmes multi-agents traditionnels. On va donc étudier la possibilité de collaborations, la formation de coalition etc. On pourra par exemple associer des schémas d'interactions aux rôles etc. Les groupes peuvent être exprimés à l'aide de diagrammes entité-relation.

Documentation liée à ce processus B.3 . On crée pour chaque classeur d'agent une partie *Phase sociale*. On y adjoint une *fiche de connaissance des autres*, une *fiche des besoins en communication*, une *fiche des actions et des perceptions collaboratives*, une *fiche d'aspect organisationnel* et une *fiche des comportements collectifs* qui n'impose aucun formalisme pour l'analyse. Il est important de consulter les fiches des modes de marche et d'arrêt car certains modes peuvent affecter des comportements collectifs.

Application au cas d'étude : Aspect social

Connaissance des autres : que sait-on d'autrui?

- position si possibilité de capter les envois de messages du système caméra (communication sans fil - module WIFI),
- direction : estimation possible si mémorisation des positions précédentes,
- intention des amis s'ils le signalent (volonté d'obtenir la balle, d'effectuer un tir etc.)

Primitives de communication

Il est possible de communiquer en utilisant des reconnaissances de trajectoire. Cependant cette solution étant contraignante, peu expressive et pouvant entraîner des erreurs d'interprétation, elle ne sera pas retenue. Les interactions entre les agents sont réalisées par échange de messages. Il est nécessaire de :

- pouvoir communiquer avec toute son équipe (diffuser son intention de tir),

– pouvoir communiquer avec un individu de son équipe (résoudre un conflit de tir, demander à un co-équipier de se déplacer etc.),

Définir des actions collaboratives

– on peut demander la balle quand on a une occasion de tir
– on peut demander à changer la personne que l'on marque
– on peut demander à quelqu'un de changer de position pour attirer un adversaire ailleurs

Organisation du système multi-agents

On a déjà constaté la présence de deux équipes sur le terrain : un agent peut donc déjà faire la distinction entre son équipe et l'équipe adverse. Une EQUIPE (voir fig. 5.13) selon le cahier des charges est composée d'un GARDIEN qui peut entrer dans la zone de réparations et de trois autres joueurs que l'on désignera sous l'appellation ATTAQUANT qui peuvent y accéder. Les stratégies de jeux étant très simple aucun autre besoin organisationnel n'est soulevé.

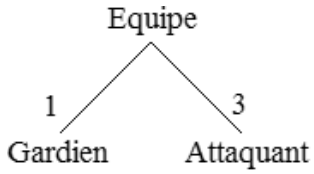


FIG. 5.13 – *Cas d'étude - Structure d'une équipe*

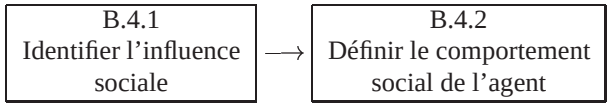
Comportements collectif

– Comportement collectif si l'agent est attaquant
  – Si la balle est à un ami alors cherche une possibilité de tir. Si on l'obtient, on le signale.
  – Si la balle n'est à personne et si on est le plus proche de l'équipe, on s'en rapproche.
  – Si plusieurs joueurs d'une même équipe ont une possibilité de tir alors, on estime la probabilité de succès (qui tient compte des éventuelles passes) et on fait parvenir la balle au joueur le mieux placé(s'il ne possède pas déjà la balle) pour qu'il tire.
  – Un passeur prévient toujours le destinataire qu'il va lui donner la balle.
  – Si quelqu'un demande la balle je regarde si je peux aider le demandeur de la balle à l'obtenir (passeur intermédiaire)
– Comportement collectif si l'agent est gardien de but
  – Le gardien peut participer à des passes.

5.3.4 Phase d'intégration (B.4)

Cette phase permet la socialisation des individus. A ce point, les influences possibles sur les comportements purement individuels doivent être analysées. Ces influences sont intégrées dans les agents par l'intermédiaire des capacités d'évaluation de la communication et de la perception détaillées dans le modèle de l'agent. Jusqu'ici la décomposition a occulté la notion de contrôle de l'agent, c'est-à-dire

comment il va gérer son attention, sa décision, et enchaîner ses actions. Cet aspect que l'on peut qualifier de mixte se greffe sur les deux précédents (social et individuel). C'est par l'intégration des influences sociales dans les agents que l'on va donner une dynamique au SMA.



Identifier l'influence sociale (B.4.1). Le but de cette activité est d'identifier les influences sociales qui peuvent affecter le comportement purement individuel de l'agent. Ainsi, on doit déterminer les possibles situations d'indépendance, de collaboration (simple ou ordonnée), d'encombrement, de compétition (individuelle pure ou collective), les situations de conflits (individuels ou collectif) pour des ressources. Ces différentes situations d'interactions sont précisés en §2.2.3.3.

Définir le comportement social de l'agent (B.4.2). Il va maintenant falloir adapter le comportement des agents (jusqu'à lors il était "autiste"). On va donc prendre en compte les perturbations identifiées précédemment et dues aux autres agents.

Cette phase a pour objectif de réaliser l'intégration dans les agents des aspects décrits dans les phases précédentes. Elle impose en cela de construire le contrôle de l'agent. Cette phase a pour objectif de réaliser l'intégration dans les agents des aspects décrits dans les phases précédentes. Elle impose en cela de construire le contrôle de l'agent. Il s'agit donc de choisir une architecture et de l'instancier dans le contexte de l'application.

Nous devons décrire les connaissances et des tâches de l'agent. Les tâches de l'agent peuvent être de simples primitives ou des services évolués.

A ce niveau de la méthode nous nous intéressons au premier niveau de description de l'agent, c'est-à-dire son modèle. Les traditionnels second et troisième niveau de description d'un agent, l'architecture et l'implémentation, ne concerneront respectivement que les étapes de *Conception générique* et de *réalisation* du cycle de développement.

Documentation liée à ce processus B.4 . On créé pour chaque classeur d'agent une partie *Phase d'intégration*. On y adjoint une *fiche d'identification de l'influence sociale*, une *fiche de comportement sociale*, une *fiche d'adaptation des comportement individuels et sociaux* où pour chaque influence sociale identifiée on apporte une correction du comportement de l'agent.

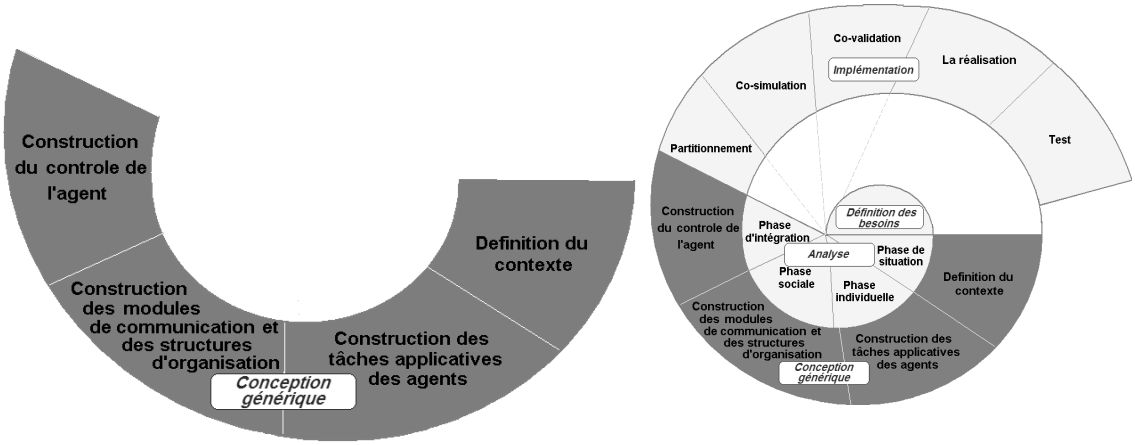
Application au cas d'étude : Phase de d'intégration

Identification de l'influence sociale / *Adapter les comportements individuels*

- Si on veut se rendre à un point, quelqu'un (ami ou non) peut être sur notre trajectoire,
- Deux joueurs d'une même équipe peuvent vouloir effectuer un tir en même temps *(on résoud le conflit en calculant une estimation de réussite de l'action, en tenant compte des passes qui abaisseront cette estimation)*,
- Si on cherche une possibilité de tir un ami peut diminuer le nombre des possibilités si, par exemple, il est sur une trajectoire de tir *(on lui demande de bouger))*,

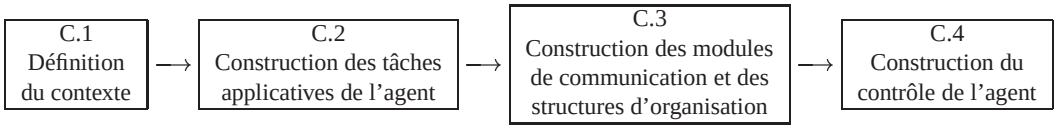
On obtient ici une première description souvent suffisante, grâce à la phase de validation on peut faire le point, décider si l'on itère ou si l'on passe) à la phase "conception générique".

5.4 L'étape de conception générique (étape C)



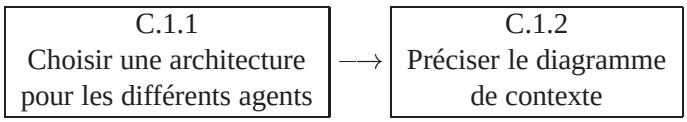
Tout n'est pas modélisable, ni plus encore implémentable avec un même modèle de système multi-agents. Par contre, les mêmes schémas se répètent souvent au travers des différentes applications. Un des objectifs de notre approche est de fournir des modèles et des outils basés sur la réutilisabilité. L'expérience accumulée dans le domaine de la conception d'applications des SMA nous permet d'envisager de réutiliser des modèles génériques développés pour chacune des notions A, E, I, O, en les instanciant pour les applications visées.

L'objectif de ce niveau est d'obtenir un système de composants à partir de la description informelle du système multi-agents. La construction de nos agents repose sur l'utilisation de composants. Les composants sont de très bons outils car ils sont interchangeables et réutilisables. De plus, ils facilitent le travail futur concernant le partitionnement de ce qui deviendra logiciel et matériel.



5.4.1 Définition du contexte (C.1)

Dans cette étape nous allons préciser le contexte qui a été esquissé durant l'étape précédente.



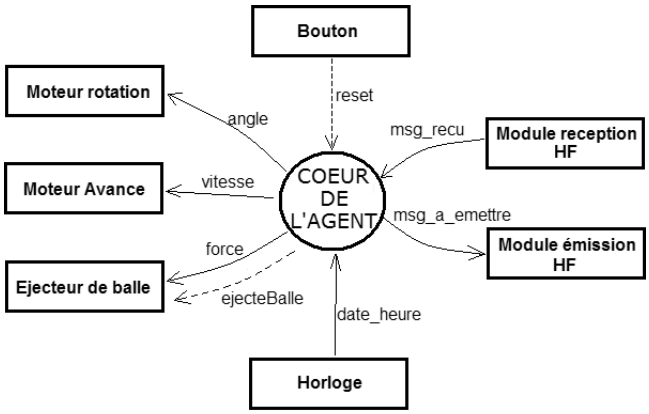
Choisir une architecture pour les différents agents (C.1.1). On fait les choix des architectures pour les agents en accord avec le modèle choisi et les contraintes spécifiées précédemment. Cette architecture comme nous le verrons dans l'étape suivante, servira de patron pour la conception détaillée en composants permettant ainsi de la guider et de la structurer.

Préciser le diagramme de contexte (C.1.2) A ce niveau de l'étude il est temps de procéder aux choix technologiques des actionneurs et des effecteurs de chaque agent. On prendra garde les contraintes à ne pas violer figurant dans la fiche des contraintes technologiques. Les paramètres qui influent sur ce choix sont traditionnellement le coût, la consommation électrique, le rendement, la fiabilité etc. En fonction de ces choix il nous faut caractériser les interfaces qui permettront de numériser les informations (échantillonner, quantifier, coder). On complète ainsi les informations apparaissant sur le diagramme de contexte.

Documentation liée à ce processus C.1 . Au point de départ du travail fait dans cette activité sont les fiches de contextes élaborée dans la phase individuelle. A ce niveau de l'étude on s'appuie sur les à ne pas violer les contraintes figurant dans la fiche des contraintes technologiques. On va créer une partie *Conception* dans la documentation de l'agent. On commence par créer une *fiche de description architecturale* de l'agent. On y précise les raisons qui ont amené à ce choix. On créé une nouvelle *fiche de diagramme de contexte*. On y reprend le contenu du diagramme de contexte mais en précisant la nature physique des informations (il se peut que dans certains cas il soit nécessaire de dégrader le diagramme c'est-à-dire l'adapter à des contraintes particulières de conception).

Application au cas d'étude : Définition du contexte

Le diagramme de contexte donne les spécifications suivantes :



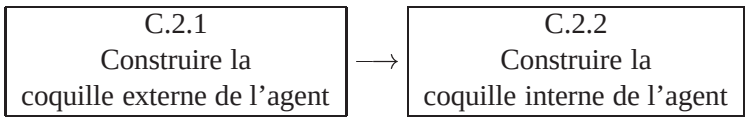
Le concepteur va donc procéder aux différents choix technologiques et spécifier les interfaces avec les éléments dépendant du matériel. Les interfaces choisies, on va spécifier la nature des signaux qui transiteront entre le système que l'on conçoit et ces élément matériels (voir tableau 5.3).

TAB. 5.3 – *Spécification des interfaces*

Information	Spécification
Reset	Actif sur niveau logique haut (1bit).
Moteur rotation	Angle, relatif à la direction actuelle, dans l'intervalle [-180,+180] codé entier signé sur 10 bits.
Moteur Avance	Deux vitesses possibles plus l'arrêt. Entier codé sur 2 bits. (00: arrêt / 01: vitesse lente / 10: vitesse rapide)
Force	Deux niveaux de forces possibles. Niveau codé sur 1 bit. (0: passe / 1: tir)
Ejecte balle	Front montant sur 1bit
Date_heure	Nombre de millisecondes écoulées depuis la mise sous tension (Entier codé sur 32 bits).
Msg_recu	Suite de bits (émetteur 1octet,dest 1octet,taille_données 1octet, données 1-256octets).
Msg_emi	Suite de bits protocole spécifique (émetteur 1octet,dest 1octet,taille_données 1octet, données 1-256octets)

Il est à remarquer que le diagramme de contexte décrit un composant de très haut niveau. Ce dernier contiendra le pattern ainsi que tous les composants que l'on créera ou utilisera.

5.4.2 Construction des tâches applicatives de l'agent (C.2)



Nous allons construire les différentes tâches de l'agent. Cette construction se fera via des composants. Un composant peut être défini comme une unité indépendante de production et de déploiement, qui est combinée à d'autres composants pour former une application (dans notre cas ce sera une application multi-agents). Une approche orientée composants permet de bénéficier de plusieurs avantages :

- Réutilisation des composants par une autre application,
- Mise à jour des composants, sans que l'ensemble de l'application ne soit modifié,
- Les composants peuvent reposer sur des ressources (Base de données, images).

Néanmoins, comme généralement "le langage est l'environnement", il n'est pas suffisant d'inventer de nouveaux langages de programmation ou des mécanismes de composition. Ceux-ci doivent être aidés par des environnements de développement, de sorte que les développeurs puissent appliquer les nouveaux dispositifs.

Construire la coquille externe de l'agent (C.2.1). On construit la coquille de l'agent c'est-à-dire :

– les parties de l'agent qui vont acquérir l'information provenant des interfaces capteurs afin son interface avec le monde extérieur afin d'établir la représentation du monde,

– les parties de l'agent qui vont agir sur le monde extérieur pour changer son état.

Construire la coquille interne de l'agent (C.2.2). On s'attache ensuite à la création des actions évoluées : actions paramétrées, composées et situés ainsi que les actions composées. Il est conseillé de respecter l'ordre dans lesquels ont été définies préalablement les différentes actions et perceptions afin d'éviter de devoir revenir sur un composant déjà créé à cause d'une mauvaise appréciation de l'interface d'un composant utilisé.

Documentation liée à ce processus C.2 . Pour chaque composant créé, on saisit une *fiche de description de composant* qui permet d'identifier sa description externe et sa description interne.

Application au cas d'étude : Construction des tâches applicatives de l'agent

A ce stade, on agence les composants pour construire l'application. L'architecture de l'agent servira de patron pour la décomposition en composants (figure 5.16).

Les composants seront dotés d'une description externe (figure 5.14) et d'une description interne (figure 5.15). La description interne peut être un agencement de composants, un formalisme type réseau de Petri, ou une description textuelle pouvant engendrer du logiciel ou du matériel.

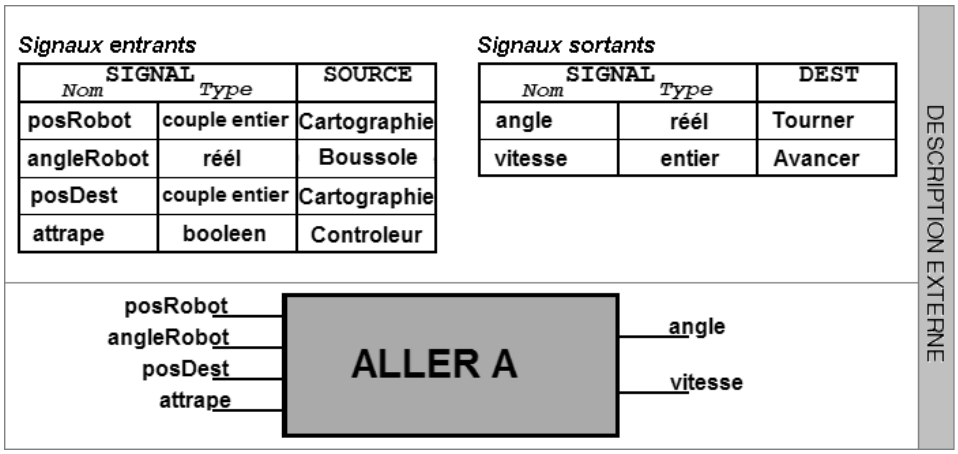


FIG. 5.14 – *Spécification externe des composants*

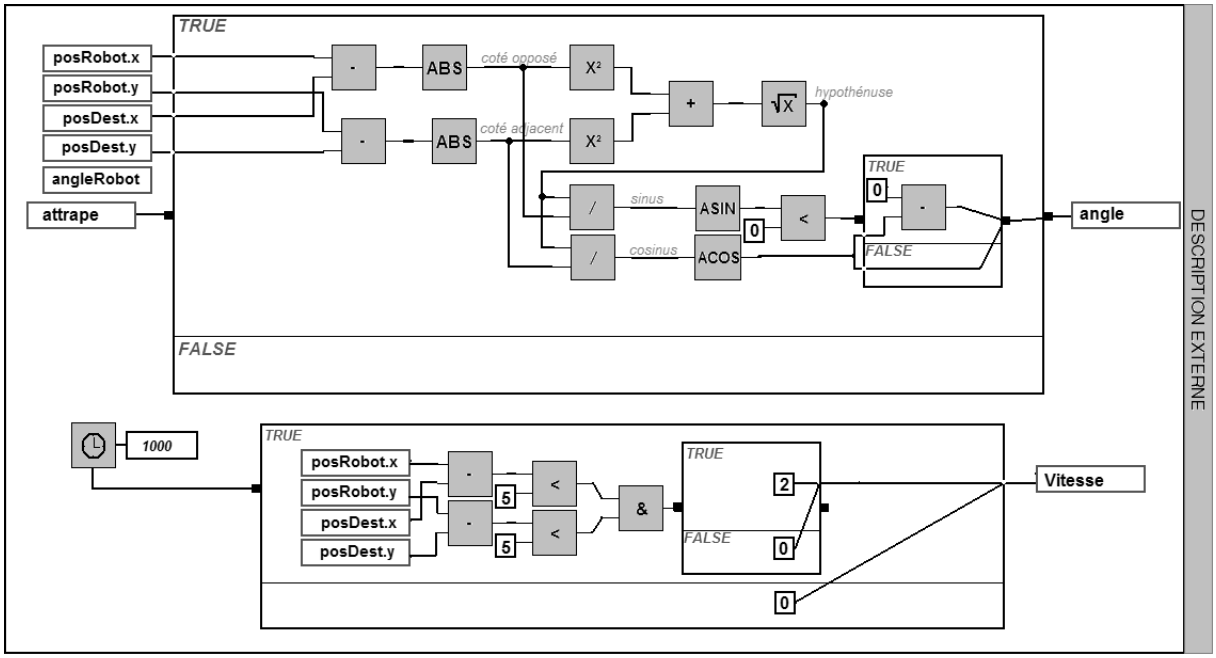


FIG. 5.15 – *Spécification interne des composants*

5.4.3 Construction des modules de communication et des structures d'organisation (C.3)

Ce processus a pour objectif la construction des modules de communication. Cette construction se fait, comme précédemment avec des composants. Une fois cette opération effectuée, on va s'intéresser à la construction des protocoles d'interaction et de la structure organisationnelle toujours avec des composants.

Documentation liée à ce processus C.3 . Similaire à la construction de la documentation du processus C.2 vue en 5.4.2.

Application au cas d'étude : Construction des modules de communication et des structures d'organisation

Cette partie de l'analyse s'intéresse à la création des modules de communication et des structures organisationnelles définies durant l'étape d'analyse avec des composants comme précédemment. La nature des composants manipulés est différente mais la philosophie est la même 5.4.2.

5.4.4 Construction du contrôle de l'agent (C.4)

Cette étape compose avec l'élaboration des comportements des composants/agents. Les parties "évaluation" et "décision" de l'agent seront créées dans cette étape. Quand un message est reçu le composant/agent doit être capable de fournir une réponse appropriée, de choisir les comportements appropriés qui doivent être exécuter. Cette exécution de tâche va changer alors l'état du composant/agent.

C.4.1 Construire le contrôle (partie environnementale) *Dis moi comment tu réagis et je te dirais d'où tu viens?*	$\longrightarrow$	C.4.2 Construire le contrôle social de l'agent *Dis moi comment tu penses et je te dirai de quelle société tu viens?*

Documentation liée à ce processus C.4 . Similaire à la construction de la documentation du processus C.2 vue en 5.4.2.

Application au cas d'étude : Construire le contrôle de l'agent

La figure 5.16 présente l'agencement des composants selon l'architecture eASTRO.

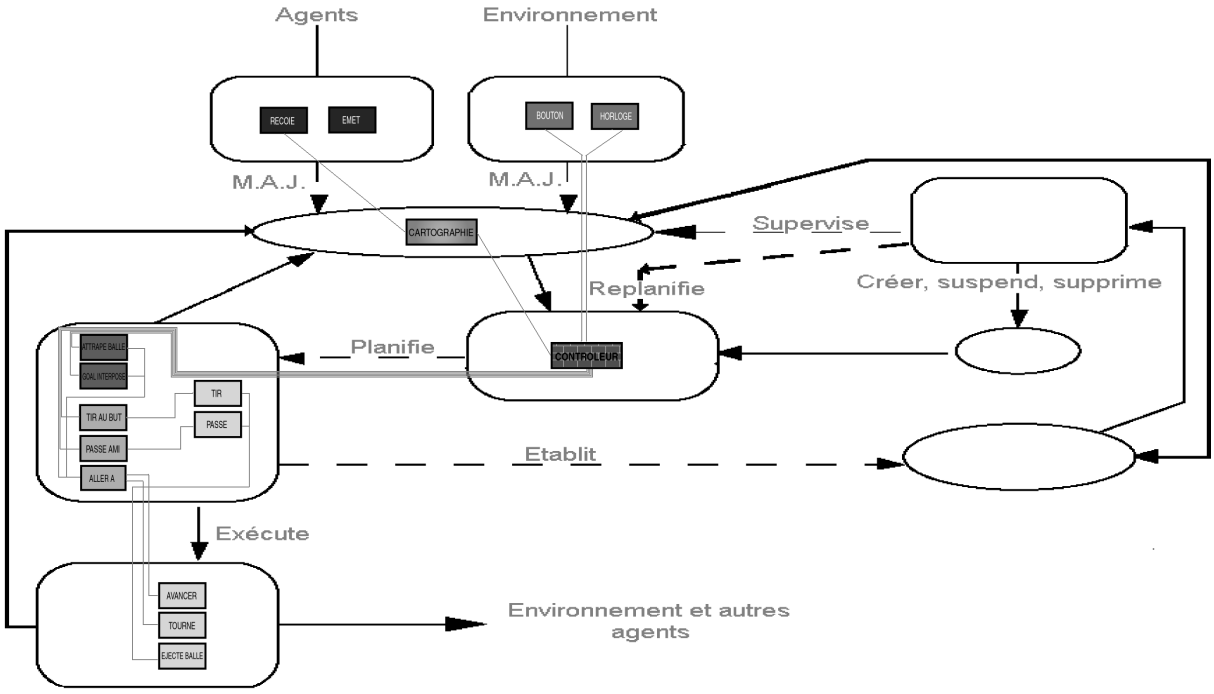


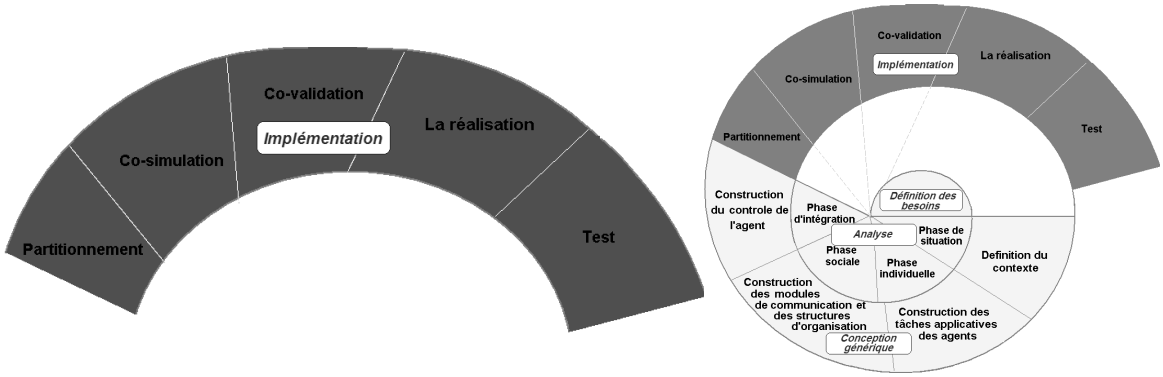
FIG. 5.16 – *Architecture eASTRO comme patron*

Le modèle des agents eASTRO[3] ("embedded Agent" Spécialiste pour le Temps Réel et la cOopération) est basé sur le paradigme perception / évaluation / raisonnement / action. Le centre de l'agent est

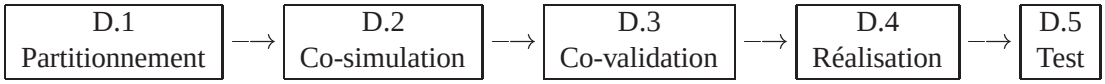
3. aménagement du modèle ASTRO [Occello and Demazeau, 1998]

son modèle du monde. Ce modèle contient les connaissances de l'agent sur l'environnement, les autres agents et son propre état. L'état de l'agent contient en particulier les plans qui sont en cours d'exécution ou sur lesquels travaille le processus de raisonnement de l'agent. Ce modèle est rafraîchi par un processus d'interprétation des données perçues. Comme il évolue dans un monde dynamique, l'agent est muni de dispositifs de perception qui sont gérés par les modules de perception. Pour assurer la réactivité de l'agent, des évaluateurs examinent en permanence le modèle du monde. Les modules de contrôle de l'agent détectent les situations pour lesquelles l'agent doit réagir, les évaluent, et décident des réactions, qui peuvent consister en l'introduction, la suspension ou la suppression de buts de façon à changer le contexte du processus de planification et de décision. La supervision continue de la situation de l'agent assure la réaction de l'agent à l'occurrence d'événements imprévus à tout instant. Le même mécanisme (modules de communication) prend en compte l'interaction avec les autres agents en utilisant si besoin des protocoles d'interaction. Quand un but est créé (ou modifié) un plan est recherché pour satisfaire ce but avant sa date limite, cette tâche est réalisée par les modules de raisonnement. Les plans construits sont ensuite ordonnancés par les modules de décision. L'ordonnanceur choisit la meilleure façon de réaliser le plan. Le planificateur et l'ordonnanceur ont la possibilité d'introduire des actions internes comme la replanification, ou l'établissement de conditions de veille pour garantir une meilleure d'adaptation au contexte. Ces gardes fournissent des informations sur la situation courante durant la phase d'exécution des actions, elles sont stockées dans l'état interne. Ce mécanisme assure l'adaptation de l'agent à la rapidité de l'évolution de son environnement ; il est indispensable si l'agent poursuit plusieurs buts simultanément.

5.5 L'étape d'implantation (étape D)



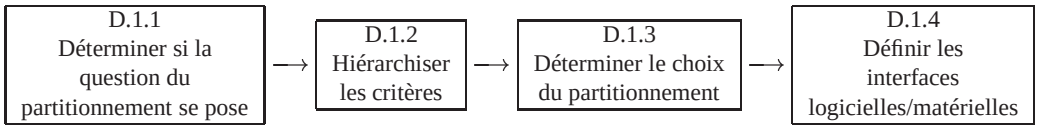
Cette étape constitue à la fois la phase terminale de conception et la phase de réalisation.



5.5.1 Le partitionnement (D.1)

La principale utilisation du co-design figure dans le partitionnement logiciel/matériel qui sera fait des différents composants se rapportant aux différents axes de la décomposition AEIO.

Il devra être fait manuellement (dans notre cas) en accord avec les spécifications technologiques. Il inclura, à terme, la co-simulation pour la co-synthèse. Cette étape produit les spécifications du logiciel et du matériel.



Il nous est indispensable de nous interroger sur les critères qui vont orienter nos décisions vers leur implémentation logicielle ou matérielle. Nous pensons qu'il y a deux niveaux à prendre en compte : tout d'abord on va commencer par écarter les composants pour lesquelles la question du partitionnement ne se pose pas (D.1.1). Pour chacun des composants pour lesquels se pose la question du partitionnement, on va hiérarchiser les critères que l'on souhaite satisfaire (D.1.2). On en déduira le choix du partitionnement (D.1.3) et on s'interrogera quant aux interfaces logicielle/matérielle à implémenter[4] (D.1.4).

Déterminer si la question du partitionnement se pose (D.1.1). Dans ce premier niveau de sélection du partitionnement on va déterminer les parties de l'agent pour lesquelles la question du partitionnement ne se pose pas. En effet certains éléments, par essence même, devront être matériels. Comme on peut le voir sur la figure 5.17, c'est le cas des périphériques d'entrées/sorties comme par exemple les capteurs et les actionneurs.

Hiérarchiser les critères (D.1.2). Ce second niveau de sélection du partitionnement concerne les éléments pour lesquels il y a réellement plusieurs choix d'implémentation (fig. 5.17, zones partagées par le co-design et le logiciel ou le matériel). Pour ceux-là différents paramètres vont entrer en jeu : les critères de partitionnement peuvent être plus ou moins subjectifs. Nous présentons dans le tableau 5.4 ceux que nous avons jugés pertinents pour les agents à partir de différents travaux effectués dans le co-design [De Micheli, 1995, Ismail, 1996, Berrebi, 1997, Koudil, 2002, Kumar, 2002] et de notre propre expérience.

Déterminer le choix du partitionnement (D.1.3). Le tableau 5.4 définit plusieurs critères et indique la partitionnement qui sera incité. Il est possible d'avoir des indications contradictoires : c'est la raison pour laquelle les critères ont été hiérarchisés. C'est le concepteur qui devra déterminer la pertinence d'une ou l'autre solution. Un autre critère non mentionné précédemment est à prendre en compte : ce qui a déjà été réalisé par l'entreprise. En effet, certains composants ont éventuellement pu être construits d'un point de vue purement matériel ou purement logiciel. Le concepteur pourra donc être incité à choisir cette solution. Un autre critère qui n'entre pas dans le niveau précédent provient de la nécessité de créer des interfaces entre le logiciel et le matériel. Selon ce que propose l'outil, il peut être compliqué d'interfacer

4. La co-synthèse n'est pas effective dans notre outils.

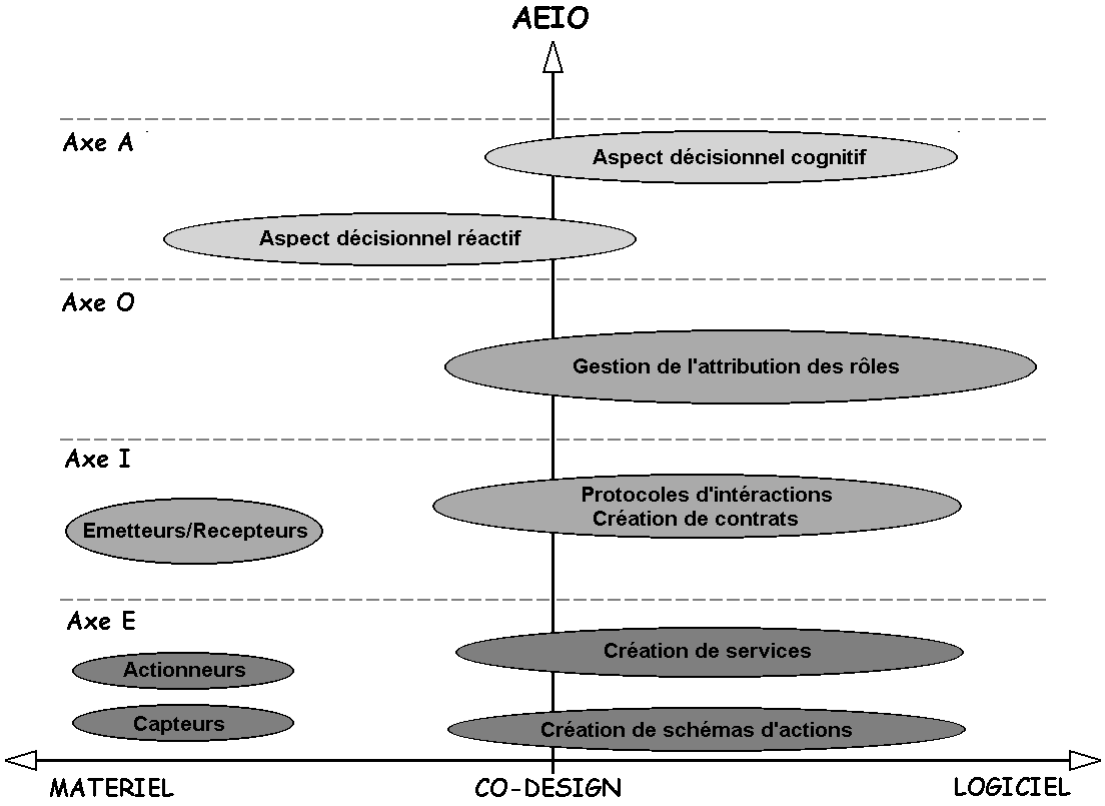


FIG. 5.17 – *Aperçu du partitionnement prévu pour les axes de la méthode AEIO*

logiciel et matériel. Ainsi, on évitera d'avoir une succession de composants logiciels, puis matériels, puis logiciels etc.

Définir les interfaces logicielles/matérielles (D.1.4). Lorsque l'on fait collaborer du logiciel et du matériel il est nécessaire de définir des interfaces. Généralement on établit une norme pour tout composant et on fournit un bus.

Documentation liée à ce processus D.1 . Une partie partitionnement est créée. Elle regroupe une *fiche de critère de partitionnement* qui indique la hiérarchisation des critères. De plus une *fiche de partitionnement* précise pour chaque composant la déclinaison choisie. Il peut exister plusieurs implémentations différentes logicielles ou matérielles d'un même composant : la référence de l'implémentation choisie est précisée et ce choix doit être motivé. Si des contraintes influant sur le partitionnement sont spécifiées dans les fiches de contraintes techniques et technologiques, elles ne devront pas être violées.

5.5.2 Les activités de co-simulation (D.2) et de co-validation (D.3)

Ces deux activités sont centrés sur les agents et non sur le système.

La co-simulation est une simulation simultanée des parties logicielles et des parties matérielles sans oublier leurs interactions. Elle permet d'analyse les propriétés dynamiques du système comme les temps d'exécution. Elle permet de co-vérifier le système c'est-à-dire de garantir que :

– Les performances et fonctionnalités du système à concevoir sont conforment aux co-spécifications.

TAB. 5.4 – *Les critères intervenants dans le choix du partitionnement*

Critère	Description
Coût	Critère omniprésent pendant tout le processus de conception du système. Sur de très petites séries c'est le coût du développement du système logiciel et du matériel que l'on va chercher à diminuer autant que possible tandis que dans le cas de grandes séries, c'est la diminution des coûts de fabrication qui revêtira une grande importance.
Performance	Son importance varie selon les problèmes traités. Les applications temps-réel et pour lesquelles la robustesse est fonction du temps d'occupation processeur sont celles pour lesquelles ce critère est capital. C'est alors un partitionnement matériel qui sera à privilégier autant que possible.
Flexibilité	Ce critère qui joue en faveur du logiciel. En effet, les effets de bords liés aux modifications du logiciel ont généralement un impact moins important sur le système global que dans le cas du matériel. Cependant, la flexibilité des EPLD et autre FPGA s'améliore grandement et ils sont même re-programmables in-situ, c'est-à-dire que l'on peut modifier leur programme sans les extraire de la carte électronique sur laquelle ils sont fixés.
Tolérance aux pannes internes	De part leur nature les systèmes logiciels sont moins tolérants aux pannes. En effet, les microcontrôleurs, sur lesquels ils reposent, utilisent des mémoires, des structures de piles sujettes au débordement etc. Ce critère qui jouera en faveur d'un partitionnement matériel.
Ergonomie	Il regroupe toutes les caractéristiques physiques du système (le poids, le volume, la consommation énergétique, le dégagement thermique etc.). Selon l'application ce critère peut être hautement critique (cas des applications relevant des métiers de l'aéronautique). C'est au concepteur d'apprécier correctement ce critère.
Complexité algorithmique	Plus une tâche est complexe, plus il y aura de chance pour que l'on s'oriente vers un partitionnement logiciel. La synthèse matérielle des tâches hautement cognitives par exemple est beaucoup trop compliquée.

– Que des déviations du fonctionnement n'aient pas été introduits par les différentes étapes du processus de co-design.

La co-validation se fait à l'aide d'outils formels. Elle seule permet de vérifier que le système réalisé respecte toutes les contraintes fixées : elle est cependant très difficile à mettre en oeuvre.

Documentation liée aux processus D.2 et D.3 . Dans cette partie de la documentation doivent figurés les protocoles mis en jeu et les résultats commentés des différentes simulations et validations (*fiches de co-simulation* et *fiches de co-validation*).

5.5.3 La réalisation (D.4)

Elle sera réalisée à terme grâce à un outil associé en cours de développement. Chaque composant est spécifié grâce à une notation graphique commune pour la partie matérielle et la partie logicielle. Pour chacun de ces composants le concepteur choisit s'il désire une implémentation matérielle ou logicielle.

L'outil pourra prendre en charge la génération automatique du code pour les composants dont il a été choisi une implémentation logicielle. Le code sera écrit dans un langage portable comme le langage Java ou le langage C (facilement embarquable). Pour ce qui est des composants matériels une spécification en VHDL[5] sera générée. Après cela, la compilation des différents codes et la synthèse matérielle des différentes spécifications en VHDL seront réalisées comme l'illustre la figure 5.18. Cette figure met en évidence le traitement d'un composant matériel et d'un composant logiciel.

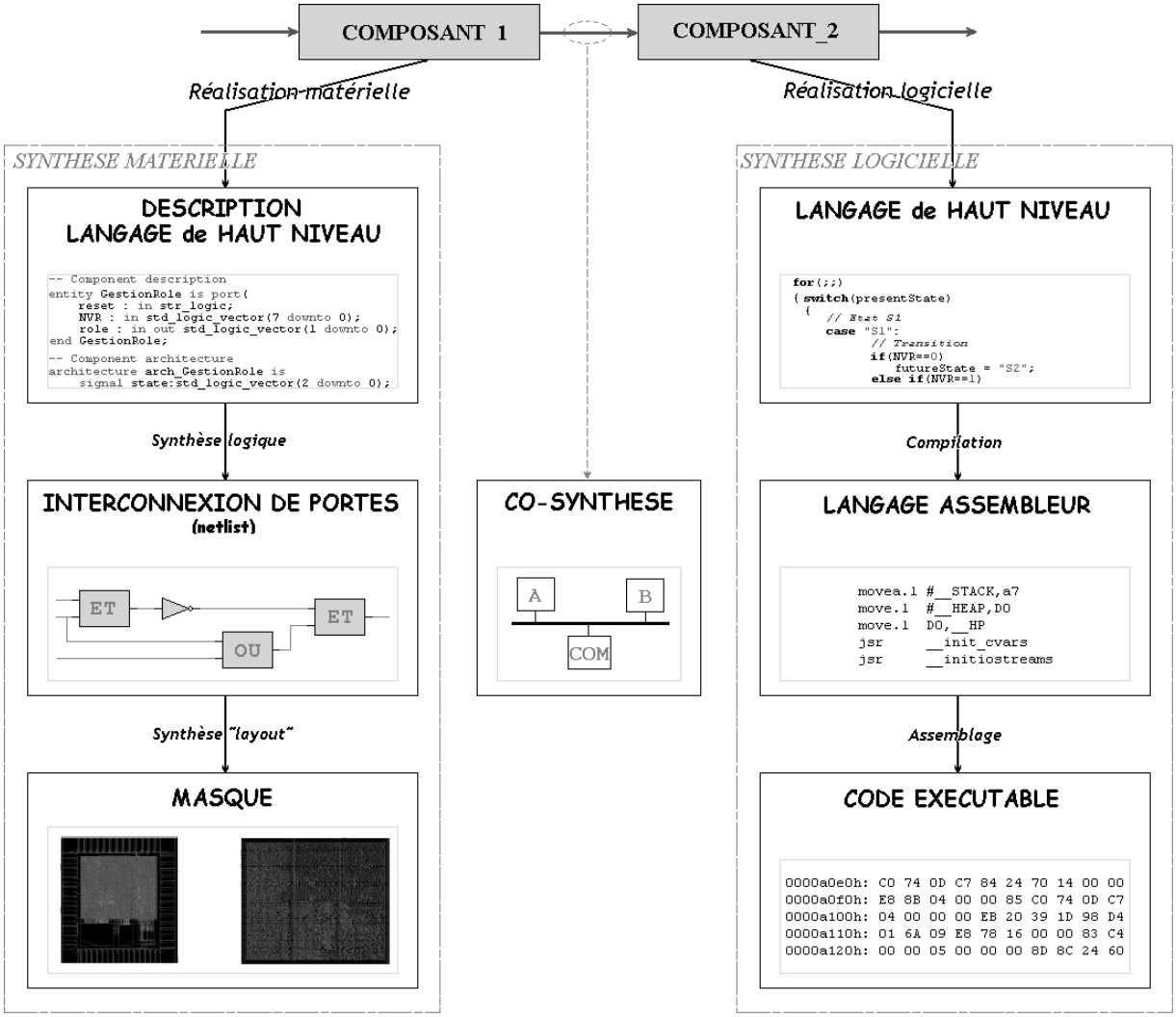


FIG. 5.18 – *Synthèse logicielle et matérielle des composants*

Le composant matériel est décrit avec un langage de haut niveau prévu à cet effet (ici le VHDL). On y voit sa description externe (Component entity) et le début de sa description interne (Component architecture). Par "synthèse logique", nous entendons synthèse comportementale qui traduit la description matérielle en une construction fonctionnelle du circuit (logique combinatoire, machine d'états, registres etc.) suivie de la synthèse RTL qui prend en compte le déroulement temporel des traitements. La synthèse "layout" permet la création de masques. Le résultat de ces traitements est un composant matériel.

5. Le lecteur ne connaissant pas VHDL est invité à consulter l'annexe B

Le composant logiciel voit son fonctionnement codé via un langage de haut niveau (ici le langage C). La compilation permet la traduction du programme en instructions machines. Ces instructions primitives dépendent généralement du processeur cible. L'édition de liens permet d'ajouter des bibliothèques d'exécution, contenant principalement la gestion mémoire, et de faire la jonction avec le système d'exploitation pour les primitives. Le résultat de ces traitements est un code binaire qui sera placé en mémoire et exécuté par un processeur.

Documentation liée à ce processus D.4 . La documentation consiste en :
- *fiches de code sources* qui est une copie textuelle des codes générés,
- *fiches de description architecturale matérielle* qui rend compte des spécifications matérielles générées,
- *fiches de co-synthèse* qui rend compte des interfaces entre les différentes parties logicielles, les différentes parties matérielles et les parties logicielles/matérielles.

5.5.4 Le test (D.5)

Ce processus est centré sur le système et non sur les agents. Dans ce processus on va s'intéresser à tester le produit finalisé logiciel/matériel. On va s'attacher à vérifier la conformité entre la spécification initiale et le produit réalisé. On portera une attention particulière aux critères soulevés dans l'activité de hiérarchisation des critère.

Documentation liée à ce processus D.5 . Une *Fiche de conformité* est saisie. Elle servira de support aux personnes chargées de mesurer la satisfaction du demandeur.

5.6 Conclusion : discussion sur la méthode

La discussion que nous proposons est articulée en deux parties. La première concerne le cycle de vie (les phases et leur ordonnancement) et la seconde les modèles et les notations.

5.6.1 Le Cycle de vie

5.6.1.1 Leur ordonnancement

Comme signalé dans [DeLoach et al., 2001], la plupart des méthodes multi-agents ne distinguent qu'une phase d'analyse et de conception. Très rares sont les méthodes qui couvrent d'autres phases. Il s'agit probablement d'une conséquence du fait que les méthodes ont souvent été développées pour des modèles/architectures cibles précis. Par exemple, la méthode AAII [Kinny et al., 1996] et la méthode Prometheus [Padgham and Winikoff, 2002] ont été développées pour l'architecture BDI tandis que la méthode AALAADIN [Ferber and Gutknecht, 1998] a été développé pour le modèle AGR. Depuis très récemment, les méthodes semblent commencer à se détacher des architectures pour se consacrer à la résolution de problèmes indépendamment de celles-ci.

Nous n'avons identifiés à ce jour, qu'une seule méthode qui n'utilise aucun cycle de vie traditionnel mais la notion d'*extreme programming*. En effet, dans [Knublauch, 2002] une méthode de programmation de systèmes multi-agents dites extrême est proposée. Elle est basée en quelques sorte sur le code-and-fix (voir §3.4.1). Cette méthode fonctionne par observation du processus à modéliser et sollicite bien

trop le demandeur pour en faire une méthode exploitable. De plus, elle ne semble pas adaptée pour les systèmes réellement complexes.

Toutes les méthodes multi-agents sont très structurées et respectent un cycle de vie généralement en cascade (§3.4.2) [DeLoach et al., 2001]. Certaines méthodes utilisent cependant des modèle en V (§3.4.4) comme pour la méthode ADELFE [Bernon et al., 2002] ou pour la méthode MAMOSACO [Adam, 2000] (utilisation de ∇ [Kolski, 1997]). Des cycles itératifs ont aussi déjà été introduits dans Cassiopeia [Collinot and Drogoul, 1998] (cycle en W), Gaia [Wooldridge et al., 2000] et MASSIVE [Lind, 2001].

La considération des vues globales et locales propres à la conception des systèmes multi-agents encourage l'établissement d'un cycle de vie itératif. La recherce de la généricité milite pour une approche incrémentale [Mylopoulos et al., 2001, Burrafato and Cossentino, 2002].

La méthode que nous proposons couvre, dans son intégralité, le cycle de vie du système complexe physique via une spirale iterative et incrémentale. Notre méthode s'intéresse à toutes les étapes du développement d'un système complexe physique alors qu'une approche multi-agents traditionnelle ne couvrira que la branche logicielle du cycle de vie du système complexe physique. Avec de telles méthodes multi-agents, la partie décisionnelle du système est répartie dans les agents (partie logicielle): le reste du cycle de vie ne se résumerait qu'à effectuer le choix d'une plate-forme matérielle (en réalité les méthodes n'abordent pas réellement ce problème). Dans une telle optique, en plus du cycle de vie du système mixte (qui serait un simple enchaînement linéaire des activités (type cascade, §3.4.2), un cycle de vie parallèle est utilisé pour le développement du logiciel (celui de la méthode multi-agents).

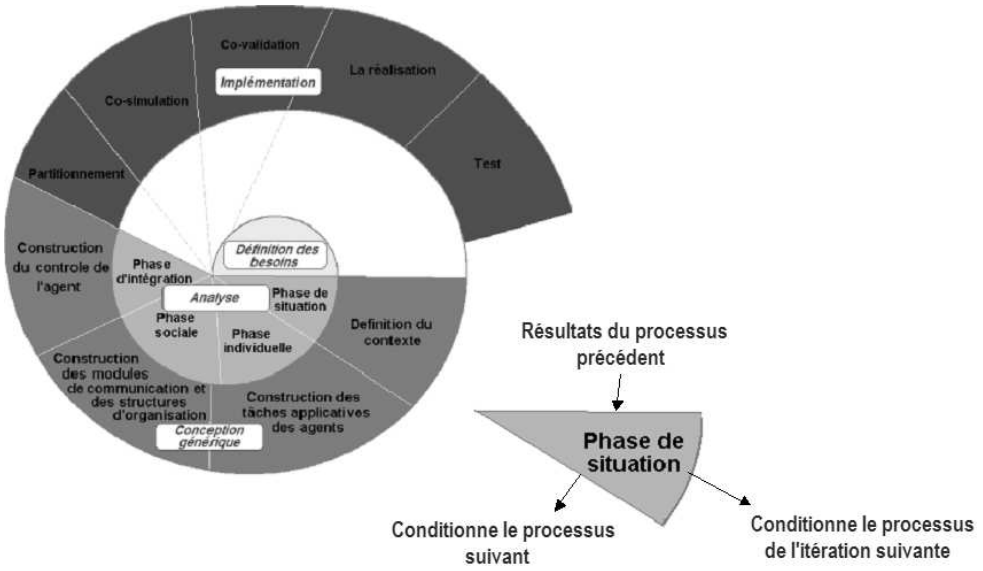


FIG. 5.19 – *Influence des activités dans notre cycle de vie en spirale*

5.6.2 Les phases

Pour couvrir la totalité du cycle de vie, différents formalismes sont nécessaires pour exprimer les différents éléments des différents niveaux d'abstraction [Herlea et al., 1999]. C'est pour cette raison que nous avons adopté un cycle axé sur quatre phases exploitant différents notations provenant de paradigmes semi-formels et de langages (agents, composants, machines à états finis, Hardware Description Langage).

Dans la phase de recueil des besoins, nous avons introduit une étude des modes de marche et d'arrêt qui n'existe qu'à cause de la criticité de nos applications physiques. En effet, dans les systèmes physiques, il est intéressant et important de pointer des modes précis de fonctionnement du système, que les méthodes tendent à négliger. Il y a bel et bien une problématique à traiter mais le système à concevoir est plus que cela : l'utilisation des méthodes multi-agents est généralement axée sur ce problème principal et a tendance à mettre de coté d'autres aspects tels que les transitions fonctionnement normal vers arrêt, les arrêts d'urgence, les arrêts pour maintenance, les marches dégradées (que les entreprises exploitent pour produire malgré des dysfonctionnements et amoindrir l'impact financier des problèmes) et surtout la sécurité des biens et personnes (primordial pour une catégorie de systèmes physiques). Un déploiement industriel de système multi-agents doit prendre en compte ces paramètres.

Les phases de déploiement sont aussi des phases assez marginales dans les méthodes multi-agents. Dans MASE cette phase permet de spécifier la place des agents sur les plates-formes existantes. Dans PASSI [Burrafato and Cossentino, 2002] cette phase a pour objectif de définir la topologie du réseau et l'attribution des ressources aux agents mobiles. Dans notre approche, cette phase transparaît à deux niveaux :

– Au niveau de l'agent via la phase du partitionnement qui donne une importance particulière à la partition matérielle/logicielle de l'agent dans le sens ou chacune des fonctionnalités de l'agent peut être déployée sur du matériel et/ou du logiciel.
– Au niveau système ou chaque agent est déployé dans l'environnement comme précisé dans la phase de recueil des besoins (voir §5.2).

Les principaux critères intervenant pour le déploiement de l'agent ont été étudiés (voir tableau 5.4).

Notre phase de conception est originale de part son côté générique. Dans les méthodes de génie logiciel, la conception générique [Roques and Vallée, 2003] fait référence au développement d'une solution qui répond aux spécifications techniques mais restant entièrement indépendante des aspects fonctionnels. Dans notre cas, elle fait en plus référence à l'abstraction de l'aspect matériel : un composant traduit le fait de traiter une fonctionnalité mais n'est en aucun cas son implémentation réelle. Ce composant pourra être décliné de plusieurs manières matérielles à logicielles.

Cette troisième phase de notre méthode a pour objectif la construction des entités élémentaires de notre système, que sont les agents, en respect de l'analyse effectuée précédemment. Cette construction des agents démarre par la coquille externe et se poursuit en direction de son coeur (l'aspect décisionnel). Cette phase est effectuée sans discriminer ce qui deviendra logiciel ou matériel. Pour cela on utilise des composants comme unité opératoire (cf §5.4). L'utilisation de composants pour implémenter des agents préoccupent de nombreux chercheurs [Kendall and Malkoun., 1997, Guillemet et al., 1999, Meurisse and Briot., 2001, Occello et al., 2002, Vercouter, 2004] mais les travaux sur des cycles de développement complets de systèmes multi-agents intégrant une dimension componentielle sont rares [Lind, 2001, Brazier et al., 2002]. Nous rappelons que nos intérêts pour l'utilisation des composants comme unité opératoire sont:

– La simplicité de compréhension et de mise en oeuvre par "l'utilisateur" du composant (programmation graphique et abstraction de la nature matérielle/logicielle),
– La gestion de la complexité par une décomposition des fonctions,

– L'incitation à la généricité (quand un composant est bien fait, il est facile de le ré-utiliser),
– Le découpage fonctionnel clair inhérent à leur utilisation qui facilite le partitionnement ,
– L'analogie composant logiciel/composant électronique qui permet aux outils du logiciel et de la description du matériel de partager une vision unifiée (voir le découpage entre description externe/interne dans les spécifications VHDL).

Il est intéressant de faire une analogie avec d'autres méthodes utilisant des composants. Dans MaSE l'ensemble des composants sont représentés sous forme de classes ou d'ensembles de classes. Les composants permettent de détailler les différents types d'agents, de définir l'architecture de l'agent. Dans MASSIVE, les composants interviennent après la sélection d'une vue[6] du système dans le raffinement de celle-ci et est en relation avec une base d'expérience qui regroupe tout les composants prédéfinis.

Les méthodes multi-agents sont beaucoup trop récentes pour intégrer des démarches qualité. Cependant, dans MASSIVE des mesures qualités sont utilisées pour évaluer la vue travaillée. Elle permet ainsi de savoir s'il faut réitérer le processus ou non.

L'utilisation de qualité dans notre approche n'est pas encore effective. Les documents créés associés à une gestion documentaire permettront une traçabilité complète des différentes étapes de la vie de l'application. Un de nos objectifs est d'intégrer une démarche complète de qualité dans notre méthode. Pour cela, nous devrons intégrer une réelle politique de qualité, c'est-à-dire permettre l'identification des exigences clients et la mise en place de mesures sur chaque activité afin de mesurer la corrélation en ce qui est produit et ce qui était initialement attendu par le demandeur. Une phase ne pourra être quittée que lorsque la qualité sera satisfaisante (comme dans MASSIVE). Un cycle de vie en spirale s'y prête d'ailleurs très bien via son évaluation des risques avant de quitter toute phase (voir §3.4.7 et §5.1.2.1).

De plus il faudra intégrer le processus qualité à la co-simulation (afin de vérifier la *qualité dynamique*[7] du système) et la co-validation.

Le tableau 5.5 présente une synthèse des cycles de vies des méthodes systèmes multi-agents. Il met aussi en évidence les points forts et faibles des activités de ces cycles [Picard, 2004].

Les critères sont:

– Besoins : l'étape de recueil des besoins est-elle prise en compte?
– Analyse : l'étape d'analyse est-elle prise en compte ?
– Conception : l'étape de conception est-elle prise en compte ?
– Implémentation : l'étape d'implémentation est-elle prise en compte ?
– Test : le processus de test est-il pris en compte ?
– Déploiement : le déploiement est-il pris en compte ?
– Maintenance : la maintenance est-elle prise en compte ?
– Délivrables : les délivrables sont-ils bien identifiés et associés à des étapes précises ?
– Gestion de la qualité : la gestion de la qualité est-elle prise en compte?
– Gestion de projet: les directives de conduite de projet sont-elles claires?

6. La méthode propose une vue environnement, une vue tâche, une vue rôle, une vue interaction, une vue société, une vue architecture qui permettent d'arriver à la vue système

7. Le bon respect des contraintes dynamiques.

	Type de cycle de vie	Besoins	Analyse	Conception	Implémentation	Test	Déploiement	Maintenance	Délivrables	Gestion de la qualité	Gestion de projet
ADELFE	V	+	++	++	+	+	±	±	++	+	++
AAII	Cascade	-	++	+	--	--	--	--	+	--	--
Aalaadin	Cascade	-	++	+	++	-	+	-	-	--	--
Cassiopée	Itératif	--	++	+	--	--	--	--	±	--	--
DESIRE	Cascade	-	±	++	±	++	--	--	-	--	--
Gaia	Itératif	-	++	++	--	-	--	--	++	--	--
MAMOSACO	Itératif	+	++	++	+	++	+	±	±	+	-
MaSE	Cascade	--	++	++	+	±	--	--	++	--	--
MASSIVE	Incrémental	+	++	++	+	+	++	±	+	-	-
MESSAGE	Itératif	+	++	++	+	±	±	±	++	+	+
PASSI	Incrémental	+	++	++	+	±	++	±	++	--	--
Prométheus	Cascade	-	++	++	+	-	--	--	+	--	--
Tropos	Incrémental	++	++	+	+	±	--	--	-	--	--
Voyelles	Cascade	-	++	++	+	+	+	--	--	--	--
DIAMOND	Spirale	+	++	++	++	(+)	++	±	+	(?)	(?)

TAB. 5.5 – *Comparaison des cycles de vie*

$(++)$: les propriétés sont pleinement et explicitement prises en charge. $(+)$: les propriétés sont prises en charge de manière indirecte. $(\pm)$: les propriétés sont potentiellement prises en charge. $(-)$: les propriétés ne sont pas prises en charge. $(--)$: les propriétés ne sont explicitement pas prises en charges.

5.6.3 Modèles et notations

5.6.3.1 Les Modèles

Les méthodes multi-agents adoptent des analyses centrées sur :

- Les rôles des systèmes à modéliser (AAII, AALADIN, Cassiopée [Collinot and Drogoul, 1998], Gaia [Wooldridge et al., 2000]),
- Les *buts* (MaSE [DeLoach et al., 2001], et Prometheus [Padgham and Winikoff, 2002]),
- La *tâches* comme pour DESIRE [Brazier et al., 2002],
- Les scénarios commme pour MASB [Moulin and Brassard, 1995].

Dans les systèmes physiques, les entités sont plongées dans des environnements qui sont eux-mêmes physiques. Quand on construit une entité, elle est faite pour évoluer dans ceux-ci. L'environnement conditionne donc naturellement développement du système complet et donc de toutes ses entités. Une fois la dimension environnementale intégrée, la suite de l'analyse dépend essentiellement de la nature du problème. Si on part d'un système pour lequel il existe déjà une représentation (le système à concevoir va automatiser des tâches manuelles par exemple) il est naturel de poursuivre en focalisant sur les agents (et leurs organisations) qui vont modéliser, par exemple, ces acteurs humains. Dans ce cas, il sera naturel de s'intéresser aux tâches qu'ils réalisent (opérations de bases) puis aux buts (comment agencer ces opérations de base). Si le système n'existe pas il va falloir, avant de se préoccuper des agents, se construire une représentation du problème et du système : il sera donc naturel de s'intéresser alors aux buts avant

même de s'intéresser aux agents.

Notre démarche pour l'analyse et la conception multi-agents unifie les approches de conception centrée sur la société (plus couramment utilisée pour les systèmes multi-agents réactifs) et centrée sur l'agent (plus couramment utilisée pour les systèmes multi-agents cognitifs). Elle effectue une itération entre le niveau société et le niveau individuel (figure 5.20). Le niveau société considère l'observation du système multi-agents tout entier. Le niveau individuel construit les agents du système.

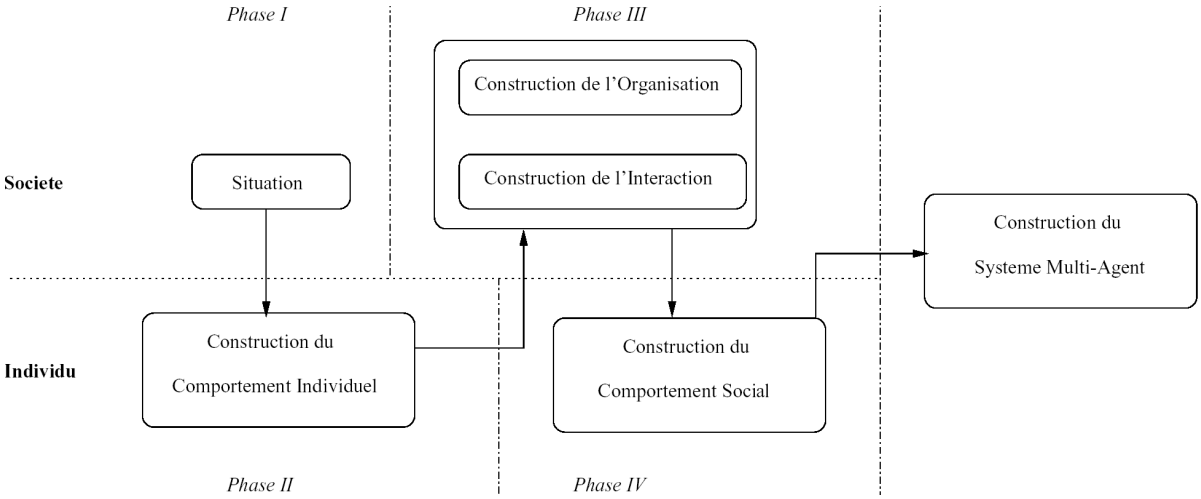


FIG. 5.20 – *Itération niveau société/niveau individu*

5.6.3.2 Les Notations

Les méthodes multi-agents opérationnelles adoptent le plus souvent des notations et des modèles d'une seule origine [Bernon et al., 2002]. Elles s'inscrivent généralement dans le prolongement des méthodes objets et reposent donc sur les notations de UML (Mase, AAII, MESSAGE, PASSI). D'autres méthodes utilisent des notations qui leurs sont propres comme DESIRE qui reposent sur les graphes (graphes d'états/dépendances pour traduire le couplage des comportements et les influences qui permettent de dériver les rôles), Aalaadin qui repose le modèle AGR (les diagrammes de structures organisationnelles, les diagrammes d'organisations concrètes et les diagrammes d'interaction) ou Gaia qui repose sur des tables (modèle de rôles et description des protocoles). D'autres méthodes exploitent plusieurs notations comme TROPOS [Mylopoulos et al., 2001, Castor et al., 2004] (notation i* issue de l'ingénierie des connaissances, A-UML pour les interactions et les diagrammes états/transitions pour les capacités et les plans) et DESIRE (notation à base de graphes pour le modèle de connaissance et une notation compositionnelle propriétaire pour les tâches) et MASSIVE (notation UML sauf pour les interactions ou ce sont des graphes d'états qui sont utilisés).

Le tableau 5.6 résume les notations utilisées par les différentes méthodes.

Notre méthode a pour vocation de couvrir toutes les étapes du cycle de vie du développement du système. Pour cela il va être nécessaire, pour exprimer les différents éléments des différents niveaux d'abstraction, de faire appel à différents formalismes [Herlea et al., 1999].

Ainsi nous commençons notre étude avec les notations UML qui ont fait leurs preuves dans le recueil

	Besoins	Analyse	Conception
ADELFE	diagrammes UML de cas d'utilisation, de collaboration, de séquence	diagrammes UML de classe, de séquence, protocoles A-UML	diagrammes UML de classes, de paquetages, stéréotypes
AAII		diagrammes UML de collaboration, de classes	diagrammes UML d'objets
Aalaadin		diagrammes d'organisation AGR	diagrammes A-UML
Cassiopée		graphes états/dépendances	
DESIRE		diagrammes de type entité/relation, graphe d'états	composants
Gaia		Tables, logique	
MaSE		diagrammes UML de séquence	diagrammes UML de classes
MAMOSACO	tableaux	diagrammes UML de classes, réseaux de Petri paramétrés, actigrammes SADT, modèle de traitement OSSAD	diagrammes UML de classes, réseaux de Petri paramétrés
MASSIVE	diagrammes de cas d'utilisation	diagrammes UML d'activités, états/transitions	diagrammes UML de classes
MESSAGE	diagrammes UML de classes	diagrammes UML de classes, d'activités, de collaboration A-UML	diagrammes UML de classes
PASSI	diagrammes cas d'utilisation UML,de séquence UML, de paquetages stéréotypés UML	diagrammes de classes UML, de séquence UML	diagrammes de déploiement UML
Prométheus		diagrammes UML et A-UML	diagrammes UML et A-UML
Tropos	i*	diagrammes états-transitions, protocoles A-UML	
DIAMOND	diagrammes UML de cas d'utilisation, de séquence, spécifications textuelles des modes de marches et d'arrêt, glossaire	diagrammes UML de séquence, protocoles A-UML, diagrammes de contexte (SART), diagrammes entité-relation (organisation)	FSM, composants, VHDL

TAB. 5.6 – *Notations utilisées par les méthodes multi-agents*

des besoins (diagrammes de cas d'utilisation, diagrammes de séquences). Nos diagrammes de cas d'utilisation différent des diagrammes habituellement manipulés (comme dans ADELFE [Bernon et al., 2002] par exemple). Cette différence ne se situe pas au niveau des notations mais dans l'interprétations des diagrammes. Tout d'abord, dans l'utilisation que l'on en fait, les acteurs sont forcément extérieurs au système. Dans un diagramme UML, les capteurs/effecteurs qu'utilise le système seraient représentés par des acteurs secondaires sur le diagramme qui reprendrait les interfaces qui leurs sont associées. Cela est dû à la nature purement logicielle de UML qui se focaliserait, dans le cas de systèmes physiques, sur la création de la partie de commande par un logiciel. Dans notre étude, on ne différencie pas les parties logicielles et les parties matérielles à ce niveau. Une deuxième différence réside dans le fait qu'il se peut que, dans notre utilisation des diagrammes provenant d'UML, des acteurs, présents dans le diagramme de cas d'utilisation, n'interviennent dans aucun des diagrammes de séquence. Dans une utilisation traditionnelle de UML cela signifierait qu'une erreur s'est glissée dans le recueil des besoins : ce qui a été identifié comme un acteur n'en était pas un. Là aussi, c'est la nature physique du système qui entraîne ce semblant d'illogisme. L'interprétation qu'il faudra avoir de cet acteur dans notre cas, est tout simplement qu'il peut avoir des interactions physiques (non désirées) tel un choc. Dans le cas des robots footballeur, un choc -dans une certaine mesure- n'aura un effet que sur la position et la direction du robot. La conséquence d'un tel choc pourra être corrigée via les informations récoltées par le système caméra.

Dans la phase d'analyse nous nous servons des diagrammes de contexte. Ce type de diagramme in-

tervient au niveau de l'agent pour mettre simplement en évidence les données qu'il peut percevoir de l'environnement et le moyen qui le lui permet (les capteurs). Il permet aussi de mettre en évidence les flots de contrôle entre les éléments purement physiques qui composent l'agent et la partie décisionnelle (celle que l'on va concevoir). Il n'est pas utilisé au même niveau que les diagrammes de cas d'utilisation. Les diagrammes de cas d'utilisation montrent les interactions acteurs/système dans leur globalité et permettent de voir les différentes utilisations possibles du système. Le diagramme de contexte permet lui de décrire la coquille externe de l'agent à l'aide d'entités situées dans un environnement physique. Il nous permet, en effet, de définir les composants de l'environnement du système tels qu'ils sont perçus du point de vue de l'agent, en les identifiant comme une série de terminaisons rattachées aux interfaces de l'agent. Il n'y a qu'un seul diagramme contextuel par agent, bien qu'il puisse s'étendre sur plusieurs pages s'il est très complexe.

Dans la phase de conception nous utilisons les composants comme unités opératoires. Ce choix a été précédemment justifié. A l'intérieur des composants on peut trouver un *container*, qui permettra l'assemblage d'autres composants, ou un formalisme qui permet de traduire un comportement. A l'heure actuelle, nous avons opté pour les machines à états finis et les spécification VHDL. Les machines à états finis sont faciles à manipuler et permettent de produire facilement une traduction en code source (partie logicielle : java, langage C) et une description matérielle (VHDL, verilog).

Les méthodes multi-agents insistent sur l'analyse pour diminuer la complexité des système auxquels ils s'intéressent : elles en oublient souvent la documentation, activité importante pour les concepteurs de l'application, l'utilisateur et même l'acquéreur. Dans Gaia, des modèles de fiches [Garro, 2003] pour la description des rôles sont créés. Cette méthode étant axée sur les rôles, ces fiches sont très importantes. Nos documents sont, pour certains, beaucoup moins spécialisés. Cela tient surtout au fait que notre méthode est très jeune. Comme nous le verrons en conclusion de nos travaux, ils nous faut approfondir la démarche qualité et la documentation. L'expérience que l'on acquerra en utilisant la méthode permettra de mieux pointer les informations et les démarches qu'il est pertinent d'adopter. Elle se retrouvera en grande partie dans les fiches qui constitueront à terme un véritable guide d'activité.

Conclusion

Cette partie, organisée en trois chapitres, a permis de répondre à bon nombre de questions que soulevait notre problématique.

Dans un premier chapitre, nous avons exposé les généralités de la production de logiciel en insistant sur les cycles de vie. Le second chapitre a présenté le domaine des systèmes embarqués et s'est rapidement centré sur le co-design présenté comme une alternative au cycle de vie traditionnel du développement de ces systèmes. Le co-design tente, en effet, d'unifier en un cycle de vie unique la production du logiciel et du matériel. Le troisième chapitre expose notre méthode et une discussion de celle-ci.

Nous proposons, en guise de conclusion, de répondre aux questions soulevées dans l'introduction.

Quel cycle de vie pour construire des systèmes complexes physiques ouverts? Nous souhaitons, pour notre méthode, avoir un cycle de vie qui crée des liens forts en chacune des phases, chacun des processus, chacune des activités afin de limiter, entre autres, les erreurs d'appréciation notamment entre l'analyse et la conception.

La nature complexe des applications qui nous intéressent rend quasiment obligatoire la possibilité de pouvoir procéder à des itérations successives (c'est un des moyens d'approcher la complexité). Ce besoin est encore renforcé lorsqu'on traite de système multi-agents car les points de vue individuels et sociaux peuvent être contradictoires.

Nous souhaitons avoir une réelle démarche qualité car la qualité peut aider les analystes et les concepteurs dans leur tâches. Quand une démarche qualité est clairement établie, le risque de ne pas prendre en compte des paramètres (contraintes techniques ou technologiques, erreurs d'interprétation etc.) est moins élevé.

Tous ces éléments, nous ont conduit à choisir un cycle de vie en spirale.

Comment spécifier un système mixte logiciel/matériel? Comment harmoniser au mieux, dans ce cycle de vie, les spécifications du logiciel et les spécifications matérielles? Par souci de simplicité, nous souhaitons permettre la construction des applications multi-agents en utilisant un formalisme de programmation essentiellement graphique. L'objectif n'est pas d'écrire des lignes de programme dans un langage textuel mais de manipuler des objets graphiques (les composants) à la manière de Ptolémy qui avait été le précurseur des méthodes co-design. Un composant peut être construit à l'aide d'autres composants ou contenir un formalisme qui décrit son comportement. La méthode invite tout de même à se focaliser sur les machines à états finis, mais on pourra utiliser un formalisme au choix. Pour être en harmonie totale avec l'esprit de la méthode, il suffit que ce formalisme permette la génération d'un code logiciel ou d'une spécification matérielle.

Le cycle de vie que nous avons proposé ne donne pas lieu à un partitionnement précoce comme dans

les méthodes traditionnelles pour lesquelles, dès le début du cycle de vie, on détermine un cahier des charges pour le logiciel et pour le matériel.

La spécificité des systèmes multi-agents physiques va-t'elle demander de procéder à une analyse des besoins particulière? La spécificité des systèmes multi-agents physiques nous a conduit à introduire un processus d'étude des modes de marches et d'arrêt. Cette étude permet de préciser des contraintes, qui auraient pu être oubliées, en structurant le fonctionnement global. Cette structuration peut aider à identifier des situations de coopération, de raffiner sur le diagramme de cas d'utilisation etc.

Comment intégrer au mieux les contraintes locales des éléments physiques aux fonctionnalités globales du système? Pour cela notre méthode utilise une analyse qui commence par se focaliser sur l'individu et son environnement (phase de situation et phase individuelle). L'étape suivante permet de considérer le niveau social du système (phase sociale). Enfin, la dernière phase permet d'identifier les influences sociales sur le comportement individuel de l'agent afin d'assembler et d'adapter judicieusement les deux types de comportements.

Comment concevoir les parties logicielles, matérielles et les intégrer en un tout fonctionnel? Pour cela nous avons utilisé une approche co-design. Durant tout le cycle de vie nous n'avons pas discriminé le logiciel et le matériel (hormis pour les périphériques d'entrées/sorties qui sont identifiés et représentés via un diagramme de contexte). Ce n'est qu'à partir de la phase d'implémentation, notamment au niveau du partitionnement, que l'on distingue ces deux parties. La co-synthèse permet de les intégrer.

Hormis pour les effecteurs et les capteurs, comment choisir ce qui doit être logiciel ou matériel? Nous avons défini pour cela différents critères présentés sous la forme d'un tableau (table 5.4). Ces critères sont le coût, la performance, la flexibilité, la tolérance aux pannes internes, l'ergonomie et la complexité algorithmique.

Comment tester un système multi-agents physique? Dans notre méthode il y a deux niveaux de "tests". Les activités de co-simulation et de co-validation permettent de tester la qualité de l'agent comme entité autonome soumise à des contraintes (temporelles, de tolérance aux pannes etc.).

Enfin, en fin de cycle, le test du système dans son intégralité est à effectuer.

Comment débugger un système multi-agents physique? Cette question n'a pas réellement été abordée. Dans un système physique, tout comme dans un système logiciel, cette opération reste difficile : un comportement qui semble absurde d'un agent viendrait-il d'un problème de l'agent? d'une mauvaise représentation de son environnement? d'une demande farfelue ou erronée d'un autre agent?

Quelle démarche qualité pour la conception de tels systèmes? Le point de départ de notre démarche qualité réside dans la volonté d'avoir une traçabilité complète des activités. Le cycle de vie, pour sa part, permet une interpénétration plus forte entre les activités de tous les processus de toutes les phases. De plus, de par sa nature, le cycle de vie est prévu pour permettre l'identification du risque à chaque itération (que nous n'avons pas pour le moment totalement exploitée).

troisième partie

UNE ARCHITECTURE SMA POUR LA GESTION DES COMMUNICATIONS DANS LES SYSTEMES COMPLEXES PHYSIQUES AUTONOMES : L'INTERGICIEL MWAC

Introduction

La seconde problématique soulevée par la confrontation des systèmes multi-agents aux systèmes complexes physiques ouverts consiste en des besoins spécifiques architecturaux. Ces besoins concernent entre autres une abstraction de la gestion des communications entre les entités autonomes qui composent le système. Cette gestion des communications doit obéir à certains critères tel que le coût énergétique ou le temps d'obtention d'une route.

Dans cette partie, le premier chapitre va nous permettre de nous intéresser à la gestion des communications sans fil dans les systèmes complexes physiques ouverts. Ces communications relevant du domaines des réseaux sans fil nous présenterons des généralités sur les différents type de réseaux et insisterons sur la famille qui nous parait être la plus adéquate : les réseaux ad-hoc. Nous donnerons les principes généraux sur le routage et présenterons une revue des protocoles de communication établis par la communauté des réseaux. Des travaux sur des systèmes multi-agents dans ce contexte seront présentés.

Dans un second chapitre, nous introduirons les techniques de maintien d'intégrité fonctionnelle et insistons grandement sur les techniques basées sur l'auto-organisation. Nous procédons à une revue des mécanismes d'auto-organisation.

Dans un dernier chapitre nous présentons notre architecture multi-agents pour le management des communications sans fil. Ce modèle s'inspirera en partie de CGSR qui a été identifié dans la revue des protocoles de routage ad-hoc. Notre modèle utilise l'auto-organisation qui est mise en oeuvre via un mécanisme d'attribution de rôle (abordé dans la revue des mécanismes d'auto-organisation). Par souci de généricité, ce support pour la communication sans fil résidera dans un intergiciel dont la mise en oeuvre sera détaillée dans l'application de la partie suivante.

Chapitre 6

Communication dans les systèmes complexes physiques sans fil

Ce chapitre présente tout d'abord les généralités sur les réseaux sans fil qui sont en partie à l'origine de la complexité inhérente aux applications qui nous intéresserons par la suite. Ce chapitre fait aussi la revue des protocoles de routages utilisés dans les réseaux ad-hoc : il présente les notions de base des protocoles de routages et propose des critères d'évaluation de ces protocoles.

6.1 Généralités

Dans l'avenir, les réseaux de communications permettront aux utilisateurs de profiter de tous services indépendamment leur position géographique. Ces réseaux du futur regrouperont toutes les infrastructures déjà existantes comme par exemple :

- Les petits réseaux tels que les PAN[1] (Personnal Area Network) ou PCN (Personnal Communication Network)
- Les réseaux locaux et industriels traditionnels comme les LAN[2] (Local Area Network), HiperLAN[3] (High-Performance radio LAN) et Ethernet[4]
- Les grands et très grands réseaux tels que les MAN[5] (Metropolitan Area Network) et WAN[6] (Wide Area Network)
- Les réseaux téléphoniques cellulaires (puis probablement picocellulaires[7]).

Nous allons nous intéresser dans cette partie aux concepts qui vont nous être nécessaires pour bien comprendre la problématique du projet ENVSYS.

1. Nom donné aux réseaux domestiques permettant d'interconnecter ordinateurs, téléphones portables...
2. Ces réseaux sont particulièrement adaptés à la majorité des entreprises : la distance entre les deux points les plus éloignés du réseau doivent être de quelques kilomètres
3. Normalisation européenne de réseaux locaux sans fil
4. Réseau local très répandu autorisant des débits élevés via câble coaxial, fibre optique, paires torsadées...
5. Réseaux atteignant la taille d'une métropole
6. Réseaux étendus sur plusieurs centaines de kilomètres
7. Cellule de rayon inférieur à 50m

6.1.1 Le modèle OSI

Le modèle de référence de connexion entre systèmes ouverts OSI (reference model of Open Systems Interconnection) a pour vocation de permettre la création de logiciels modulaires et réutilisables indépendamment des techniques mises en oeuvre. En d'autres termes, il faut que la conception et la physionomie du système ne soient pas propriétaires[8].

Ainsi, le modèle OSI propose une architecture composée de sept couches (voir figure 6.1). Chacun de ces sept modules propose un service aux couches qui lui sont adjacentes via un ou plusieurs protocoles.

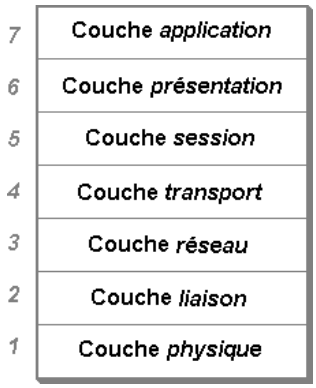


FIG. 6.1 – *Modèle en couches OSI*

Voici brièvement, le rôle de chacune des couches OSI :

- La couche *physique* sert à transmettre les données via un canal de communication[9] et comprend le matériel nécessaire à la réalisation de cette fonction,
- La couche *liaison* a pour fonction initiale la détection d'erreurs dans la couche physique,
- La couche *réseau* permet l'échange de données entre les hôtes du réseau,
- La couche *transport* permet de faire parvenir les données aux applications exécutées par les hôtes,
- La couche *session* est l'interface utilisateur du réseau,
- La couche *présentation* s'occupe des détails relatifs aux interfaces comme, par exemple, les imprimantes, les formats de fichiers et les écrans,
- La couche *application* contient les informations sur les applications réparties sur le réseau.

6.1.2 Architectures de réseaux sans fil

Le terme générique de réseaux sans fil cache différents types d'architectures allant des structures de réseaux sans fil simples aux réseaux ad-hoc. Cette évolution a suivi un ordre chronologique.

Dans un réseau sans fil simple (voir figure 6.2, cas 1), comme wifi ou WiMAX, il existe un point d'accès et des équipements qui échangent avec ce point plutôt que directement entre eux. La fonction première de ce point d'accès est de permettre la communication entre les équipements. Il peut cependant aussi les relier à un autre point d'accès ou directement à un autre réseau (éventuellement filaire) comme. internet.

8. un système est propriétaire s'il ne peut pas être utilisé pour une autre application que celle pour laquelle il a été conçu
9. le canal de communication est le milieu dans lequel se propage le signal

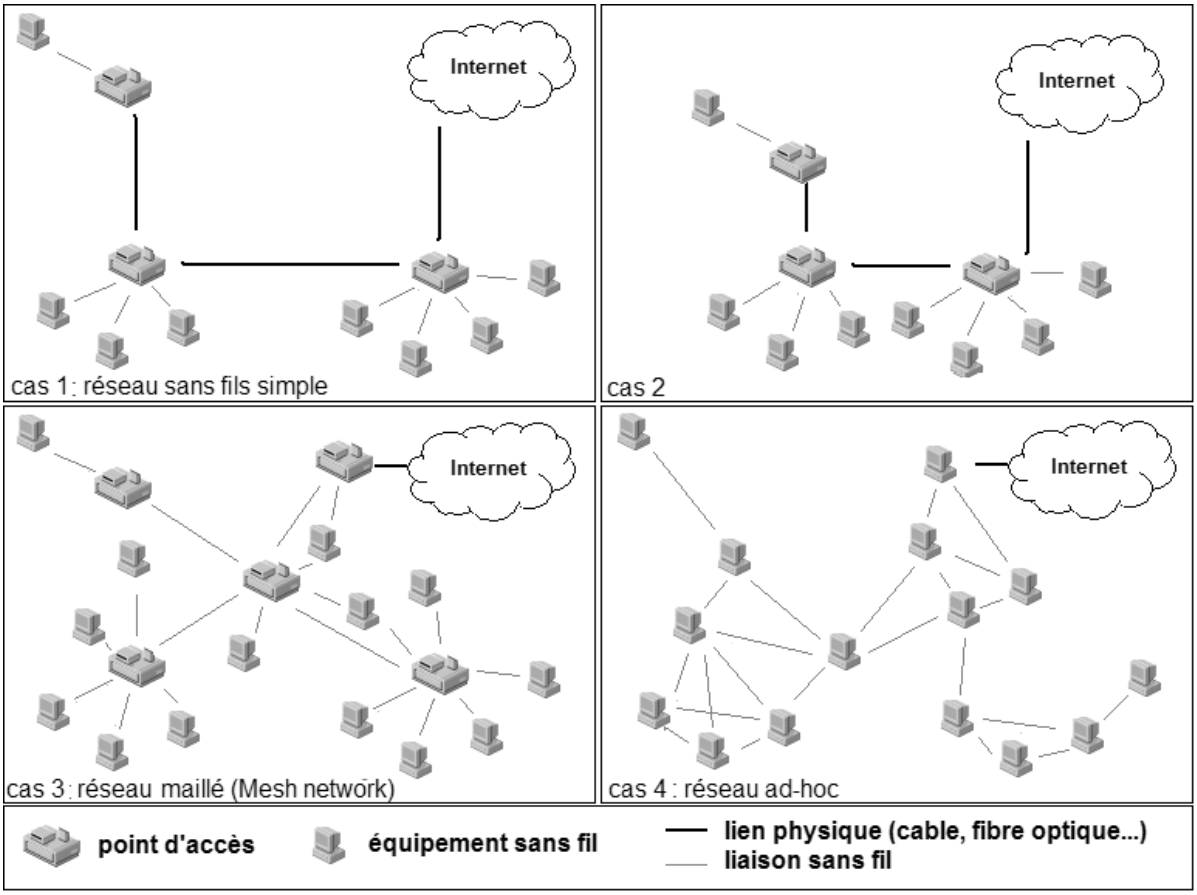


FIG. 6.2 – *Les différentes architectures sans fil*

L'étape suivante (voir figure 6.2, cas 2) a consisté à réaliser un réseau sans-fil utilisant la fonctionnalité de "Handover" afin de rendre transparent [10] le passage d'une zone associée à un point d'accès à celle d'un autre. Il est ainsi possible de passer en continu d'une zone à l'autre en se déplaçant comme dans les réseaux de téléphonie mobile ou réseaux Wi-Fi intégrant le protocole IEEE 802.11f. Le réseau cellulaire obtenu est équivalent à un grand réseau sans-fil dans lequel on peut se déplacer. La contre-partie de cette simplicité et de cette transparence d'accès par un utilisateur à un grand réseau sans fil est qu'il faut interconnecter tous les points d'accès entre eux via un deuxième réseau ce qui augmente la complexité des infrastructures.

L'objectif de l'évolution suivante (voir figure 6.2, cas 3) est de supprimer le réseau d'interconnexion entre les points d'accès. Le postulat est qu'un point d'accès se trouve dans la zone de couverture d'au moins un autre point d'accès pour échanger directement avec lui à travers son propre réseau sans fil. Il n'est donc plus nécessaire de relier les points d'accès avec des moyens différents comme des câbles électriques. Ces réseaux sont constitués de "mailles" qui communiquent directement avec leurs voisines : ce sont les réseaux Mesh. Le rapprochement des points d'accès entre eux entraîne une réduction des distances entre points par un rapport deux (on a alors quatre fois plus de points d'accès pour une zone

10. On parle de passage transparent quand une communication n'est pas coupée lors d'un changement de zone

donnée). Le réseaux sans fil obtenu ne contient donc plus qu'un seul type d'infrastructure : les réseaux sans-fil constitués des différentes zones de couvertures des points d'accès. La contre partie est qu'il faut installer plus de points d'accès. Il a aussi fallu adapter les protocoles car le routage entre deux cellules ne se fait plus sur un réseau d'interconnexion séparé mais directement dans les réseaux sans fils entre les points d'accès.

Enfin, la dernière génération de réseaux sans fil sont les réseaux ad-hoc (voir figure 6.2, cas 4). Dans ces réseaux, l'objectif consiste à supprimer les points d'accès, ou plus exactement à transformer tous les équipements en points d'accès. Il n'y a donc plus besoin de mettre en place une infrastructure fixe de points d'accès : sous la couverture d'un élément du réseau on a automatiquement accès à tout le réseau. Cependant les équipements-points d'accès peuvent changer de place, disparaître, etc. Il faut aussi que la densité des équipements sur la zone que l'on souhaite couvrir soit suffisamment importante pour que chacun ait un ou plusieurs "voisins" dans sa zone de couverture. Cela ne pose pas de problème lorsque les équipements sont suffisamment proches pour communiquer directement entre eux (c'est le mode "ad-hoc" qui existe dans les réseaux Wi-Fi par exemple, par opposition au mode "infrastructure" qui nécessite un point d'accès). Les choses commencent à devenir plus complexes lorsque les équipements qui doivent communiquer entre eux sont plus éloignés et doivent passer par d'autres équipements intermédiaires. Chaque équipement ad-hoc doit disposer alors de fonctionnalités de routage et communiquer avec ses voisins pour constituer un réseau maillé (mesh network). Un routeur permet grâce à ses "tables de routage" de constituer de proche en proche une "route" pour acheminer les paquets d'information d'un point à un autre. Mais dans le cas du réseau ad-hoc, il n'y a plus de points d'accès supposés fixes. Il n'est donc plus possible de garantir la pertinence des tables de routage à l'avance. Comme nous le verrons en §6.4, les réseaux ad-hoc essaient malgré tout de constituer périodiquement des tables de routage à l'avance (cas des protocoles proactifs) ou bien cherchent à constituer une route à chaque fois au moment où l'on souhaite communiquer (protocoles réactifs). Chaque type d'approche dispose de ses avantages et de ses inconvénients qui les destinent plus particulièrement à tel ou tel type de réseaux.

6.2 Les réseaux ad-hoc

6.2.1 Définition

> Les réseaux ad-hoc sont des réseaux composés uniquement de station qui doivent assumer les fonctions de routage. Ces réseaux doivent donc s'organiser automatiquement afin d'être opérationnels dès qu'ils sont mis en place [Agha et al., 2002]. Lorsque ces réseaux possèdent des noeuds mobiles on parle alors de réseaux MANET[11] (Mobile Ad-hoc NETwork)

La direction des recherches actuelles sur ces réseaux tend à assimiler complètement la notion de mobilité, la généraliser à toutes les composantes de leur environnement. Il est important de garder à l'esprit que dans ces réseaux, contrairement aux réseaux cellulaires, il n'existe aucune administration centralisée. De plus, aucune contrainte ne limite sa taille en terme d'étendue ou de nombre d'unités.

Actuellement les applications des réseaux ad-hoc sont essentiellement militaires mais on les retrouve de plus en plus dans des applications telles que les bases de données parallèles, le télé-enseignement, la simulation distribuée etc...

6.2.2 Caractéristiques de ces réseaux

On peut dégager six grandes caractéristiques [Agha et al., 2002, Lemlouma, 2000]:

– La topologie est dynamique. C'est une conséquence directe de la mobilité des unités qui composent le réseau. Le tracé des routes peut changer pendant l'émission d'un paquet.
– Les liens sont asymétriques. En effet, la liaison entre deux unités n'est pas forcément bidirectionnelle.
– La bande passante est limitée car les communications par voies hertziennes imposent un partage du médium de communication entre les différents hôtes. On appelle ce phénomène réutilisation spatiale. Ce phénomène est dû à l'atténuation des signaux avec la distance qui fait que même si un médium peut être utilisé simultanément par plusieurs hôtes, il peut y avoir des collisions et donc la nécessité de procéder à des ré-émission.
– Les contraintes énergétiques sont fortes. Cela est du au fait que chacune des unités doit bien souvent embarquer une alimentation autonome.
– Il n'y a pas d'infrastructure pré-existante et persistante. En effet, les hôtes sont mobiles et c'est a eux qu'il incombe de maintenir les connexions.
– Il y a de nombreuses interférences entre les hôtes du réseau ou encore d'une onde avec elle même (cas d'une réflexion d'onde). Elles accroissent le nombre d'erreurs sur la transmission et imposent un amoindrissement des performances.

6.3 Généralités sur le routage dans les réseaux ad-hoc

6.3.1 Le routage : principes fondamentaux

Comme nous l'avons vu en §6.1.1, c'est à la couche réseau qu'incombe le rôle d'acheminer les informations d'un hôte à un autre : les protocoles de routage sont donc des protocoles associés à la couche 3 car leur fonction est de déterminer le chemin que devra emprunter un message (ou les paquets) dans le réseau maillé des différents hôtes.

Nous donnons ci-dessous, une brève explication des termes et des notions que nous utilisons par la suite.

Routage par la source et routage par la cible. Dans les routages par la source, ce sont les noeuds qui émettent les messages qui déterminent la liste des noeuds que les paquets de données doivent traverser. A contrario, dans le cas du routage par la cible le noeud source signale qu'il veut transmettre un message mais c'est le destinataire qui lui dira comment y parvenir.

L'inondation. L'inondation est une technique de routage fréquemment utilisée et qui consiste à faire parvenir un paquet à tous les noeuds du réseau. Ainsi, un noeud qui reçoit un paquet le transmet à tous ses voisins directs. Ce type de routage entraîne une charge importante sur le réseau et engendre des problèmes tels que des bouclages dit de routage [12].

Le multihoping. Dans les communications de type cellulaire les communications passent par des stations dites de base et un réseau filaire : les stations mobiles ne servent jamais de routeurs intermédiaires,

12. Nom donné au fait qu'un paquet passe plusieurs fois par les mêmes noeuds

ce modèle est donc dit *Single Hop*. Dans un modèle de communication sans infrastructures de ce type (c'est le cas des réseaux ad-hoc), les noeuds participent au routage : le modèle est dit *multihop*.

L'Algorithme *Distant Vector*. Dans cet algorithme chaque élément qui participe au routage diffuse à ses voisins directs sa table de routage à chaque modification de sa table et à des intervalles de temps fixes. Cette diffusion périodique permet de diminuer le nombre de boucles de routage.

Si un noeud ne reçoit pas la table d'un de ses voisins pendant une période de temps fixée à l'origine il considérera ce noeud comme défaillant.

A chaque fois qu'une des stations reçoit une table de la part de son voisin, elle rajoute les entrées qu'elle n'avait pas et, s'il existe des routes à moindre coût, elle les adopte. Cet algorithme est celui de Bellman-Ford [Cormen et al., 2002].

L'Algorithme *Link State* et l'*Open Shortest Path First*. L'algorithme vu précédemment peut présenter des problèmes de convergence c'est-à-dire conduire à des boucles de routage. Pour éviter complètement les boucles il faut que chacun des noeuds ait une connaissance globale de la topologie du réseau. L'objectif de l'algorithme Link State [Cormen et al., 2002] est donc de donner aux noeuds cette connaissance.

Les protocoles basés sur la diffusion de tables de routage comme RIP (Routing Information Protocol), peuvent convenir pour les petits réseaux. Par contre, pour les grands réseaux il est nécessaire d'utiliser d'autre protocoles tels que OSPF.

6.3.2 Critères d'évaluation d'un protocole de routage

Les critères qui permettent l'évaluation d'un protocole de routage découlent de leur nature même. Ces critères sont :

- La fiabilité du protocole : est-on sûr qu'un message arrive à bon port? Sinon, est-on prévenu?
- La part des messages de contrôle utilisés par le protocole par rapport à la quantité utile d'informations transmises : il faut donc éviter de concentrer le trafic en un seul point, éviter les boucles de routage, réduire au strict nécessaire le nombre de messages de contrôle transmis.
- Le caractère optimal des routes qui fait référence au chemin pris par les paquets (ont-ils pris le chemin le plus court? etc...)

6.4 Les différents protocoles existant

Il existent trois familles de protocoles de routage ad-hoc :

– Les protocoles ad-hoc *réactifs* sont des protocoles qui ne cherchent des routes que lorsque cela est nécessaire (on-demand protocol) c'est-à-dire lorsqu'un message doit être transmis. Ces protocoles utilisent beaucoup les techniques d'inondation car lorsqu'une station souhaite transmettre des données, elle ne sait pas encore le chemin qu'il faut prendre pour joindre le destinataire. Les tables de routages, si elles existent, ne le sont que temporairement. Un des avantages majeurs de cette famille de techniques de routages est le fait que la bande passante n'est essentiellement utilisée que pour la transmission de données.

– Les protocoles ad-hoc *proactifs* qui nécessitent l'utilisation d'une table de routage mise à jour continuellement via l'échange de paquets de contrôle. En quelque sorte, ils créent un modèle du réseau (généralement partiel).

– Les protocoles *hybrides* qui se comportent comme des protocoles réactifs utilisant, à la demande, des tables de routage pour accroître leur rendement.

Dans cette partie nous allons nous intéresser qu'aux principes utilisés dans les protocoles existants, dont le critère de classification sera la famille à laquelle ils appartiennent.

6.4.1 Les protocoles de routage réactifs

Comme nous en avons brièvement parlé précédemment, les réseaux ad-hoc qui exploitent des protocoles réactifs n'utilisent pas de table de routage. Lorsqu'une station décide d'émettre un paquet, elle ne connaît pas le chemin à prendre. Cet hôte va donc procéder par inondation, c'est-à-dire qu'il va émettre à tous ses voisins le paquet et ceux-ci agiront de même. Ainsi, le réseau est littéralement inondé par le message initialement transmis et qui n'a pourtant à l'origine qu'un seul destinataire. Il arrive ainsi très fréquemment que le destinataire reçoive plusieurs fois le même message. Cette famille de protocole est donc très gourmande en ressources et à une performance très faible.

Voici les neuf protocoles de cette famille que nous avons recensés :

6.4.1.1 Le protocole DSR

Le protocole DSR (Dynamic Source Routing [Johnson and Maltz, 1996]) est un protocole utilisant le principe du routage par la source. Ne possédant pas de table de routage, les noeuds procèdent par inondation à l'aide d'un message ROUTE REQUEST. La succession de stations est mémorisée dans le message, ce qui permet de connaître la route qu'il a emprunté. Le destinataire, lorsqu'il reçoit ce message, décide du meilleur chemin pour l'acheminement des paquets et répond par un ROUTE REPLY. La station source, une fois qu'elle a reçu le ROUTE REPLY précise dans tous les paquets le chemin qu'il faut suivre (voir figure 6.4.1.1.

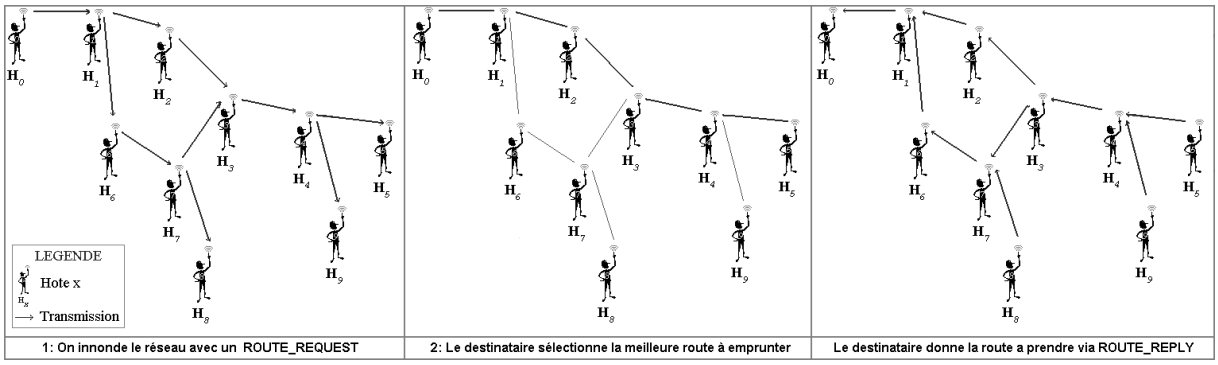


FIG. 6.3 – *Illustration du fonctionnement du protocole DSR*

Un des inconvénients de cette méthode est donc la surcharge en messages utilisés pour déterminer une route.

6.4.1.2 Le protocole AODV

Les concepteurs du protocole *Ad-hoc On-demand Distance Vector* [Perkins and Royer, 1999] ont améliorés le fonctionnement de DSDV (cf §6.4.2.1). Ce protocole utilise donc les mêmes techniques basées sur le principe du vecteur de distance (algorithme de Bellman-Ford) mais diminue le nombre

de diffusions de messages nécessaires à la mise à jour des tables de routage. Contrairement à DSDV qui maintient à chaque instant la totalité des routes, ce protocole ne les construira que lorsqu'il en aura besoin (il est donc bien réactif). Il utilise pour cela deux mécanismes :

- Le premier consiste à créer une route. Cela passe par l'utilisation d'une "requête de route" appelée RREQ (Route REQuest). Tous les noeuds par lequel passe cette requête (il y a une inondation du réseau) modifie leur table et seul le destinataire répondra par un RREP (Route REPly). Ainsi, ce sont des tables locales à chaque noeud et construites à la demande de la source du message qui serviront à aiguiller les paquets.
- Le second mécanisme consiste à veiller à la validité de la route tant que l'on s'en sert. Ainsi, si la route devient invalide (suite par exemple au dysfonctionnement d'un des hôtes), le noeud victime de cette rupture délivre un message d'erreur à la source que l'on appelle URREP (Unsolicited RREP). Pour assurer cette validité d'un lien on procède à l'utilisation périodique de message HELLO. Si trois de ces messages ne sont pas reçus consécutivement par un voisin, il le signale via le URREP.

6.4.1.3 Le protocole CBRP

Le protocole CBRP (Cluster Based Routing Protocol [Jiang et al., 1999]) propose de décomposer l'ensemble des noeuds en de petits groupes à la manière de CGSR (développé en §6.4.2.7). Pour cela chaque noeud maintient une table de voisins précisant leur identifiant, leur statut ainsi que l'état du lien qui les sépare (uni ou bidirectionnel). Le représentant du groupe maintient en plus une table des groupes adjacents représentant les noeuds de son groupe qui mènent à des représentant d'autres groupes. Un noeud qui veut en contacter un autre diffuse ainsi une requête à tous les différents représentants. Chaque représentant regarde si le destinataire est dans son groupe : si c'est le cas il répond en utilisant le chemin pris par l'autre paquet sinon diffuse la requête aux autres représentants.

6.4.1.4 Le protocole SSR

Le protocole SSR (Signal Stability-based Routing [Dube et al., 1997]) se base sur la puissance des signaux entre les noeuds et de leur stabilité de localisation. Ces deux critères permettent de sélectionner les routes les plus stables.

Pour cela le protocole SSR fait coopérer deux protocoles :

- Le protocole DRP (Dynamic Routing Protocol) qui, par l'intermédiaire d'une table de stabilité de signal (dit SST pour Signal Stability Table) qui pour chaque voisin détermine si le canal est à forte puissance ou non et une table de routage traditionnelle RT (Routing Table), traite tous les messages reçus par un noeud et le transmet au protocole SRP (Static Routing Protocol).
- Le protocole SRP se sert de la table de routage RT pour déterminer le voisin à qui faire suivre le message. Si la destination n'est pas connue alors elle diffuse un paquet afin de trouver la route. Ce sera la route prise par la première réponse reçue à cette requête qui servira car la probabilité que cette route soit la plus courte est plus élevée.

Cette méthode privilégie les chemins les plus fiables grâce à la SST (Signal Stability Table).

6.4.1.5 Le protocole LAR

Le protocole LAR (Location-Aided Routing [Ko and Vaidya, 1998]), comme son nom l'indique, utilise des critères de location des noeuds. Ce protocole fonctionne comme le protocole DSR vu en §6.4.1.1

mais utilise les informations de localisation que lui fournit le système de positionnement global appelé GPS. Il en découle deux stratégies pour diminuer la diffusion des paquets par inondation. Une première qui consiste à déterminer une région dans laquelle on est sûr de trouver le noeud et de ne diffuser le message que dans cette région et une seconde qui consiste à calculer à chaque réception par un noeud intermédiaire la distance de la cible. Si elle est inférieure à celle donnée par le paquet précédent, on continue l'envoi.

Pour ces deux cas, si le message n'est pas reçu (on utilise pour cela un timeout) le noeud source diffuse un message requête de route.

6.4.1.6 Le protocole RDMAR

Le protocole de routage RDMAR (Relative Distance Micro-Discovery Ad hoc Routing) est basé sur la découverte des distances relatives [Aggelou and Tafazolli, 1999]. Il a été conçu avec pour objectif de minimiser la charge induite par les changements rapides de topologie. Il utilise un mécanisme de découverte de routes appelé RDM (Relative Distance Micro-discovery). L'idée à l'origine de ce protocole est de diffuser les requêtes qu'à une partie des noeuds : le critère de sélection étant la distance, entre l'émetteur et le destinataire, estimée par un algorithme itératif. C'est le noeud destination qui décidera du chemin à prendre.

Un noeud qui détecte un problème de lien le signale en diffusant un avertissement.

6.4.2 Les protocoles de routage proactifs

La famille des protocoles ad-hoc que l'on nomme proactifs est, comme nous l'avons vu précédemment, celle qui utilise des tables de routage. Pour cela, il est nécessaire que chaque noeud procède à des échanges de messages avec les autres afin de rechercher à chaque fois les routes optimales. Ces tables de routage sont dynamiques et s'adaptent aux changements de configuration du réseau.

Voici les neufs protocoles que nous avons recensés pour cette famille :

6.4.2.1 Le protocole DSDV

Le protocole DSDV (Destination Sequence Distance Vector [Perkins and Royer, 1999]) est l'un des premiers à être spécifié par le groupe de travail MANET. Il s'inspire très naturellement du protocole RIP. Pour illustrer cet exemple considérons le réseau R1 (figure 6.4.2.1 et la table de routage associée gérée par ce protocole.

La table gérée par le protocole DSDV (voir table 6.1) nous permet de savoir que lorsque H1 veut joindre H5, la métrique sera de 2 et la station qui servira de relais sera, dans un premier temps, H2.

Destination	Nombre de sauts	Prochains noeuds
H1	0	H1
H2	1	H2
H3	2	H2
H4	2	H2
H5	2	H2
H6	1	H6

TAB. 6.1 – *Table de routage de l'hôte H1*

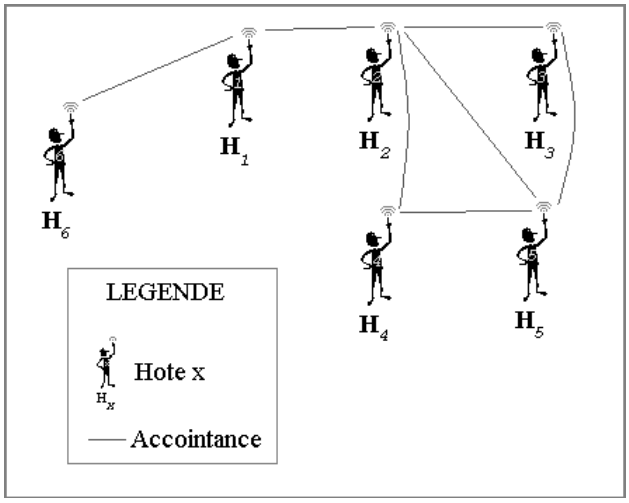


FIG. 6.4 – *Illustration du fonctionnement du protocole DSDV*

Le principal défaut de ce protocole (qui existe aussi dans RIP) concerne la convergence des tables; il provient du fait que chaque hôte transmet sa table de routage à ses voisins. Une station qui reçoit une table compare son contenu au sien et corrige sa propre table pour déterminer le plus court chemin. Cependant, le nombre de sauts n'est pas toujours déterminé! Certains noeuds sont donc non joignables.

De plus, le ratio messages utiles envoyés sur le total des messages envoyés est mauvais surtout si les noeuds sont mobiles.

Le protocole DSDV, comporte donc de nombreux défauts : il n'est, aujourd'hui, plus utilisé. Cependant il a été amélioré et est devenu le protocole AODV et a inspiré bon nombres d'autres protocoles.

6.4.2.2 Le protocole OLSR

Le protocole OLSR (Optimized Link State Routing [Clausen et al., 2001]) applique des règles de routage centrées lien[13]. En effet, dans les réseaux ad-hoc équipés de ce protocole, c'est l'état du lien qui renseigne sur le placement des autres hôtes. Le choix de la route (quelle soit en fonction de la rapidité ou de la métrique) fait intervenir l'algorithme Dijkstra[14] [Cormen et al., 2002], très utilisé pour la découverte de route.

Lors d'une communication, un noeud élit parmi ses voisins un représentant qui deviendra MPR (Multi-Protocol Router). Ce noeud, qui joue le rôle de relais multipoints, a pour mission d'acheminer l'information. Les MPR sont choisis grâce à l'envoi de message HELLO que les noeuds s'échangent pour connaître la nature des liens[15] qui les sépare. Un noeud aura le statut de MPR s'il peut atteindre avec un lien symétrique un voisin éloigné de plus de deux sauts. L'identifiant de tous les MPR est communiqué au réseau via des messages TC (Topology Control) à intervalles de temps réguliers ce qui permet aux noeuds de mettre à jour leur table de routage.

Ainsi voyons ce que nous obtenons pour le réseau figurant en 6.5.

Les MPR de ce réseau seront :

- H1 pour H0, H2, H6.

13. Un routage centrée lien est un routage qui tient compte de l'état et de la qualité du médium de communication
14. cet algorithme permet de calculer le plus court chemin le long d'un graphe d'état
15. Par nature des lien on entend symétrique ou asymétrique

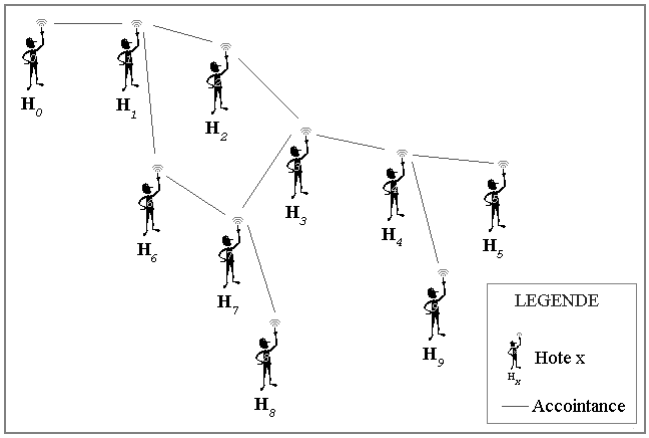


FIG. 6.5 – *Illustration du fonctionnement du protocole OLSR*

- H2 pour H1 et H3
- H3 pour H2, H4 et H7
- H4 pour H3, H5 et H9
- H6 pour H1 et H7
- H7 pour H3, H6 et H8

Le fait qu'il ne fasse pas intervenir l'inondation augmente les performances.

6.4.2.3 Le protocole WRP

Le protocole WRP (Wireless Routing Protocole [Murthy and Garcia-Luna-Aceves, 1995]) est basé sur ce que l'on appelle les algorithmes PFA (Path-Finding Algorithm). Ce protocole oblige les noeuds à posséder :

- une table de distance qui est en fait une matrice contenant pour chaque couple de noeuds possibles leur distance,
- une table de routage,
- une table de coûts des liens qui contient entre autres les "timeout" associés à chaque noeud,
- une liste de retransmission de messages qui permet de connaître les noeuds voisins qui n'ont pas acquitté le message de mise à jour et de pouvoir ainsi leur retransmettre.

Les mises à jour sont envoyées à chaque changement d'état d'un des liens voisins ou après réception des données de mise à jour d'un voisin.

Ce protocole a donc pour caractéristique principale de vérifier la cohérence entre ses tables et la réalité.

6.4.2.4 Le protocole GSR

Le protocole GSR (Global State Routing [Chen and Gerla, 1998]) est similaire au protocole DSDV (voir §6.4.2.1). En effet, il est aussi basé sur les états de lien entre noeuds et tente d'en supprimer les inondations qui abaissent le rendement d'un tel protocole.

Les noeuds du réseau, lorsque ce protocole est mis en oeuvre utilisent :

- Une table de ses voisins

- Une table de topologie qui contient pour chaque destination l'état du lien qui les sépare et une estampille de l'information
- Une table des noeuds qui contient pour chaque destinataire l'identifiant des voisins à qui doit être envoyé le paquet.
- Une table de distance qui contient pour chaque noeuds destination la distance minimale.

Un noeud met à jour ses tables à chaque réception d'un message de routage et le retransmet à ses voisins. On peut, pour coder la logique de sélection de chemin, utiliser n'importe quel algorithme comme l'algorithme Dijksra (qui nécessite tout de même d'être adapté afin de mettre à jour toutes les tables).

6.4.2.5 Le protocole FSR

Comme son nom l'indique, le protocole Fisheye State Routing [Iwata et al., 1999] est basé sur la technique dite FishEye [Kleinrock and Stevens, 1971]. Il a été conçu dans le but de réduire le volume d'informations nécessaires pour représenter les données graphiques et consiste à garder une grande précision dans une zone proche de ce que l'on appelle le point focal. Plus on s'éloigne de ce point, moins on est précis.

Cette approche FishEye se traduit dans notre contexte, pour un noeud, par le maintien des données concernant la précision de la distance et la qualité du chemin pour les proches voisins. Plus un noeud est éloigné, plus les données possédées sur celui-ci seront floues.

6.4.2.6 Le protocole ZHLS

Le protocole ZHLS (Zone-Based Hierarchical Link State Routing [Joa-Ng and Lu, 1999]) est basé sur la décomposition d'un réseau en zone. Contrairement à la plupart des protocoles dit hiérarchiques, il n'y a pas ici de représentant pour chaque zone. La topologie d'un réseau est ainsi partagée en deux niveaux :

- Un niveau noeud permet de connaître les noeuds qui sont connectés (nature des liens...)
- Un niveau zone qui renseigne sur le schéma de connexion des différentes zones.

Ces niveaux différents entraînent donc deux différents types de liens : les liens inter-noeuds et les liens inter-zones.

Le réseau est donc décomposé comme l'illustre la figure 6.6.

Il résulte de cette décomposition un routage inter-zone et un routage intra-zone qui est permise par l'adressage mise en place et qui consiste en un identificateur de zone et un identificateur de noeuds et l'utilisation de LSP (Link State Packet) qui renseignent sur l'état des liens. On peut là aussi distinguer deux classes de LSP : ceux orientés noeuds pour lesquels un noeud donné contient des informations sur son voisin et ceux orientés zone qui sont, quant à elles, échangées de manière globale.

6.4.2.7 Le protocole CGSR

Le protocole CGSR (Clusterhead Gateway Switch Routing [Chiang et al., 1997]) est issu du protocole DSDV (voir en §6.4.2.1). Ce protocole rassemble chaque noeud du réseau en groupes contenant :

- Un représentant du groupe qui à pour voisins tous les noeuds du groupe
- Un ou plusieurs noeuds dits de liaisons qui sont commun à plusieurs groupes (à leur frontière)
- Aucun ou plusieurs noeuds membres du groupe mais n'ayant aucun statut particulier.

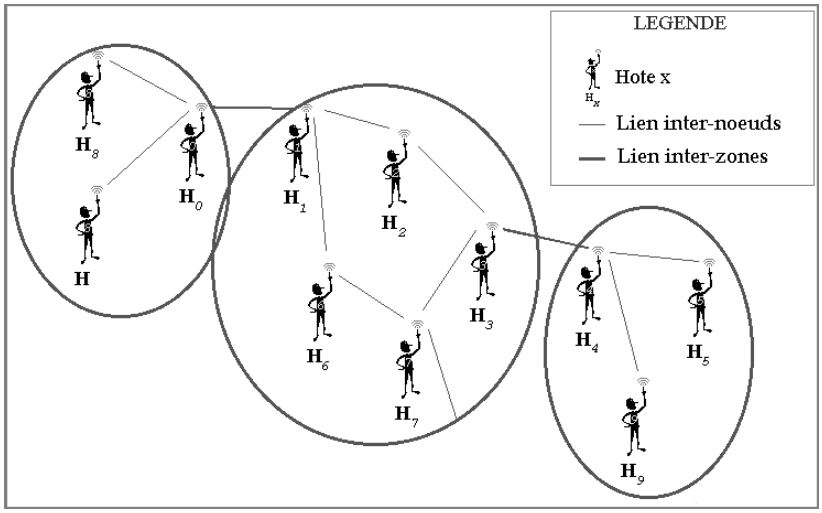


FIG. 6.6 – *Illustration du fonctionnement du protocole ZHLS*

Le réseau est ainsi décomposé comme l'illustre la figure 6.7.

Ce protocole est en fait la juxtaposition de deux protocoles :

1. Un protocole interne au groupe pour assurer une topologie en étoile,
2. Un protocole externe au groupe pour assurer la communication entre les groupes.

Les représentants se créent une représentation du réseau via un graphe de recouvrement. Ce graphe est en fait le squelette de l'architecture du réseau.

Chaque noeud possède une table indiquant les membres et groupes associés. Chaque noeud diffuse cette table périodiquement et met la sienne à jour en fonction de celles qu'elle reçoit de la même manière que le fait DSDV. C'est donc la fréquence d'émission de ces tables qui assure le degré d'adaptivité du réseau.

Ce routage est déterministe mais ne donne pas le chemin optimal que l'on pourrait espérer (dans notre cas si H0 veut joindre H6 la route proposée par le protocole sera (H0,H1,H2,H3,H7,H6) et non (H0,H1,H2,H4,H6)).

6.4.2.8 Le protocole HSR

Le protocole HSR (Hierarchical State Routing) utilise les notions de groupes dynamiques, niveaux hiérarchiques avec une gestion efficace de localisation tout comme pour le protocole CGSR vu précédemment en §6.4.2.7.

L'ensemble des noeuds est partitionné en un ensemble de groupes représentés par un élu mais il existe une récursion : les représentants des groupes dans un niveau n, deviennent des membres dans le niveau n + 1.

6.4.3 Les protocoles de routage hybrides

6.4.3.1 Le protocole TORA

Le protocole TORA (Temporary Ordering Routing Algorithm [Park and Corson, 1997]) a pour objectif principal de minimiser au mieux l'effet des changements de topologie. Sa méthode est de mémo-

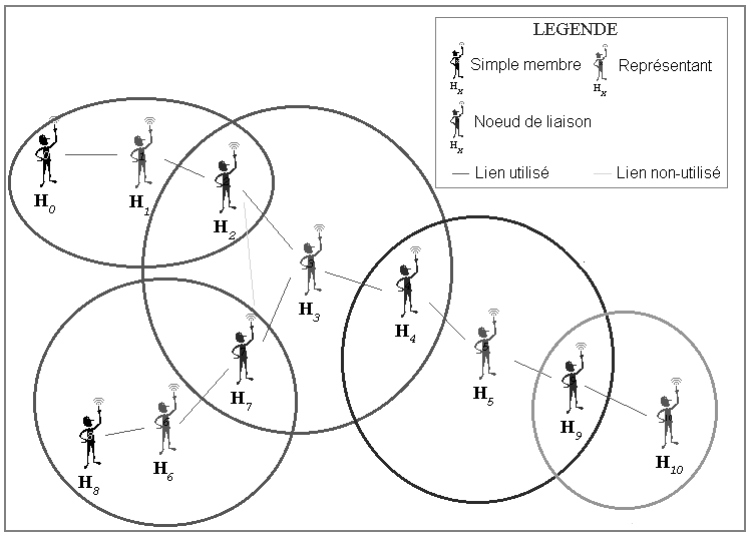


FIG. 6.7 – *Illustration du fonctionnement du protocole CGSR*

riser, pour chaque destination, plusieurs routes. Il en résulte donc une adaptabilité aux changements de topologie. Ce protocole ne privilégie donc pas les chemins les plus courts dans ce choix. A moins que tous les chemins qui mènent à la destination ne soient rompus, le protocole proposera toujours un chemin d'accès pour les informations.

Dans ce protocole, un noeud n'échange des paquets de contrôle qu'avec ses voisins, la gestion des routes passe par 4 primitives qui sont : la création de routes, la suppression de routes, la maintenance de routes et l'optimisation de routes. Ces fonctions permettent d'assurer la propriété *orientation destination*[16] des graphes acycliques orientés.

6.4.3.2 Le protocole ABR

Dans le protocole ABR (Associativity Based Routing [Toh, 1997]), les concepteurs ont mis l'accent sur la découverte de route à longue durée de vie. Pour cela ils se basent sur une métrique de routage qu'ils appellent degré de stabilité d'association.

Ce protocole fonctionne en trois phases :

- la découverte de routes : le noeud source envoie un message BQ (Broadcast Query) afin de trouver les noeuds qui mènent vers la destination. Le noeud destination choisit le chemin qu'il veut que la source prenne pour le contacter et répond par un BQ-REPLY.
- la reconstruction des routes gère les routes existantes et les met à jour. Pour cela les noeuds utilisent des LQ (Localisation Query) afin de vérifier qu'une route est toujours valide et des RN (Route Notification) pour éliminer les routes incorrectes.
- la suppression des routes est utilisée lorsqu'une route devient inutile.

Ce protocole privilégie donc les routes les moins perturbées dans le temps par la mobilité des noeuds.

16. Cette propriété est celle qui assure qu'une route existe pour chaque destination

6.4.3.3 Le protocole DREAM

Le protocole DREAM (Distance Routing Effect Algorithm for Mobility [Basagni et al., 1998]) est un protocole proactif mais procède tout de même à des inondations... partielles. Chaque noeud signale périodiquement sa position aux autres. La distance influe sur la fréquence de cet envoi, un peu comme pour FSR (voir §6.4.2.5).

Lors de l'envoi de données le noeud source va envoyer le message aux voisins qui sont dans la direction du noeud cible (la trame contiendra le chemin à suivre) ou procédera par inondation s'il ne possède pas ces informations ou si elles sont trop vieilles. La cible enverra un acquittement vers la source (à moins que le message lui soit parvenu par inondation) de manière similaire.

6.4.4 Conclusion

La revue des protocoles ad-hoc a permis d'identifier plusieurs techniques pour le routage. Quatre tendances se dégagent de cette revue.

Diffusion La méthode la plus simple pour s'assurer qu'un destinataire reçoive un message est de le diffuser via un algorithme d'inondation. Chaque noeud qui reçoit le message d'inondation pour la première fois le réémet pour ses voisins. Les hôtes du réseaux mémorisent généralement les couples (émetteur, numéro de message) identifier les messages. Ces protocoles sont très adaptatifs mais ont un coût très important en terme de volume émis. Plusieurs variantes d'algorithmes d'inondations sont proposées dans [Ni et al., 1999]. Le protocole DSR est l'exemple le plus significatif de cette approche.

Clusters L'utilisation des clusters pour le routage dans les réseaux ad-hoc est apparu tardivement dans la littérature [Gerla and Tsai, 1995, Krishna et al., 1995, Das et al., 1997]. Ces protocoles les clusters permettent de minimiser le volume de la recherche d'une route et le coût des changements de topologies. Cependant le coût pour la gestion du cluster et pour que chaque noeud maintiennent ses informations à jour est considérable [Jiang et al., 1998]. D'après ce même article, l'utilisation la plus simple de clusters figure dans [Gerla and Tsai, 1995]. Le protocole CGSR nous parait être le plus représentatif de cette approche.

Hiérarchique Avec cette approche, le réseau est décomposé en zones. Chacune de ces zones peut contenir plusieurs sous-domaines. Lors de l'émission d'un message, ce dernier est transmis a son supérieur hiérarchique. Ce schéma est répété tant que le destinataire n'est pas joignable. Le protocole FSR nous parait être le plus représentatif de cette approche.

Théorie des graphes Certaines approches sont basées sur la théories des graphes. Plusieurs approches sont alors possible :

- On peut utiliser des algorithmes de plus calcul de plus court chemin comme OLSR via l'algorithme de Dijkstra,
- On utilise la notion de groupe dominant. La difficulté et de s'assurer que cet ensemble est connexe [17] pour être sûr que la diffusion est complète. Le routage s'appuiera donc sur cet ensemble dominant pour joindre le destinataire. L'inconvénient de cette technique est qu'elle est peu adaptative. Une perspective d'utilisation pour le protocole DSR est proposée dans [Shah and Zelikovsky, 2004].

17. Un graphe est dit connexe si tout les noeuds qui le compose sont joignables

6.5 Les travaux SMA dans ce contexte

La nature distribuée et ouverte des réseaux sans fil rend l'approche multi-agents particulièrement adaptée. Le fait que cette approche fournisse une représentation externe des interactions et de l'organisation permet plusieurs fonctionnalités comme le contrôle par un observateur externe. Nous avons identifiés plusieurs domaines d'applications des systèmes multi-agents dans le contexte des réseaux sans fil : le routage de l'information, la fusion de données, la recherche et la description de services. Nous avons créé une autre catégories pour introduire des travaux connexes où les systèmes multi-agents sont utilisés.

Le routage de l'information. Peu de travaux, traitant de la problématique du routage de l'information dans les réseaux ad-hoc d'entités physiques autonomes, utilisent les systèmes multi-agents. On peut cependant citer les travaux relatés dans [Choudhury et al., 2000] utilisant des agents mobiles[18] et centrés sur l'optimisation des temps d'acheminements de messages. Leur technique repose sur la gestion des affinités entre agents (similaire aux techniques de routage dites par état des liens). Le projet militaire ActComm [Gray, 2000] pour lequel le routage de l'information est essentiel : l'objectif est de gérer les communications qu'échangent les soldats entre eux et le quartier général via un satellite.
D'autres approches de l'intelligence artificielle distribuée connexes aux systèmes multi-agents sont utilisées. Dans [Arabshahi et al., 2001] des algorithmes sont décris pour le routage d'information reposant sur l'intelligence Swarm.

La recherche et la description de services. La recherche et la description de services est un domaine de recherche très actif où les systèmes multi-agents sont utilisés dans le contexte des réseaux ad-hoc [Chen et al., 2000, Langley, 2003, Ratsimor et al., 2002, Campo et al., 2002]. Dans ce cadre FIPA, qui travaille sur la normalisation des interactions dans les systèmes multi-agents, s'est d'ailleurs intéressé au cas de cette famille de réseau de communication [Berger et al., 2003]. Certains travaux sur ce standard on été réalisés [FIPA, 2002a, FIPA, 2002b] et des résultats intéressants on été mesurés pour certains types de réseaux ad-hoc [Helin and Laukkanen, 2002].
Dans [Minar et al., 1999] un outil pour aider à la conception et au management de réseaux sans fil. Ces travaux concernent la gestion de services en utilisant des agents mobiles selon l'idée d'un écosystème où les agents évolueraient dans l'environnement que représente le réseau. Le problème décrit dans [Zhang et al., 2002] est similaire au notre mais leur approche utilise les algorithmes stochastiques distribués.
Dans [Campo et al., 2002] les systèmes multi-agents proposent une infrastructure multi-agents pour proposer un service type pages jaunes au système.

La fusion de données. Le projet Unmanned Ground Vehicle Program ARPA [Cook et al., 1996] utilise l'approche multi-agents pour fusionner des données provenant de plusieurs véhicules militaires afin d'avoir une vue globale du champ d'opération à partir des différentes vues locales. Plus récemment, Petriu et al [Petriu et al., 2002] présente un projet dans lequel les agents sont utilisés pour réaliser une structure ouverte et flexible d'assemblage de cellules dans laquelle les données collectées par les agents sont fusionnées pour créer un environnement virtuel.

18. les agents mobiles sont des agents qui se déplacent d'unité de calcul en unité de calcul. Ils encapsulent leur code, données et autres objectifs globaux.

Autres travaux. Dans [Sansores and Pavón, 2004], il est proposé une architecture en trois couches (couche applicative, couche ontologie et une couche de communication) pour minimiser l'impact des déconnexions. Une implémentation tourne sur PDA et utilise des modules Bluetooth. Elle permet la réservation de tickets de cinéma et places de théâtre via une connexion internet.

6.6 Conclusion

Dans ce chapitre, nous avons présenté le domaine des réseaux sans fil en focalisant sur les réseaux ad-hoc. Cette présentation des réseaux ad-hoc permet de les situer dans l'évolution des réseaux sans fil et insiste sur le routage des messages dans de tels réseaux. Les principes fondamentaux des techniques de routage sont exposés afin de rendre plus compréhensible la partie qui traite des différents protocoles représentatifs de ce domaine.

Cette revue des protocoles est organisée selon le critère de la réactivité/pro-activité. Ces protocoles reposent généralement sur l'inondation. Les protocoles réactifs ne maintiennent pas de tables de routage. Si ils en utilisent, ce n'est que de façon temporaire. Ils sont les plus adaptatifs mais aussi ceux qui ont la moins bonne efficacité. Les protocoles pro-actifs utilisent des tables de routage qu'ils maintiennent pour être plus efficaces mais leur degré d'adaptabilité est fonction de la fréquence d'envoi de message dont le rôle est d'assurer la cohérence des données.

Cette revue met en évidence quatre techniques exploitées par les protocoles. Ces techniques sont basées sur différentes approches qui sont les approches par diffusion, les approches par clusters, les approches par hiérarchies et les approches basées sur la théorie des graphes.

L'idée qui semble la plus prometteuse est celle qui consiste à regrouper les agents en clusters ou en groupes. Le degré d'adaptabilité est souvent fonction de la fréquence d'envoi des messages de configuration. Un inconvénient majeur de ces techniques est la non optimalité des routes que prennent les messages. La formation d'un groupe ne dépend généralement que du critère de localité géographique.

Chapitre 7

Le maintien de l'intégrité fonctionnelle

Informatique : Alliance d'une science inexacte et d'une activité humaine faillible.

Luc Fayard

Ce chapitre présente le maintien d'intégrité fonctionnelle. Dans une première partie, nous définissons le maintien, le traitement classique des erreurs dans les réseaux et des critères qui permettent d'évaluer la qualité du maintien de l'intégrité fonctionnelle. Nous introduisons dans une deuxième partie les techniques basées sur la réplication de services. Enfin dans une troisième partie nous présentons l'auto-organisation comme moyen d'assurer l'adaptation. Nous insistons particulièrement sur les moyens de mettre en oeuvre une auto-organisation (les mécanismes).

7.1 Généralité sur le maintien d'intégrité fonctionnelle

7.1.1 Définitions

Un système multi-agents est conçu pour satisfaire un ou plusieurs objectifs. Toute perturbation (faute ou panne) qui amène le système à ne pouvoir accomplir ces buts nuit donc au bon fonctionnement de ce système : elle a violé son intégrité fonctionnelle, sa cohérence.

> Nous utiliserons indifféremment, dans la suite du manuscrit, le terme *faute*, qui fait référence à un manquement aux règles, et *erreur*, qui est une faute commise par méprise.

> Nous utiliserons le terme de *panne* pour désigner un arrêt de fonctionnement temporaire ou définitif. Il est à noter que d'une panne peut découler des erreurs.

Un système maintient donc son intégrité fonctionnelle s'il est en mesure d'effectuer toutes les tâches que lui a imposées son concepteur et ce indépendamment des changements qu'il peut subir (internes ou externes).

> Nous appellerons *maintien d'intégrité fonctionnelle* l'ensemble de mécanismes qui aident à garantir le bon fonctionnement du système construit, afin qu'il soit conforme aux spécifications de l'utilisateur et pérenne en cours d'exécution (d'après [Equipe MAGMA, 2002].

Il apparaît donc clairement que le maintien de l'intégrité fonctionnelle est un problème inhérent aux systèmes multi-agents ouverts. En effet, les agents, qui sont les éléments actifs travaillant (en collaboration ou non) à l'accomplissement des objectifs du système multi-agents, sont supposés dans le contexte d'un système ouvert, pouvoir le quitter librement. Si les services qu'ils offraient ne sont connus d'aucun autre agent alors le système multi-agents peut devenir défaillant. A l'inverse on pourrait imaginer que, dans un système, l'arrivée trop massive d'agents désorganiserait le système multi-agents et nuirait à son intégrité fonctionnelle.

7.1.2 Le traitement d'erreurs

Maintenir l'intégrité fonctionnelle d'un système, qu'il soit ou non multi-agents, revient à traiter les erreurs qui apparaissent (à moins de réussir à les empêcher). Le traitement d'une faute se décompose en quatre étapes [Pujolle, 2000] :

- La signalisation du fonctionnement anormal
  Il existe trois outils pour détecter un mauvais fonctionnement :
  - les messages d'erreurs : le composant qui fonctionne mal en est "conscient" et le signale aux autres,
  - les tests : le système vérifie son état en procédant à des tests de sécurité (on vérifie un maximum d'éléments en un minimum de temps) ou de diagnostic (on teste les composants un à un),
  - les seuils : le système possède des seuils qui déclenchent automatiquement des alarmes s'ils sont dépassés.

  Pour ce qui est des systèmes multi-agents, il est envisageable dans certains cas d'utiliser des agents diagnostic (du type de ceux abordés dans [Schroeder et al., 1996]) ou d'utiliser des agents sentinelles [Haegg, 1997]. Dans notre contexte, ces agents espionneraient toutes les communications à la recherche d'erreurs. Cette approche n'est pas efficace lorsque les communications sont volumineuses.
- La localisation du défaut qui consiste à procéder à une analyse plus poussée des tests qui ont amené à détecter un défaut, voire à en utiliser d'autres.
- La réparation : suivant la nature de la panne on peut la réparer, dans le cas de problèmes logiciels, en réinitialisant et en rechargeant certaines parties de l'application.
- La confirmation du retour à l'état normal : Il s'agit en fait de confirmer le fait que la panne a été traitée, voire de procéder à un test de vérification.

7.1.3 Recherche de critères d'évaluations

Nous pouvons tirer de ces causes de perturbation des critères qui permettent d'apprécier la qualité d'un maintien de l'intégrité fonctionnelle. Ces critères sont:

- La complexité de la méthode de maintien de cohérence.
  Il est nécessaire de pouvoir quantifier la complexité d'une méthode de maintien d'intégrité fonctionnelle. En effet, si une méthode demande par exemple de procéder à de longs calculs, cela aura des répercussions sur la puissance de calcul qu'il faudra affecter aux stations, cela peut induire une consommation d'énergie importante etc.
- Le caractère prédictif / curatif.

On peut donc imaginer plusieurs types de solutions pour le maintien d'intégrité fonctionnelle :

- une solution prédictive : on veille à toujours empêcher une violation de la cohérence du système,
- une solution curative: après constat de la perte de l'intégrité fonctionnelle, on travaille pour revenir dans un état cohérent. Cette solution implique de pouvoir faire des diagnostics.

- Le nombre de stations. Le nombre d'éléments maximum supporté par une méthode est un critère important. En effet, on pourrait imaginer des solutions qui seraient efficaces pour un nombre limité de stations et complètement inintéressantes dans un contexte différent.
- L'efficacité. En effet, il ne faut pas que le temps qui sépare l'envoi d'un message de sa réception soit trop important (bien que cela dépend en fait de l'application que l'on souhaite en faire). On peut aussi estimer l'efficacité d'un système en calculant le ratio entre le nombre des messages utiles pour le transport des informations sur le nombre total des messages envoyés (incluant donc les trames utilisées pour la configuration du réseau).

7.2 Méthodes classiques de maintien d'intégrité fonctionnelle

Il existe plusieurs méthodes pour maintenir une tolérance aux pannes et plus généralement l'intégrité fonctionnelle d'un système. Nous ne présenterons pas les techniques utilisés pour l'intégrité des données (brouillage, chiffrement et utilisation de droits d'accès) mais nous nous concentrons sur le maintien d'intégrité d'un système dans sa globalité.

7.2.1 L'utilisation de points de reprise

Cette technique de maintien d'intégrité procède à des sauvegardes régulières de l'état d'un système sur des supports fiables et stables. Les algorithmes sont basés sur deux différentes approches :

- Approche par *coordination* pour laquelle on sauvegarde l'état global du système. Son inconvénient est que la pose de points de reprise est coûteux et que le recouvrement est lent,
- Approche reposant sur des *points de reprise indépendants* pour lesquels les composants du systèmes procèdent eux mêmes à leur pose de points de recouvrement. L'inconvénient de ces techniques est le coût des communications pour assurer et synchroniser les sauvegardes (généralement les recouvrements sont confinés).

Un exemple d'utilisation de points de reprise dans le contexte d'un système à mémoire partagée figure dans [Bronevetsky et al., 2004].

D'une manière générale, ces techniques ne sont pas adaptées au passage à l'échelle.

7.2.2 La réplication de services

Cette technique, qui peut s'appliquer aux systèmes distribués pour améliorer la sûreté de fonctionnement d'un système ainsi que pour améliorer ses performances, est très utilisée. Elle consiste à multiplier les services, à en faire plusieurs copies. L'idée est qu'un service souvent sollicité, soit disponible sur plusieurs " processeurs ". Un autre avantage de cette méthode est de rapprocher les services des zones géographiques où ils sont le plus demandés afin que la proportion des accès locaux augmente. Cela induit une diminution des communications inter-machines voire inter-réseaux.

Cependant, cette méthode fait apparaître un autre problème : il faut assurer le maintien de la cohérence des copies du service car elles ne partagent pas de mémoire commune. Il existe trois types de

réplications qui se différencient par les ressources consommées, leurs performances et leur tolérance aux fautes inter-copies.

La réplication passive Avec ce type de réplication, c'est une seule copie, que l'on appelle *primaire* ou *coordinateur*, qui reçoit la requête d'un client et l'exécute. Cette copie diffuse régulièrement son nouvel état aux autres copies, appelées secondaires ou sauvegardes, afin d'assurer la cohérence. Ces copies permettent d'avoir des points de reprise. En effet, si la copie primaire est défaillante alors une des copies secondaires, choisie par un protocole dit d'*élection*, prend le relais. Si la copie était différente elle corrigerait son état grâce à un protocole dit de *réplication* comme Gina [Berlage and Genau, 1993], Bayou [Terry et al., 1995], IceCube [Shapiro et al., 2001].

Cette méthode a été implémentée dans DarX [Zimmermann, 1999] qui est une plate-forme, transparente et modulaire, tolérante aux pannes distribuées.

La réplication active Avec ce type de réplication, toutes les copies reçoivent les requêtes clients, les exécutent et y répondent. Cette méthode ne nécessite donc pas de point de reprise, de protocole d'élection ou de réplication. Cependant, les requêtes ne sont pas déterministes: il faut donc un mécanisme de collecte des réponses ainsi qu'un mécanisme permettant de choisir celle à donner.

Cette méthode est implémentée dans DarX. Elle est aussi utilisée par IBM [Pleisch and Schiper, 2001] dans le cadre d'études sur les agents mobiles notamment pour gérer des clusters dans les middlewares [Goldberg et al., 2001].

La réplication semi-active Cette réplication est active car chaque copie exécute la requête mais, par contre, les sources d'indéterminisme sont résolues par le choix d'une copie primaire qui diffuse aux autres copies, ses synchronisations. C'est elle qui fournira le résultat au client. Le projet JuxMem [Antoniu et al., 2004], dont l'objectif est le partage de services transparents utilise la réplication semi-active du rôle de gestionnaire des informations et la réplication active pour les données manipulées sur le réseau.

Les techniques de réplication présentées ici sont les plus courantes. Des travaux sur d'autres types de réplication existent comme la réplication coordinateur-cohorte [Birman, 1985] ou la réplication semi-passive [Défago et al., 1998]. Cependant ces méthodes présentent toujours des inconvénients, dont le majeur est la surcharge en communications.

7.3 Méthodes basées sur l'auto-organisation

Les systèmes multi-agents sont, comme nous l'avons vu en §2.2, composés de nombreux agents qui interagissent. Il en résulte évidemment une forte dynamique, qui sera d'autant plus accentuée si le système est ouvert. Il est donc impossible de prévoir, pour cette famille de systèmes multi-agents, tous les cas de figure qui vont se présenter : il est nécessaire que le système s'auto-organise. Ainsi l'auto-organisation pourrait permettre d'être tolérant aux pannes.

Ce que l'on entend par organisation ayant été précisé en §2.2.3.4, nous pouvons commencer par donner un sens à l'auto-organisation dans les systèmes d'intelligence artificielle. Suite à cela, nous nous intéresserons à l'importance de l'émergence dans les Systèmes Multi-Agents Auto-Organisé (SMAOA) puis, pour finir, nous établirons un inventaire des principaux mécanismes d'auto-organisation existants.

7.3.1 L'auto-organisation dans les systèmes d'intelligence artificielle.

> L'*auto-organisation* est la capacité d'un système à s'organiser seul, à partir des interactions entre les entités qui le compose.

Tout comme dans [Groupe MARCIA, 1996] nous différencierons l'aspect statique de l'organisation en la nommant structure (pour structure organisationnelle) et l'aspect dynamique en l'appelant organisation.

L'auto-organisation est étudiée dans plusieurs disciplines depuis les années 1950 [Heylighen, 1999]. Les domaines pour lesquels on étudie l'auto-organisation sont, entre autres, la biologie (exemple des insectes), la thermodynamique (exemple de la formation de cristaux symétriques dans une solution liquide) et la cybernétique (exemple des réseaux de neurones).

Les systèmes multi-agents auto-organisés Pour concevoir un système multi-agents et ainsi résoudre un problème de manière distribuée il est nécessaire de se poser des questions quand à l'organisation. Dans le cas, par exemple, d'un système multi-agents ouvert, il va être nécessaire de lui donner les moyens d'*adapter* son organisation afin de satisfaire au mieux ses objectifs. En effet, si par exemple un agent qui avait une tâche bien précise venait à quitter l'organisation, il faut trouver quelqu'un pour le remplacer ou assumer en plus de ses propres tâches celle qu'occupait l'agent. Par adapter on entend agir sur les agents ou la structure organisationnelle afin de réorganiser le travail [Foisel et al., 1996].

Cette réorganisation sera appelée auto-organisation si elle est prise en charge par les membres de l'organisation. Elle émerge donc des interactions entre les composants de cette organisation (ce point est approfondi en §7.3.2) et est étroitement lié à l'activité de l'environnement.

Il est maintenant acquis que pour les systèmes multi-agents utilisant des techniques d'auto-organisation, l'adaptation est le but premier [Ishida and Yokoo, 1992, So and Durfee, 1993] dont des travaux plus récents [Piquemal-Baluard and Glize, 1996, Camps, 1998] précisent qu'il s'agit d'un changement de topologie c'est-à-dire des connexions du réseau d'agents.

Les systèmes multi-agents et les systèmes critiques auto-organisés Les systèmes critiques auto-organisés (SCAO) sont des systèmes pour lesquels les états critiques dynamisent l'émergence des comportements [Marcenac, 1997a]. Le système est donc très dynamique et ce sont les interactions qui positionnent cet état critique et qui font donc qu'un comportement émerge.

Les SCAO peuvent être modélisés par des systèmes multi-agents comme le montrent certaines expérimentations [Cerri and Loia, 1997]. Il faut que ces modèles permettent de représenter les composants qui interagissent et évoluent dans le temps ainsi que le comportement qui émerge de ces interactions entre composants [Marcenac, 1997a, Marcenac, 1997b].

7.3.2 L'émergence de structure au centre de l'auto-organisation

Comme nous l'avons vu précédemment, les systèmes auto-organisés et les systèmes critiques auto-organisés font émerger des phénomènes issus des interactions entre les divers composants du système (éventuellement un système multi-agents auto-organisé). Pierre Marcenac [Marcenac, 1997a] interprète cette émergence en distinguant deux de ses aspects : la forme structurelle (détaillée en §7.3.2) et ses caractéristiques (développée en §7.3.2).

Emergence de structures Cette émergence est ce qui fait d'une organisation une auto-organisation. Elle est produite à partir des interactions entre les composants du système. En effet, comme l'illustrent de nombreux travaux [Drogoul, 1993, Marcelpoil et al., 1994], des phénomènes non programmés et non présents à un instant $t = 0$ naissent de la confrontation d'un environnement actif et d'une activité structurée entre les différents éléments qui composent un système. Le degré de complexité de cette structure émergente va bien au-delà de celle de ses composants et augmente au fur et à mesure que le système s'auto-organise.

Emergence de propriétés L'émergence de propriétés fait référence à ce qui concerne la *rétro-propagation*. En effet, une auto-organisation se doit d'intégrer des mécanismes de régulation afin de veiller à la cohérence du système. Le système agit ainsi comme une commande adaptative (confirmant ainsi le lien entre l'auto-organisation et l'auto-adaptation évoqué en §7.3.1).
La rétro-propagation, de par les mécanismes de régulation qui lui sont inhérents, assure une meilleure stabilité du système. Les propriétés, qui émergent du système, agissent donc comme des contraintes modulant le comportement de chacun des composants.

7.3.3 L'évaluation des auto-organisations

Il est important lorsque l'on s'intéresse aux systèmes auto-organisateurs de penser à la problématique de l'évaluation des résultats. En effet, comme nous l'avons constaté précédemment, les systèmes auto-organisés sont par essence même tolérants aux fautes et donc particulièrement indiqués lorsque l'on ne peut pas prévoir toutes les situations qu'ils vont être appelés à rencontrer. Il est donc difficile d'évaluer une méthode d'auto-organisation dont la force est justement de s'adapter à des situations imprévues.
Nous allons donc, dans cette partie, donner des éléments qui serviront de base aux évaluations des systèmes auto-organisés. Pour cela, nous allons nous intéresser dans un premier temps à l'observabilité de ces systèmes puis, dans un second temps, à l'analyse de ces observations.

7.3.3.1 L'observation des auto-organisations

Les niveaux d'observations d'un système multi-agents. On peut distinguer trois niveaux d'observation pour une organisation d'agents [PLEIAD, 1992]:

- Un niveau externe au système multi-agents (cas n°1 de la figure 7.1) si on considère le système comme une boite noire dont on ne peut observer que les entrées et les sorties.
- Un niveau interne au système multi-agents (cas n°2 de la figure 7.1) si on prend comme objet d'étude les interactions entre les éléments de la société d'agent. L'observateur serait ainsi plongé dans le système et regarderait ce qui se passe entre ses différents membres.
- Un niveau interne à l'agent (cas n°3 de la figure 7.1) si on prend comme objet d'étude les membres de la société et plus précisément leurs architectures.

Comme nous l'avons vu en §7.3.2 l'auto-organisation émerge des interactions entre les composants du système. C'est donc le niveau d'observation *interne au système multi-agents* qui va nous intéresser par la suite.

Que faut-il observer? Dans les systèmes multi-agents auto-organisés les états stables du processus d'auto-organisation se traduisent par l'émergence de structures. Ce qu'il faut donc observer dans le systèmes, ce sont ces structures de reconnaissance.

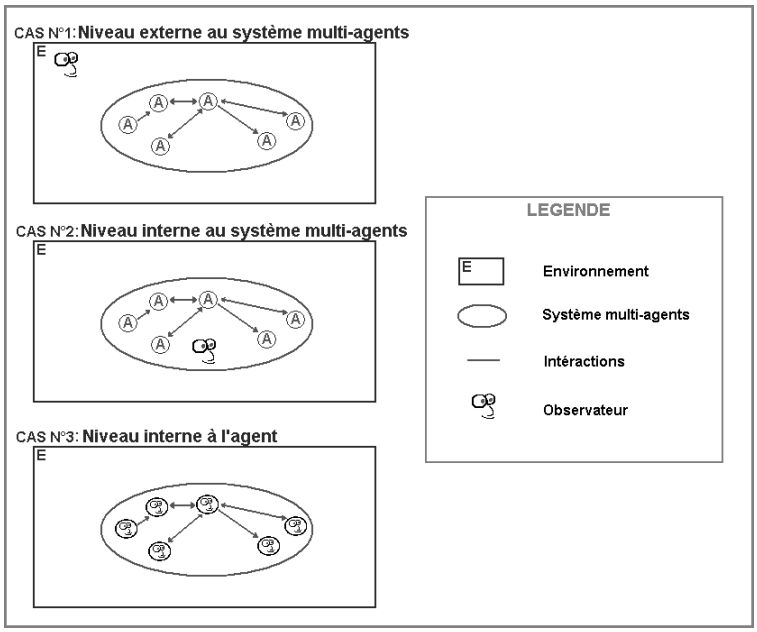


FIG. 7.1 – *Différents types d'observation*

7.3.3.2 L'analyse des résultats observés

L'évaluation sera faite sous les deux aspects proposés dans [Groupe MARCIA, 1996]. En effet, dans un premier temps, nous aurons pour but d'évaluer l'auto-organisation sans prendre en compte l'application que l'on souhaite en faire. Enfin, dans un second temps, nous nous intéresserons à l'application et prendrons donc en compte l'adéquation entre le domaine applicatif et le résultat du processus d'auto-organisation.

Analyse en terme d'auto-organisations. D'une manière générale, pour qu'un processus d'auto-organisation soit intéressant et exploitable, il doit être à la fois stable, sensible et convergent:

- Stable : la stabilité d'un système fait référence à une constance de celle-ci dans le temps. Le système doit donc être stable afin de mettre en évidence des structures persistantes qui ne se modifieraient que sous l'influence de perturbations (internes ou externes).
- Sensible : la sensibilité du processus d'auto-organisation fait référence à sa capacité à faire évoluer une structure vers un état de reconnaissance (partiel ou total) selon que tous les critères retenus aient été satisfaits ou non. Il faut donc que le processus d'auto-organisation soit sensible afin qu'il soit possible que les structures qui émergent soient remises en cause.
- Convergent : la convergence d'un système reflète sa capacité à évoluer vers des structures connues. Le système doit donc être convergent afin que l'on aboutisse à de nouvelles structures.

Analyse en terme de résultats. Comme nous l'avons vu précédemment, cette partie de l'analyse consiste à quantifier l'adéquation entre le processus d'auto-organisation conçu et les résultats qui étaient espérés. Il nous faudra donc déterminer si le système est *valide* et/ou *pertinent*. Ces deux propriétés seront corrélées avec *l'intérêt* et la *simplicité*.

Explicitons ces termes:

– Le système est dit *valide* si le résultat produit par le processus auto-organisé est conforme avec ce que l'on attendait. L'*intérêt* est une grandeur qui estime la variation de cette *validité* en fonction de la complexité d'un raisonnement.
– Le système est dit *pertinent* si la structure peut être considérée comme simple pour ce qui est de la mise en oeuvre et de la compréhension qu'en aurait un observateur. La *simplicité*, à l'image de l'*intérêt*, est une grandeur qui estime la variation de la *pertinence* en fonction de la complexité du raisonnement.

Les perturbations. Il peut être intéressant d'injecter dans le système des éléments perturbateurs afin de créer des dysfonctionnements et ainsi nous permettre de mesurer son degré d'auto-adaptation et d'auto-organisation. Cela nous permettrait de déterminer ses facultés à atteindre des états stables et ses performances.

7.3.4 Mécanismes d'auto-organisation

Ces mécanismes décrivent les différentes étapes de l'auto-organisation. Nous commencerons notre étude en donnant quelques généralités communément admises puis nous nous intéresserons à différentes techniques d'auto-organisation.
Les auto-organisations sont articulées autour de trois points [Calderoni et al., 1997] qui sont:

– Un mécanisme qui permet de détecter les meilleurs conditions d'apparition du phénomène d'auto-organisation, de décrire la sensibilité des différents paramètres qui l'influent. Cette tâche incombe aux différents composants de ces auto-organisations car elle est répartie sur ces unités. Cette opération consiste donc, au niveau de l'organisation, à trouver les agents qui sont dans un état dit instable (l'agent ne sera stable que si ses aspirations, fixées par le rôle qu'il tient dans l'organisation, sont satisfaites). Les agents ayant en commun certaines caractéristiques sont regroupés et signalent, aux autres agents de la société, par un message (dit de *recomposition*) qu'ils peuvent participer au phénomène émergeant.
– L'apparition du phénomène : ce mécanisme est chargé d'agréger les agents et d'assurer la persistance de la structure. L'interprétation des messages de recomposition conduira à la constitution de la société, l'établissement des relations entre les agents et les groupes ainsi créés.
– L'arrêt du développement du phénomène nécessite de repérer les agents qui sont stables; il est donc nécessaire de procéder à des observations (§7.3.3.1).

Décrivons maintenant, les principales techniques d'auto-organisations que nous avons pu identifier.

Auto-organisation avec des compétences réflexes Dans ce type d'auto-organisation, les agents (qui sont réactifs) adaptent le seuil de déclenchement des actions en fonction de leur perception de l'environnement. Chaque membre de l'organisation devient alors de plus en plus spécialisé.

Le phénomène de spécialisation est bien visible dans les travaux de Marcelpoil, qui illustre bien la spécialisation d'agents "cellule" en fonction des ressources disponibles [Marcelpoil et al., 1994] et dans les travaux d'Alexis Drogoul sur les fourmis [Drogoul, 1993].

Auto-organisation par contrôle des interactions L'auto-organisation de ce type s'applique elle aussi sur des agents réactifs et consiste en des mécanismes d'ajustement des schémas d'interaction. C'est le cas dans PACO (PAtterns de COordination [Demazeau, 1993]) où les interactions ne dépendent pas d'un stimulus, comme dans le cas précédent, mais intègrent la perception de l'environnement ainsi que les interactions avec les autres agents.

Le W-learning et le Q-learning Le mécanisme *W-learning* créé par Humphrys [Humphrys, 1995a] permet aux agents (des robots autonomes) d'apprendre les actions à déclencher en fonction de ce qu'ils perçoivent de leur environnements. Ils agissent toujours avec l'action dont la résultante aura la plus grande importance. Un retour d'état permet à l'agent d'ajuster l'importance associée à l'action ainsi que sa qualité (par l'algorithme du Q-learning [Watkins, 1989, Humphrys, 1995b]).

Auto-organisation avec liens préférentiels Il peut être nécessaire, dans certains systèmes multi-agents, que les agents se connaissent entre-eux. Cela permet de choisir l'agent avec lequel on souhaite interagir suivant les services qu'ils proposent. Les interactions précédentes peuvent l'aider à faire ce choix. On peut entre autres jouer sur les relations de préférence fixées selon des critères très variés tel que le temps de réponse à une requête.

Les travaux de J. Bollen [Bollen and Heylighen, 1996] sont aussi inclus dans cette catégorie. Leur mécanisme utilise, pour augmenter l'efficacité d'internet, des fonctions d'apprentissage. En effet, il détermine les liens les plus fréquemment utilisés par les internautes en associant à chaque arc un poids. Ces arcs feront que ces liens hypertexte seront le plus souvent proposés aux utilisateurs. Les travaux de Ok-Ki Lee et Steve Benford utilisent un principe aussi similaire dans le contexte des fédérations de courtiers [Lee and Benford, 1995]. Les arcs sont les liens entre les courtiers et le poids de l'arc correspond à l'affinité (adéquation entre les intérêts du courtier et les services offerts par l'autre).

Les travaux de Rémy Foisel [Foisel et al., 1996] vont aussi dans ce sens. En effet, les agents tirent une certaine expérience des interactions précédentes et déterminent ainsi les agents avec lesquels ils préfèrent travailler par la suite.

La relaxation La relaxation s'inscrit dans les mécanismes pour auto-organisations considérées comme étant un acte d'apprentissage collectif. La connaissance mise en jeu passe par les interactions qui seront mémorisées et réutilisées [Camps and Gleizes, 1995]. Selon qu'un agent est intéressé, ou non, par le contenu des messages il peut le mémoriser. Par la suite, il choisira de procéder ou non à des activités de relaxation, c'est-à-dire retransmettre le message vers des agents qui pourront satisfaire la requête ou être intéressés par le contenu de ce message. Lors de cette opération il prendra bien soit de laisser l'identifiant de l'agent d'origine. Cependant s'il juge l'information qu'il a reçu inutile ou fausse, il peut la détruire. Le mécanisme de base fut proposé par Les Gasser (paradigme de la relaxation) et repris par Valérie Camps [Camps, 1995].

Auto-organisation et réflexivité Cette auto-organisation s'appuie sur le fait qu'un agent peut avoir besoin d'organiser un groupe d'autres individus afin de pouvoir accomplir une tâche. Dans ce cas, il joue un double rôle car il doit:

- Choisir une organisation à imposer à des agents hiéarchiquement inférieurs.
- Faire émerger une organisation après avoir contacté et négocié avec des agents du même niveau organisationnel que lui afin de partager le travail.

Ainsi on génère d'autres organisations et ce de manière réflexive : les liens de coopération entre les agents sont remplacés par des liens de coopérations entre organisations collectives. Les agents, comme le cite [Groupe MARCIA, 1996], "s'organisent pour s'organiser".

Auto-organisation par modification de l'attribution de rôles Ce mécanisme joue sur les rôles des agents : une certaine hiérarchie se construit à partir des messages échangés par les agents. C'est le cas du réseau contractuel [Smith, 1980] ou un agent joue le rôle de contractant et de contracté sur différents contrats. Ce protocole (car il s'agit bien d'un protocole) permet aux agents de s'organiser ainsi en de petites sociétés d'agents liés par des relations contractant/contracté.

Auto-organisation par instanciation de structures organisationnelles Ce mécanisme est orienté sur la sélection d'organisations connues à priori. C'est le cas, par exemple, pour les travaux d'Emmanuelle Le Strugeon [Le Strugeon et al., 1993] qui a conçu des systèmes multi-agents où les agents déterminent, parmi plusieurs modèles d'organisations connues, ceux qui semblent le mieux convenir à la situation auxquels ils font face puis l'appliquent. Les travaux de Young-pa So et Edmund H. Durfee [So and Durfee, 1993, So and Durfee, 1996] vont aussi dans ce sens. En effet, le système étudie les propriétés de plusieurs structures organisationnelles et détermine celle qu'il convient d'appliquer suivant les critères que la situation rencontrée entraîne.

Auto-organisation avec introspection Les mécanismes d'introspection ont lieu d'être dans le cadre d'action centralisée et qualifient le fait qu'une entité s'observe elle même. On procède par exemple à l'enregistrement des traces d'inférence pendant la résolution d'un problème et on tire des conclusions pour améliorer l'organisation. Cela suppose l'existence de deux niveaux en correspondance: un niveau de base et un méta-niveau. Les opérations associées sont la *réification*, qui permet de passer du niveau de base au méta-niveau, et la *dénotation* qui est sa duale. Ces deux opérations permettent de qualifier un système de réflexif [Pitrat, 1990, Kornman, 1993].

Un agent peut faire appel à deux types d'introspection:

- L'introspection physique qui permet de vérifier l'intégrité fonctionnelle d'un système multi-agents en terme de répartition de la charge de travail, de la réduction des coûts de communications etc.
- L'introspection cognitive qui permet de quantifier les utilisations de certains services que l'agent propose, la charge de travail en tâche etc.

Ces informations, conjuguées, permettent à l'agent de savoir ou non s'il doit réorganiser ses compétences.

Auto-organisation par partage des connaissances Dans [Ishida et al., 1990, Ishida and Yokoo, 1992], l'application de l'auto-organisation dans un système composé de plusieurs résolveurs de problèmes est abordée. Ces résolveurs peuvent être considérés comme étant des systèmes de production dont certaines règles sont dépendantes et interfèrent. Il peut donc être nécessaire que les agents se synchronisent pour maintenir la cohérence de leurs données. L'auto-organisation s'appuie sur la recomposition des connaissances des agents: les agents se décomposent lorsque leur charge de travail devient trop importante. Il en résulte une paraléllisation des tâches et donc une amélioration des performances du système d'autant plus qu'il coopère avec les autres agents pour libérer les ressources matériel utilisées.

Mécanismes hybrides Certains travaux composent avec plusieurs de ces mécanismes. C'est le cas de Guichard [Guichard, 1996] qui compose les mécanismes de partage de connaissances et d'introspection où le phénomène de décomposition, vu précédemment, intervient lorsque le résultat de l'application, par

un agent, du mécanisme d'introspection est qu'il a besoin d'aide (surcharge trop importante de travail, incompétence, contrainte de temps trop sévère).

7.4 Conclusion

Le maintien d'intégrité fonctionnelle sera un point critique de notre modèle pour assurer une gestion adaptative des communications. La réplication de service ne répond pas à notre problématique qui relève plus d'un problème d'infrastructure que d'une nécessité d'avoir à disposition des services répliquables.

Notre conviction est que l'auto-organisation peut nous permettre d'assurer une communication fiable entre les agents. Le choix d'un mécanisme d'auto-organisation dépend de la nature de l'organisation. Nous avons identifié dix différents mécanismes (hors hybridation). Nous pensons qu'elle pourra nous faire économiser de l'énergie et ce malgré la dépense énergétique que peut engendrer sa mise en place.

Chapitre 8

Une architecture SMA pour la gestion des communications sans fil

Dans cette partie nous proposons un modèle pour assurer une gestion adaptative des communications dans le cadre des réseaux sans fil soumis à des contraintes énergétiques. Ce modèle définit une infrastructure multi-agents et est appelée MWAC (Multi-Wireless-Agent Communication). Une évaluation de ce modèle est proposée en comparaison avec un protocole représentatif des réseaux ad-hoc. En fin de chapitre nous présenterons les différentes déclinaisons de notre intergiciel (qui repose sur le modèle MWAC).

8.1 Avant-propos

Comme vu précédemment, les entités de nos systèmes sont à la fois matérielles et logicielles. Elles sont chargées de tâches complexes et de nature très diverse : tâches d'acquisition de mesures, d'action, de comportement, de calcul, de communication. Ces entités sont généralement reliées par des réseaux de communication sans fil. Dans de tels réseaux les liens sont asymétriques, la topologie est dynamique, la bande passante limitée et aucun organe dédié au routage n'est présent. L'ensemble des éléments du réseau participe activement au routage de l'information. Il est donc rendu plus difficile que dans les réseaux filaires traditionnels [Milanosic et al., 2004].

Les spécificités de ces systèmes réels immergés dans des environnements agressifs rendent l'utilisation de techniques multi-agents encore plus attractive par l'utilisation de l'auto-organisation pour assurer la gestion des communications et un caractère hautement adaptatif. En effet, le maintien de l'intégrité fonctionnelle sera intelligent dans le sens où il prend en considération les contraintes des constituants élémentaires du réseau qui peuvent évoluer en fonction du niveau d'énergie des stations.

Les éléments du réseau sont autonomes d'un point de vue énergétique. Un des objectifs globaux du système doit donc être de gérer au mieux cette dépense énergétique. Quand il n'y aucun envoi de message ces éléments sont dans un mode sommeil. C'est donc pendant qu'il y a communication qu'une différence peut se faire sur la gestion de l'énergie. Il faut donc utiliser un bon protocole de routage. La solution idéale est d'avoir des routes optimales (généralement en terme de sauts) pour un coût d'obtention de la route aussi bas que possible. Dans le cas d'environnement agressif, comme le cas de notre application à l'instrumentation d'un réseau hydrographique souterrain, des fautes internes peuvent intervenir au niveau

des constituants du réseau. Aussi l'infrastructure de communication doit être adaptative, tolérante aux pannes : un dysfonctionnement d'un des hôtes du réseau ne doit pas avoir un impact important sur le système et ne doit pas entraîner un coût énergétique d'adaptation élevé.

8.2 Le modèle MWAC

Nous présentons dans cette section le modèle d'organisation que nous proposons pour la gestion des communications dans les systèmes complexes physiques ouverts. Ce modèle est appelé MWAC pour Multi-Wireless-Agent Communication.

Pour une meilleure compréhension de l'analyse des différents axes de la décomposition AEIO, nous précisons dès maintenant qu'à chaque équipement sera associé un agent.

8.2.1 L'environnement

L'environnement est l'espace où évoluent les agents : il est donc directement lié au domaine d'application. Un paramètre important de notre environnement est la position géographique des agents. Nous admettons que dans le cadre de notre problématique, elle n'est pas directement observable sans quoi une approche traditionnelle centralisée basée sur des arbres de recouvrement suffirait.

8.2.2 L'organisation des agents

Nous avons décidé d'organiser les agents en groupes ayant une structure hiérarchique semblable à celle que l'on trouve dans les clusters de CGSR (cf. §6.4) afin de localiser au mieux l'inondation et ainsi obtenir un gain conséquent en énergie. Cependant contrairement à CGSR :

- on n'utilisera pas des protocoles de communication inter-groupes et extra-groupes,
- les noeuds ne procéderont pas à des diffusions périodiques comme c'est le cas pour CGSR, car cela entraîne un grand coût énergétique même si le réseau est en repos (aucun envoi de message n'est requis),
- les noeuds ne s'échangeront pas de tables de routage car une croyance erronée d'un agent pourrait se répandre et entraîner des boucles de routage,
- la création des groupes, grâce aux mécanismes d'auto-organisation, tiendra compte de divers paramètres (politique de gestion de l'énergie propre à chaque station, nombre de stations joignables...),
- supportera les liaisons non symétriques[1],
- contiendra des optimisations adaptées à notre problème telles que la mémorisation de la route la plus fréquemment utilisée (en l'occurrence celle qui mène à la station de travail).

La propriété principale de l'organisation est qu'elle est dynamique. Dans ce type d'application nous ne pouvons pas contrôler l'organisation a priori : la relation entre les agents émerge des interactions qu'ils ont entre eux et de l'évolution de leurs états.

Les groupes mis en oeuvre La structure de base de notre organisation (figure 8.1) est composée de :

– Un *agent représentant* qui administre les communications au sein de son groupe,

1. Une liaison est asymétrique si une station A peut joindre une station B alors que l'inverse n'est pas forcément possible

– Un ou plusieurs *agents de liaison* : ils sont aux frontières de plusieurs (en réalité ils appartiennent à plusieurs groupes) et permettent aux différents représentants de communiquer entre eux,
– Aucun ou plusieurs *simples membres* qui n'ont aucun rôle particulier dans le groupe si ce n'est recevoir et traiter les messages qui leurs sont destinés, transmettre leurs propres messages et vaquer à leur propre tâches. (Il ne participe pas au relais des messages).

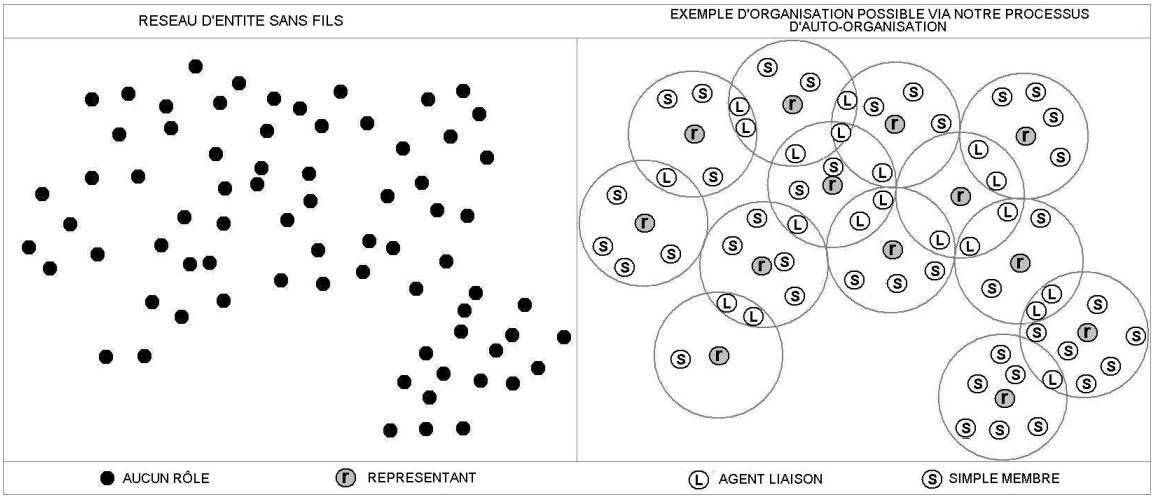


FIG. 8.1 – *Auto-organisation type obtenue avec notre approche*

On peut caractériser les groupes que nous avons créés à l'aide de différents attributs [Diaz et al., 1998]. Bien que le vocabulaire soit similaire à celui utilisé pour caractériser les systèmes multi-agents ou les organisations, le sens des attributs diffère.
Nos groupes sont *actifs* car une communication est mise en place. Ils sont *dynamiques* car les membres peuvent changer de rôle. Ils sont *ouverts* car des participants non membres du groupe peuvent intervenir dans la communication. Les groupes sont dits *partiellement déterminés* car chaque membre, hormis le représentant, ne connaît qu'un sous-ensemble des membres du groupe. Les interactions entre membres sont *asynchrones* car tous les membres n'ont pas besoin d'être présent pour qu'il y ait communication. L'interaction est *non contrôlée* car l'échange de données n'est pas restreint aux membres du groupe.
Un autre attribut concerne la durée du vie du groupe. Dans notre cas on ne peut savoir a priori s'ils sont *permanents*, à *longue durée* ou à *courte durée* de vie.

Les agents fonctionnent tous sur le même schéma mais leur rôle conditionnera des comportements spécifiques face aux messages qui proviennent des autres agents. En effet, tous les agents ont un même protocole d'introduction, ils participent tous à la recherche d'incohérence lors de l'envoi de message et ils désirent tous faire parvenir des informations à des destinataires particuliers. Cependant, un agent représentant devra en plus assumer toutes les requêtes d'envoi de message des membres de son groupe : quand le destinataire d'un message n'est pas voisin de l'émetteur, c'est à lui que revient le rôle de trouver une route, de vérifier le bon acheminement du message. C'est aussi lui qui est inondé par les requêtes de recherche de route des autres représentants. Il est donc plus sollicité que les autres agents de son groupe (hormis éventuellement les agents de liaison). Un agent de liaison est chargé d'assurer la communication entre les groupes c'est-à-dire ses voisins qui sont représentants. Ce sont donc eux qui distribuent les requêtes de recherche de route aux représentants. Les agents simples membres n'ont par contre aucun rôle particulier dans la gestion des messages.

Le fonctionnement de notre solution sera cependant très différente de CGSR. En effet, nous intégrons une réelle *politique de gestion de l'énergie* et nous procédons à la *suppression des envois périodiques* de messages qui lui permet de maintenir la cohérence de l'organisation. Le degré d'adaptation de ce protocole est donc fonction de cette fréquence d'envoi de message.

Une politique de gestion d'énergie. Un agent représentant étant très sollicité, d'un point de vue communication, il dépensera en moyenne plus d'énergie que les autres : il doit donc disposer d'une importante réserve énergétique. Plus un groupe est important, plus il présente un intérêt certain dans la réduction du volume transmis induit par l'inondation : le système préserve donc d'autant plus d'énergie. L'attribution de rôle doit intégrer ces informations. Elle se fait d'une manière très simple et avec la seule connaissance de ces voisins. La technique d'auto-organisation que nous utilisons est basée sur le mécanisme d'auto-organisation par attribution de rôle. Nous avons privilégié ce mécanisme aux autres (cf. §7.3.4) car il est à la fois simple et particulièrement adapté à la structure de groupe que nous souhaitons mettre en place. L'adéquation entre le désir qu'a un agent de devenir représentant et sa capacité à l'être est estimée grâce à une fonction $score$. C'est cette fonction qui doit mesurer le compromis entre les deux critères précédents et donc départager les agents lors de l'élection du représentant. La fonction $score$ que nous avons implémenté est très simple : elle multiplie le nombre de voisins par le niveau d'énergie de l'agent. Nous aurions pu cependant intégrer dans cette fonction la longévité d'un agent en tant que chef de groupe, la mobilité du noeuds etc.

```
SI monNombreDeVoisin ≠ 0 ALORS
    // On a des voisins
    SI nbRepresentantsVoisin = 0 ALORS
        // Aucun de nos voisins n'est représentant : on décide de le devenir. Ce cas intervient quand on vient
        // juste de créer l'agent ou quand il vient de se déplacer. On ne procède pas à un vote car on pourrait
        // rendre le système instable (la station va sûrement continuer sa route).
        monRôle = REPRESENTANT;
    SINON SI nbRepresentantsVoisin = 1 ET monRôle ≠ REPRESENTANT ALORS
        // Un de nos voisins est représentant : on se soumet à son autorité et ce même s'il est moins efficace
        // que nous : on privilégie (pour le moment) la stabilité dans l'organisation à sa performance. On
        // attendra une défaillance ou son souhait de quitter son mandat pour prendre sa place
        monRôle = SIMPLEMEMBRE;
    SINON
        //Il existe, dans notre voisinage, plusieurs représentants.
        SI monRôle = REPRESENTANT ALORS
            // Je suis moi-même représentant : j'entre en conflit avec les autres prétendants à ce rôle. Une
            // élection va avoir lieu et l'agent au meilleur score restera en place
            ProcedureElectionRepresentant()
        SINON
            // On n'est pas représentant : on devient agent de liaison pour ces représentants
            monRôle = LIAISON
        FINSI
    FINSI
SINON
    // On n'a pas de voisin : on n'a plus aucun rôle
    monRôle = AUCUN
FINSI
```

Comme le montre notre algorithme, lorsqu'un conflit intervient, une procédure d'élection du meilleur représentant est engagée. L'agent qui à détecté le conflit va calculer son $score$ et le transmettre au représentant avec lequel il est en conflit. Ce dernier, quand il recevra ce message prendra conscience (si

ce n'était pas déjà le cas) du problème et le comparera à son propre score. Si il est moins élevé, il démissionnera de son mandat, changera de rôle et le signalera à ses voisins. Cette algorithme est en réalité modulé par la correction d'incohérence (voir figure 8.3) que nous abordons par la suite.

Suppression des envois périodiques de messages. Nous adoptons le principe selon lequel l'objectif n'est pas d'avoir une organisation cohérente à tout moment : la perte énergétique associée ne serait pas forcément rentable. Cette organisation, si elle n'est pas cohérente, ne sera remise en question que lorsqu'elle posera problème dans le cadre d'un échange de communications. Pour cela, les agents font de l'*écoute indiscrète*. Ce mécanisme, que d'aucun appelle écoute flottante [Legras, 2002] consiste à espionner les messages qui passent dans son aire d'écoute afin d'en tirer des informations. Dans un médium de communication qui n'est pas point à point, tout agent muni d'un récepteur sans fil pourra entendre les messages émis par quelqu'un de son voisinage (voir figure 8.2).

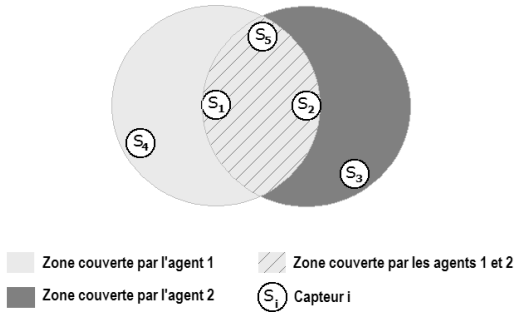


FIG. 8.2 – *Illustration de l'écoute indiscrète*

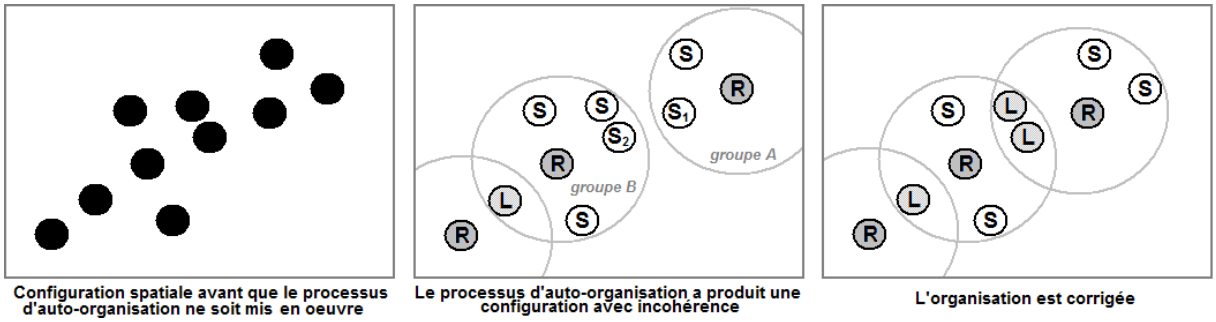
Quand l'agent s_1 envoie un message à s_2, s_4 et s_5 entendent le message contrairement à s_3.
Quand l'agent s_2 envoie un message à s_1, s_3 et s_5 entendent le message contrairement à s_4.

Une station peut, grâce à l'écoute indiscrète, vérifier :

– la cohérence de sa table des voisins ou de ses éventuelles tables de routage (ex: quand un voisin émet un message, l'agent vérifie que le triplet *<id,rôle,groupe>* de ce voisin émetteur de la trame qu'il entend soit bien dans la table de voisins).

– la cohérence structurelle dans le voisinage comme l'illustre la figure 8.3. Dans ce cas l'agent *<S1,SIMPLE MEMBRE,A>* qui reçoit le triplet *<S2,SIMPLE MEMBRE,B>* le met dans sa table de voisinage. Il signale ensuite à son représentant qu'il peut éventuellement y avoir une incohérence (message *VerificationCoherence(<B>)*). Son représentant regardera dans sa table des groupes voisins si ce groupe B est présent. Si il n'est pas joignable, selon qu'une route à faible coût existe ou non, soit le représentant avec le plus mauvais $score$ démissionne, soit les représentants décident de l'utiliser (afin d'éviter le coût d'une réorganisation). Si cette route est utilisée fréquemment, cette décision pourra être remise en question par un des représentants.

L'acquittement des messages intervient aussi dans ce processus de maintien de cohérence. Quand un message n'est pas acquitté, les représentants concernés suppriment la route de leurs tables et une nouvelle recherche de route est amorcée.



(R) Agent Représentant, (L) Agent de liaison, (S) agent Simple Membre

FIG. 8.3 – *Exemple de déclinaison cohérente/incohérente résultant de notre processus d'auto-organisation*

8.2.3 Données et tâches nécessaires aux agents

8.2.3.1 Les données de notre agent.

Les données que doivent contenir nos agents sont variables. Elles dépendent en effet du statut de l'agent dans le groupe.

Données sur soi. Tout agent connaît son identifiant, son statut dans le groupe (son rôle) et possède une table de voisins. Cette table de voisins contient, pour chacun d'eux un triplet <identifiant, rôle, groupe> auquel il appartient[2].

Données sur les autres. Les données que l'on possède sur les autres dépendent du rôle qu'a l'agent dans la société.

Si l'agent est un *représentant* (nous l'appellerons par la suite agent R) il aura en plus une *liste des groupes adjacents* qui lui permet de connaître l'identifiant des groupes (et donc de leur représentant) voisins. Il possède aussi une *table d'optimisation* qui lui permet de mémoriser un certain nombre[3] de chemins utilisés fréquemment. Cette route est en fait la liste des identifiants des représentants des groupes traversés pour joindre le destinataire.

Si l'agent est un *simple membre* (agent SM) alors il ne contient que la *table des voisins* définie précédemment. Il est important que l'agent connaisse le rôle et le groupe d'appartenance de ses voisins car il est en droit de remettre en cause l'organisation établie.

Si l'agent est un *agent de liaison* (agent L) il ne possède, tout comme l'agent *simple membre*, qu'une seule table : celle des voisins. Il sait qui sont les groupes voisins grâce à l'identifiant des représentants qu'il voit.

2. Pour les agents de liaison, ce champ est inutile car on sait qu'ils appartiennent à tous les groupes qu'ils voient et qui les voient.

3. Ce nombre sera fixé par les résultats de la simulation: il est un compromis entre l'espace mémoire occupé et le temps économisé grâce à cette mémorisation

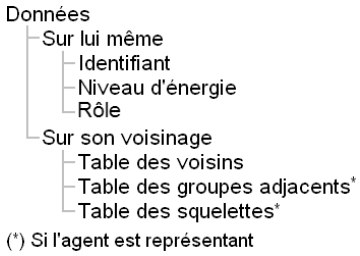


FIG. 8.4 – *Arbre des données de l'agent*

8.2.3.2 Les tâches et modules de l'agent

Conditionnées par l'architecture eASTRO sur laquelle nous avions préalablement travaillée, les tâches de l'agent peuvent se décomposer en quatre modules :

1. Le module de communication/perception. Le module de communication dote l'agent de primitives de communication : nous trouvons dans ce modules les primitives d'émission et réception de trames.

2. Le module de contrôle de l'agent. Ce module permet à l'agent de détecter une situation nécessitant une réaction. La création des buts sera sensible à l'arrivée de message (faut-il le répéter? l'interpréter?).

3. Le module de raisonnement. Ce module a pour fonction de choisir les plans à appliquer selon les buts à atteindre. Il contient donc toutes les politiques de l'agent (politique de gestion de l'énergie, politique sociale...). La fonction $score$ sera donc dans ce module.

4. Le module de décision. Ce module choisit la meilleure façon d'exécuter un plan.

8.2.4 L'axe I : Les interactions

Pour gérer les interactions internes aux systèmes multi-agents nous allons, dans cette partie, nous efforcer de définir un protocole d'interaction c'est-à-dire l'ensemble des lignes de conduite que les agents doivent suivre et qui structurent leurs échanges de messages [Occello et al., 2001].

Les protocoles d'interaction se distinguent des protocoles de communication qui existent dans les réseaux d'ordinateurs ou autres systèmes répartis [Koning et al., 1995]. En effet, les protocoles d'interaction sont utilisés pour contrôler et structurer les échanges d'informations entre agents; l'objectif étant de diminuer la quantité d'information échangée ainsi que le temps de communication. Ils définissent donc le contexte dans lequel un message est envoyé.

Dans notre système, les agents interagissent seulement avec les agents en accointance avec eux. Un agent est en accointance avec un autre si et seulement si il est conscient de son existence c'est-à-dire présent dans son aire d'émission/réception. Ils interagissent via l'envoi de message. Nous pouvons distinguer deux méthodes pour réaliser ce type d'interaction : une méthode synchrone et une autre asynchrone. Dans le premier cas les agents qui communiquent doivent être en rendez-vous. Dans l'autre cas, la version asynchrone, les messages sont mémorisés. Il n'est pas nécessaire que le récepteur et l'émetteur soient donc synchronisés. Du point de vue de la communication, la solution asynchrone nous semble être la meilleure car c'est la méthode la plus flexible.

Les types de messages possibles. Nous avons décidé de créer notre propre protocole d'interaction car ceux qui existent [FIPA, 2002a] sont beaucoup trop lourds à mettre en oeuvre pour notre application, destinée à être embarquée et où le volume des émissions est à minimiser. Pour notre protocole, nous avons créé 13 types de messages différents :

- Le message $QuiSontMesVoisins$ est un message qui permet à un agent de demander à ses voisins de se faire connaître. Ce message est émis quand un agent est créé (le premier but d'un nouvel agent est de connaître ses voisins). Ensuite, ce message ne sera utilisé que lorsque l'agent aura l'impression (qu'elle soit justifiée ou non) que sa table des voisins ne coïncide plus avec la réalité (par exemple s' il émet un message et qu'il ne reçoit pas d'ACK[4], depuis un moment qu'il juge trop long[5] il n'entend pas de communication autour de lui...).
- Le message $JeSuisUnDeTesVoisins$ permet à un agent de répondre à la précédente requête. Avec ce message, il fournit donc son identifiant, son rôle et son groupe d'appartenance.
- Le message $JeChangeDeRôle$ est utilisé par tout agent qui a changé de rôle pour en avertir ses voisins. Ce message contient comme information l'identifiant de l'agent, son rôle et son groupe d'appartenance.
- Le message $DemandeGrpAgentLiaison$ est utilisé par les représentants qui veulent remettre à jour leur connaissance sur les groupes voisins. Elle oblige les agents de liaison à répondre.
- Le message $ReponseGrpAgentLiaison$ est utilisé par un agent de liaison pour signaler à un représentant la liste des groupes que cet agent peut contacter.
- Le message $VerifieCoherenceGroupeVoisin$ est envoyé par un agent qui croit avoir détecté une incohérence avec un groupe voisin. Il y a incohérence entre deux groupes quand deux simples membres de groupes différents se voient et que leur représentant ne peuvent pas communiquer.
- Le message $ResolutionConflitRepresentant$ est un message utilisé par un représentant en conflit avec un ou plusieurs pour communiquer son score.
- Le message $PasseRepresentant$ est un ordre donné par un représentant à un des agents de son groupe. Il peut donner cet ordre s'il s'aperçoit par exemple qu'un groupe voisin est isolé et que la seule manière de briser cet isolement est qu'un agent particulier devienne représentant.
- Le message $ChercheRoute$ est utilisé par un représentant qui veut connaître la route à prendre (succession de couples (agent de liaison, représentant)) pour joindre un individu.
- Le message $ResultatChercheRoute$ est la réponse du représentant qui à le destinataire à sa charge.
- Le message $ACKMessage$ est un message de configuration qui confirme à l'émetteur que son message est parvenu à destination. Il intervient donc dans la gestion de la cohérence de l'organisation.
- Le message $DonneesEncapsulees$ est un message qui encapsule des données.
- Le message $RouteErronee$ est un message envoyé par un représentant qui s'est aperçu d'un problème. Ce message prend la route erronée.

Protocole d'introduction. Dans notre application nous utilisons, entre autres, un protocole d'introduction des agents dans l'organisation. Il correspond à une conversation entre plusieurs agents. Il ne se situe pas au même niveau que les approches facilitateurs FIPA et KQML que connaissent les systèmes multi-agents. Cependant, ils pourraient être encapsulés via les messages *DonneesEncapsulees* pour être traités par l'agent. Ce protocole permet à un agent de se faire connaître par ses voisins et d'entrer dans

4. Abréviation utilisée pour le terme anglais ACKnowledge. Il s'agit de la confirmation de bonne réception d'un message.
5. Fonction de son rôle.

l'organisation. Il s'appuie sur certains des treize types de messages que nous avons définis (voir paragraphe suivant), tous d'une petite taille (la contrainte sur l'énergie étant à prendre en considération). Ainsi s'introduire correspond à envoyer le message *QuiSontMesVoisins* qui nous fait connaître de nos voisins et leur demande de se présenter. Ces voisins répondent avec le message *JeSuisUnDeTesVoisins* précisant ainsi le triplet *<id,rôle,groupe>* qui leur est associé. Le station détermine, en fonction de ce voisinage, le rôle qu'il doit prendre. Il le signalera à ses voisins via le message *JeChangeDeRôle*.

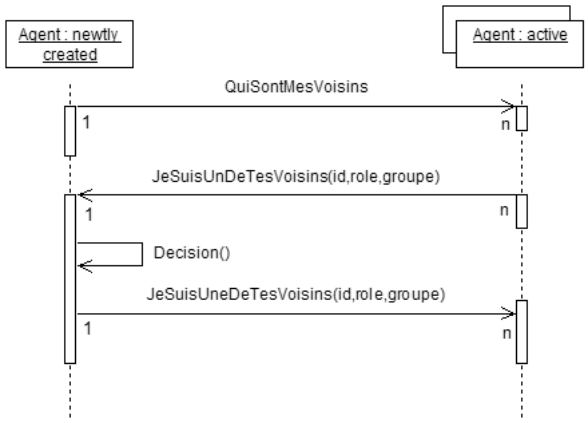


FIG. 8.5 – *Protocole d'introduction*

Protocole d'élection d'un représentant. Nous avons vu que dans l'algorithme que l'agent met en oeuvre pour s'attribuer un rôle dans l'organisation peut exécuter une procédure d'élection d'un représentant. Cette procédure intervient quand au moins deux voisins veulent assumer le rôle de représentant.

Quand au moins deux agents en accointance sont représentants, chacun d'entre eux a indiqué à ses voisins son changement de rôle via le message $JeChangeDeRôle$. Dès qu'un des représentants détecte qu'un de ses propres voisins a aussi ce rôle, le protocole d'élection (figure 8.6) est instancié.

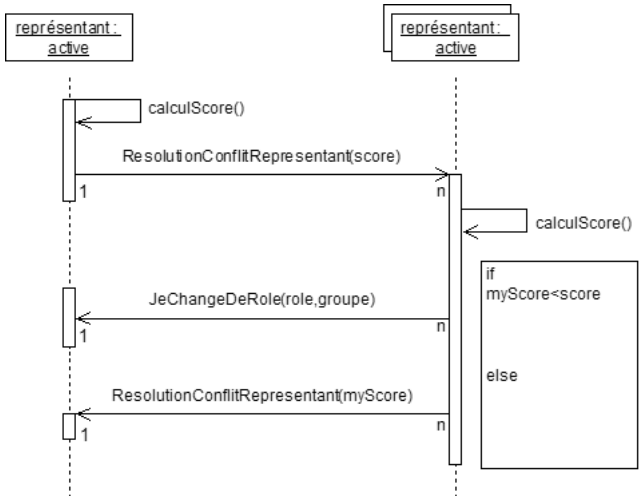


FIG. 8.6 – *Protocole d'élection du meilleur représentant*

8.3 Evaluation comparative et caractéristiques de notre approche

Les résultats obtenus sont présentés pour différentes topologies. Le but de ces manipulations est de déterminer les caractéristiques de notre approche et aussi d'avoir une première appréciation de son efficacité en comparaison avec le protocole DSR. Ce protocole est très connu et facile à implémenter. L'intérêt réside essentiellement dans le fait que de nombreux protocoles s'y comparent.

8.3.1 Evaluation du rendement de notre approche

Protocole expérimental. Nous allons comparer les performances de notre solution avec les performances du protocole DSR vu précédemment. L'évaluation sera faite pour des réseaux de 15, 50, 100 et 200 stations. Les noyaux des réseaux, c'est-à-dire les configurations spatiales initiales, sont données en 8.7. Nous avons fait figurer, sur ces illustrations, une organisation possible issu du processus d'auto-organisation. Cette situation est critique dans le sens où les groupes sont peu denses (portée des stations réduite). Une conséquence visible est le nombre réduit d'agents ayant le rôle de simple membre.

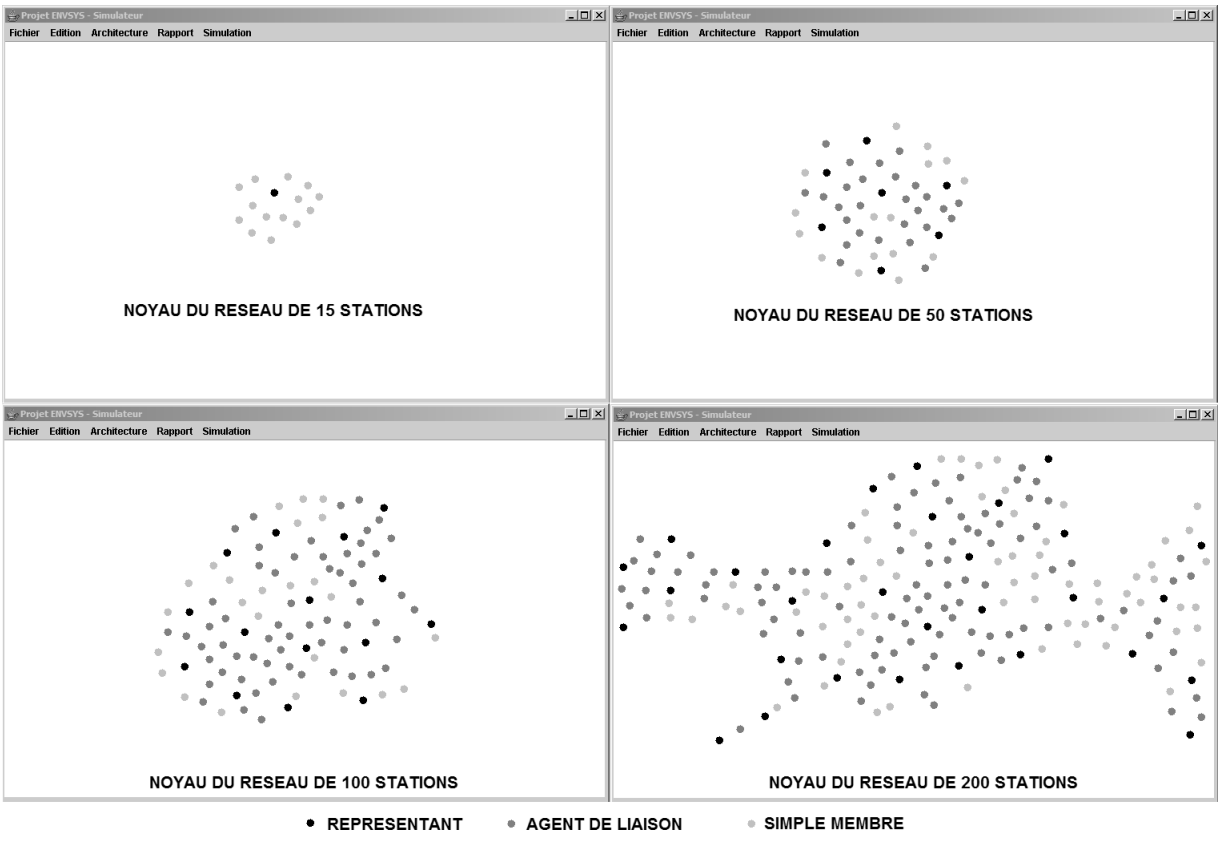


FIG. 8.7 – *Configurations spatiales initiales des simulations pour le réseau en grappe*

Pour chacune de ces configurations, toutes les secondes un agent, choisi au hasard, envoie à un autre, lui aussi choisi au hasard. Les messages envoyés ont un champ de donnée, selon le cas, de 10, 256 ou 1024 octets. Toute les dix secondes une station tombe en panne pour dix secondes.

Performances mesurées. Nous présentons en figure 8.8 les résultats obtenus en simulation. Les graphiques mettent en évidence le ratio volume total transmis par le système avec DSR sur le volume total transmis par notre solution. Pour chacune des configurations notre approche est plus efficace que DSR (ratio supérieur à 1). Il faut aussi garder à l'esprit que l'acquittement des messages intervient dans le maintien de la cohérence de l'auto-organisation au même titre que l'écoute indiscrète qui consiste à extraire des informations pertinentes des messages échangés dans le voisinage.

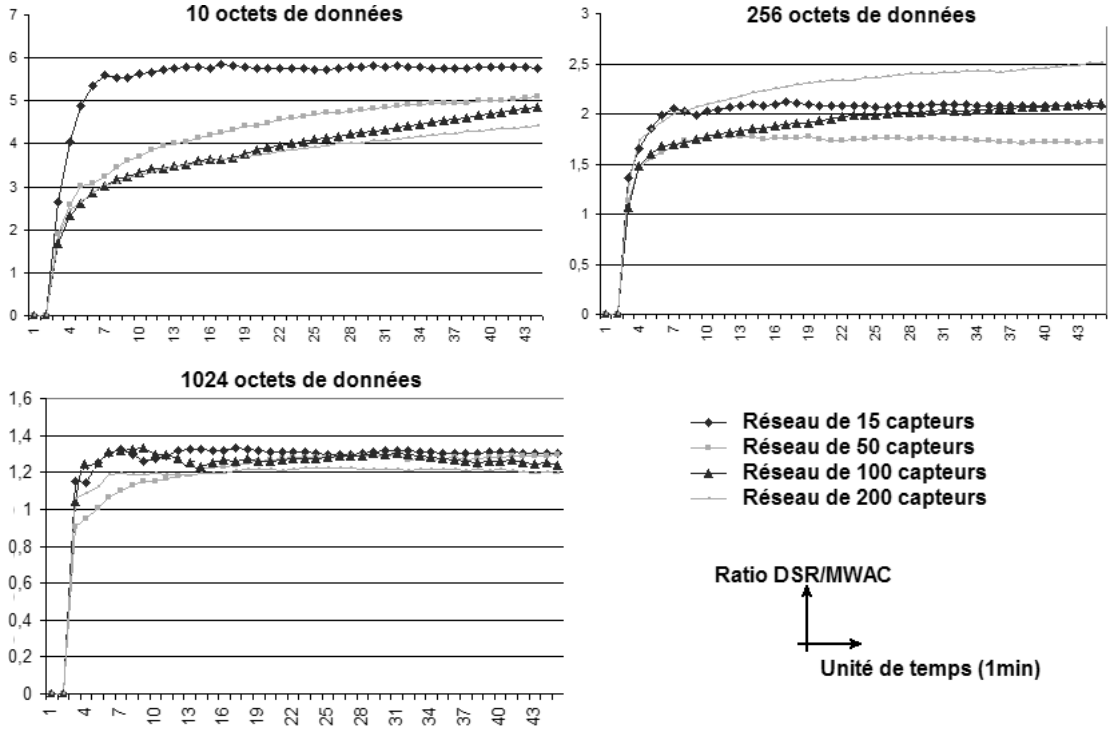


FIG. 8.8 – *Comparaison des performances de DSR/MWAC dans le cas de réseaux maillés*

Ces mesures mettent en évidence différentes caractéristiques de notre approche que nous présentons selon le critère observé.

Notre approche est pénalisée par sa non optimalité des routes. On remarque que plus le volume des messages transmis est important, plus nos performances se rapprochent de celles de DSR. Cela provient du fait que les routes que nous fournissons sont optimales en terme de groupe et non de route. Ce défaut est tout même moins pénalisant que le gain obtenu par l'auto-organisation pendant une recherche de routes. Si les ratios de DSR et MWAC se rapprochent, ce n'est pas que les performances de notre approche sont moins importantes. En effet, la majeure partie des volumes que l'on divise est constituée par les données.
D'une manière générale, plus le réseau sera vaste moins ce problème sera pénalisant comme le montre la figure 8.9.

Notre approche supporte le passage à l'échelle. Notre approche supporte très bien le passage à l'échelle. En effet, on remarque que le nombre de stations ne nuit pas beaucoup à sa performance. Cela

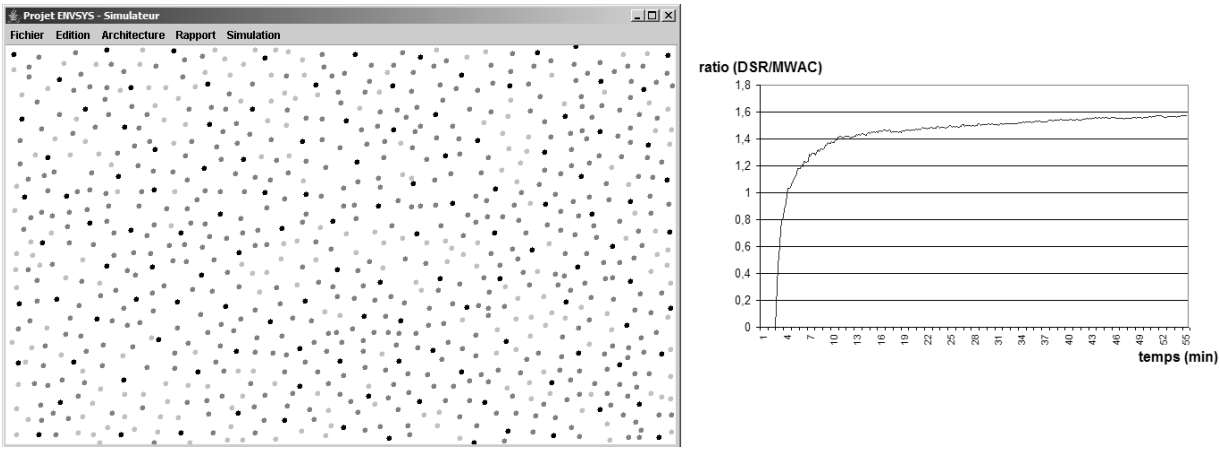


FIG. 8.9 – *Performances comparative DSR/MWAC dans le cas de réseaux maillés*

tient au fait que le nombre de groupe n'augmente pas proportionnellement avec le nombre de stations. Selon les cas des groupes apparaissent ou deviennent simplement plus denses.

La performance de notre solution est biaisé par le volume d'auto-organisation initiale.. On remarque un ratio nul pendant l'intervalle de temps [0,3] : cela vient du fait que nos agents se sont auto-organisés alors qu'aucun envoi de message n'a été demandé. Cependant, notre approche peut être assimilée à un traitement *basé sur la demande* car l'acheminement des messages s'adapte au trafic en fonction de la demande et du besoin (hormis à l'instant t=0 où la station est mise en fonction). Il n'y a en effet pas de distribution uniforme du trafic sur le réseau : on ne maintient pas l'organisation dans une situation valide si cela n'est pas nécessaire.

8.3.2 Evaluation de l'auto-organisation.

En terme de résultats, les comparaisons avec DSR montrent que notre approche est plus efficace. La structure de groupes des agents à donc bien permis une réduction du volume des messages qui permettent la localisation des autres agents et donc l'établissement de routes. La *validité* de notre approche peut donc être établie.
Les raisonnements mis en jeu par les agents qui participent au processus d'auto-organisation ne nécessitent pas de primitives très cognitives. Le résultat (les groupes) de ce processus étant facilement identifiable et compréhensible par un observateur extérieur, la *simplicité* du processus peut être constatée.

Identifiant	Evènement
A	Auto-organisation de 300 stations
B	correction d'une incohérence
C	ajout de 10 stations
D	correction survenue à la suite d'une perturbation (déplacement d'une station)

En terme d'auto-organisation, la figure 8.10 permet de constater la *stabilité*. Les structures qui sont établies font preuve d'une constance dans le temps. Elles ne sont remises en cause que lorsque une incohérence (évènement B) est détectée. La figure permet de voir les envois de messages qui ont permis

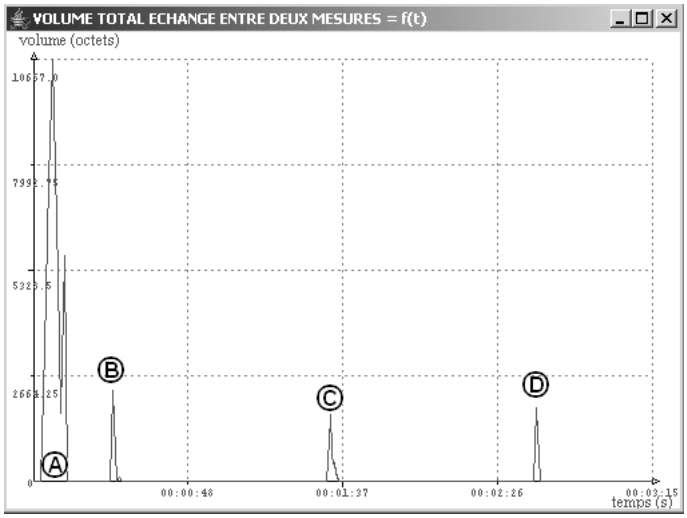


FIG. 8.10 – *Variation du volume échangé par le processus d'auto-organisation*

ces modifications. Le déplacement de stations (évènement D) peut entraîner des ré-élections si l'agent qui se déplace est un représentant ou un agent de liaison qui est le seul à permettre la communication entre deux groupes connexes. Dans les autres cas, cette mobilité n'aurait entraîné aucune autre réélection de représentant : elle change seulement la structure interne de certains groupes. Il est à noter que les messages échangés qui concernent l'ajout des 10 stations (évènement C) n'ont pas affecté la topologie existante. Les agents représentants existants n'ont pas perdu leur mandat afin de privilégier justement cette stabilité du processus. Par contre des groupes ont vu leur nombre de simples membres ou d'agents de liaison augmenter et d'autres groupes sont apparus.
La sensibilité du processus, que l'on ne peut complètement dissocier du critère de stabilité, a été abordée dans le point précédent. Les états de reconnaissance totale concernent les structures cohérentes. D'éventuelles structures incohérentes peuvent avoir été détectées précédemment (état de reconnaissance partielle) et être corrigées. Il est à noter que si le processus mis en place avait été trop sensible, il y aurait eu un impact visible sur les rendements des protocoles. A contrario, si il avait été trop peu sensible, la tolérance aux pannes qui aurait été mauvaise.
La convergence ne peut être que constatée sur les simulations.

8.4 Un intergiciel multi-agents par envoi de messages pour les systèmes multi-agents physiques

8.4.1 Avant-propos

Un intergiciel par envoi de messages. Pour concevoir des systèmes complexes physiques ouverts, il nous est nécessaire de pouvoir faire abstraction du côté sans fil des applications. Pour cela nous allons concevoir une couche support pour la gestion des communications, entre les entités mobiles, basée sur le modèle précédemment exposé. Cette couche doit ainsi accroître l'interopérabilité, la portabilité et la flexibilité des applications concues. Cet intergiciel sera de type MOM (Message Oriented Middleware [Banavar et al., 1999]).

Systèmes multi-agents et intergiciels. L'approche multi-agents semble toute indiquée pour construire des intergiciels de ce type car il s'agit d'une application par essence même distribuée, qui doit permettre de s'affranchir des multiples changements que peut imposer l'environnement, qui ne doit nécessiter aucune configuration manuelle lors de l'ajout ou la suppression de stations.

Notre approche, comme nous l'avons vu précédemment, repose sur l'auto-organisation. La pertinence de l'opérationalisation de l'auto-organisation par les systèmes multi-agents a été maintes fois vérifiée et notamment son adéquation aux domaines des réseaux [Foukia and Hassas, 2003]. Il est fréquent de parler d'agents lorsqu'on s'intéresse aux intergiciels. Il est cependant original d'adopter une approche multi-agents pour concevoir des intergiciels. Bien que ne prenant nullement en compte la gestion de l'énergie, quelques travaux traitent néanmoins du rapprochement de ces deux domaines. L'intergiciel ASAM (Adaptive Service Access Management) proposé dans [Calisti et al., 2004] a pour objectif l'adaptation dynamique aux ressources réseaux disponibles. L'utilisation d'un intergiciel pour maintenir une communication point à point malgré le déplacement des équipements est abordée dans TOTA [Mamei and Zambonelli, 2003].

8.4.2 L'intergiciel MWAC

Les applications que nous construisons sont distribuées sur de nombreux agents. Cela nous permet de réduire la complexité de conception. Nos agents, en plus de leur tâches applicatives, embarquent tous un module intergiciel et ce indépendamment de leur architecture (voir figure 8.11).

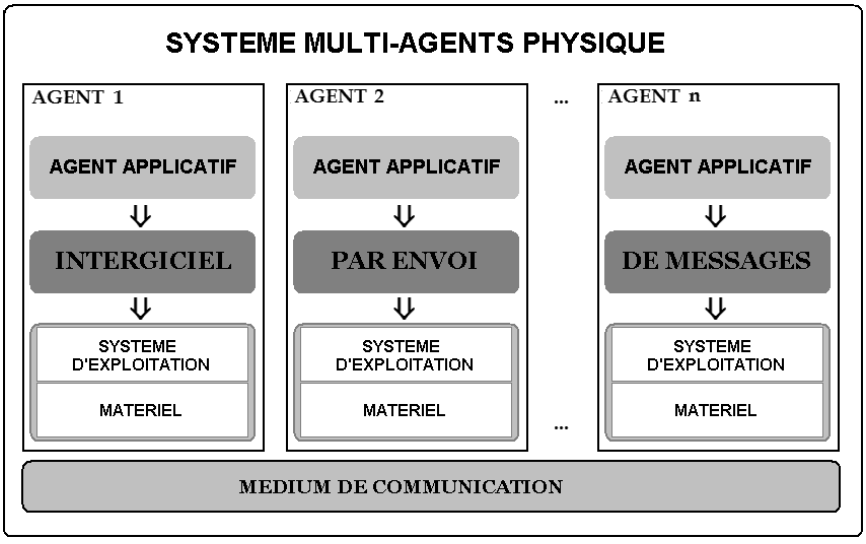


FIG. 8.11 – *Architecture de nos systèmes multi-agents physiques*

Il existe alors deux manières de voir l'intergiciel : un composant MWAC et un archétype d'agent MWAC.

Le composant MWAC. L'intergiciel peut être enfoui dans un composant que nous appelons MWAC (du nom de notre modèle). Nous partageons les entrées/sorties en deux familles : les entrées/sorties logiques (à destination de la partie commande de l'agent) et les entrées/sorties bits (à destination des

organes d'émission/réception).

- Les entrées logiques du composant sont :
  - l'identifiant de l'agent,
  - le niveau d'énergie (entier compris entre 0 et 100),
  - un couple (destinataire, message à émettre).
- La sortie logique du composant sont :
  - un couple (émetteur, message reçu).
- Les entrées au niveau bit du composant sont :
  - un flot de bits en entrée (données brutes[6] reçues par le récepteur HF).
- Les sorties logiques du composant sont :
  - un flot de bits en sortie.

Cette implémentation de l'intergiciel permet de faire totalement abstraction du modèle : il conviendra donc particulièrement bien aux entités actives non critiques d'un point de vue énergétique et à la plupart des applications multi-agents et ce indépendamment des architectures de ces derniers.

Un archétype d'agent. Certaines applications critiques (notamment d'un point de vue énergétique) nécessitent d'avoir accès à des données internes de l'agent comme la liste des voisins ou même de dégrader les interactions (voir l'application EnvSys, §9.1). Pour les applications nécessitant plus de flexibilité de ce point de vue, nous proposons un archétype d'agent. Cet archétype d'agent est basé sur l'architecture eASTRO (cf §5.4.4). Pour chacun des modules de cette architecture, des éléments du modèle sont instanciés. Par exemple, le module de représentation du monde contiendra d'office les données sur les autres et sur soi. La mise en oeuvre d'un tel intergiciel est abordé dans l'application EnvSys.

8.5 Conclusion

Nous avons proposé un modèle basé sur l'auto-organisation pour permettre une gestion efficace des communications dans un réseau sans fil soumis à des contraintes énergétiques. La structure de base de notre organisation est un groupe mettant en jeu trois différents rôles : les représentants qui gèrent les communications au niveau du groupe, les agents de liaison qui permettent les communications entre groupes et les simples-membres qui ne jouent aucun rôle actif dans l'acheminement des messages. Le mécanisme utilisé est basé sur l'attribution de rôle. Cependant, l'écoute indiscrète est utilisé par les agents pour vérifier leurs croyances sur leur environnement.

L'auto-organisation permet l'ajout ou la suppression d'éléments sans nécessiter de reconfiguration manuelle. Les capacités adaptatives de l'auto-organisation permettent une tolérance aux pannes: le dysfonctionnement d'une des entités ne nuit pas à l'intégrité de notre système. Notre phase de simulation nous a permis de comparer notre approche multi-agents par rapport à des approches classiques.

Les agents présentent des caractéristiques intéressantes en terme d'ingénierie logicielle comme la généricité qui permet une évolution facile de l'application. La solution a d'ailleurs été adaptée en intergiciel par envoi de messages exploitable sous la forme d'un composant ou d'un archétype d'agent.

6. Par brute on entend sans les encapsulations des éventuels protocoles des couches inférieures.

Conclusion

La troisième partie de ce mémoire a abordée le point dur principal en terme de modèle dans la catégorie de systèmes visés par la méthode DIAMOND : celui du maintien d'intégrité fonctionnelle et de la communication dans un réseau ouvert d'entités physiques. Le sans fil est la traduction physique extrême de la décentralisation. Ce point concerne donc essentiellement l'intégration dans les agents d'aspects relevant de l'interaction et de l'organisation.

Dans un premier chapitre, nous avons introduit un domaine de recherche très actif, celui des réseaux sans fil et plus particulièrement le routage dans les réseaux ad-hoc. Nous avons présenté une revue des protocoles de routage, solution proposée par la communauté des réseaux.

Dans un second chapitre, nous avons présenté les deux grands courant de maintien d'intégrité fonctionnelle en informatique : la réplication de services et l'auto-organisation. Nous avons insisté sur cette dernière car elle est la seule applicable au contexte du routage.

Dans le troisième chapitre nous présentons notre solution pour la gestion des communications dans les systèmes complexes physiques. Notre modèle n'est pas basé sur un protocole de routage mais par une infra-structure dont une des fonctions est cette gestion du routage. Ce modèle est le coeur d'un intergiciel.

TAB. 8.1 – *Appréciation de notre approche selon les critères de la RFC 2501*

Critère	Appréciations
Taille du réseau	Notre modèle supporte bien le passage à l'échelle (à moduler par la non optimalité des routes)
Connectivité du réseau	Les noeuds supportent un grand nombre de voisins. En fait, plus le nombres de voisins est élevé plus l'approche par auto-organisation est intéressante.
Taux de changement de topologie et mobilité	Ce critère est toujours difficile à apprécier. Les changements de topologie d'autant mieux tolérés que le nombre d'échanges de messages est élevé (conséquence de l'écoute indiscrète). Un changement de topologie localisé n'a aucune conséquence sur les autres noeuds.
Capacité des liaisons	Ce critère est dépendant de la couche physique et donc de l'implémentation.
Taux de liaisons unidirectionnelles	En présence de liaisons unidirectionnelles l'agent ayant la plus faible portée sera simple membre
Type de trafic	En cas d'un système à trafic non uniforme, la cohérence de l'organisation n'est pas maintenue.
Ratio et fréquence des périodes de sommeil des noeuds	Ce critère est dépendant de la couche physique et donc de l'implémentation.

Le modèle MWAC et son intergiciel associé vont nous permettre de faire abstraction de la gestion des communications dans les systèmes complexes physiques ouverts que nous souhaitons modéliser. Nous présentons, pour conclure, un tableau récapitulatif de l'adéquation entre notre approche et certaines caractéristiques de problèmes selon la RFC [7] 2501.

7. Les RFC (Request For Comment) sont des documents publiés par l'IETF (Internet Engineering Task Force) qui décrivent, spécifient, aident à l'implémentation, standardisent et débattent de la majorité des normes, standards, technologies et protocoles liés à Internet et aux réseaux en général.

quatrième partie

UNE APPLICATION DE DIAMOND

Introduction

Ce partie est consacrée à une application qui met en oeuvre la méthode DIAMOND et une utilisation du modèle MWAC.

Le premier chapitre de cette partie commence par présenter le projet EnvSys qui a pour but l'instrumentation d'un réseau hydrographique souterrain. Les objectifs et la problématique d'une telle instrumentation sont abordés. Il présente ensuite l'analyse du problème selon la méthode DIAMOND.

Dans un second chapitre, nous présentons la plate-forme de simulation qui nous a permis d'évaluer les performances attendues de notre solution. Nous introduisons aussi l'implémentation des agents (plate-forme cible et organisation du code).

Chapitre 9

Application à l'instrumentation sans fil : gestion d'un réseau de capteurs sans fil

Ce chapitre présente l'application de notre méthode et de notre modèle de communication pour les réseaux sans fil dans le cadre du projet EnvSys (Environment System). Dans une première section nous introduisons ce projet d'instrumentation non filaire d'un réseau hydrographique souterrain. Dans une deuxième section nous présentons le traitement du problème avec la méthode DIAMOND.

9.1 Le projet EnvSys

Le projet EnvSys[1] (ENVironment SYStem) a pour objectif l'instrumentation d'une cavité souterraine. Nous allons présenter l'origine du projet ainsi que la problématique que soulève une telle instrumentation.

9.1.1 Objectif.

Les paramètres intéressants à mesurer dans un réseau hydrographique souterrain sont nombreux: température de l'air et de l'eau, pression de l'air et éventuellement de l'eau pour les réseaux noyés, degré de pollution par certains polluants classiques, débit d'eau, vitesse de courant d'air, etc. L'objectif du projet est de proposer un système pour recueillir toutes ces informations et de les collecter à la sortie immédiate du réseau via, par exemple, une station de travail type PC. Elle pourront être ensuite traitées pour déclencher des alarmes, étudier la progression d'une certaine pollution en fonction de divers autres paramètres mesurés, déterminer un modèle prédictif de l'ensemble du réseau en mettant en relation les paramètres souterrains mesurés par notre système et les paramètres aériens mesurés plus classiquement sur le bassin d'alimentation.

9.1.2 Problématique.

La mesure des différents paramètres d'un réseau hydrographique souterrain est une chose compliquée. En effet :

– L'accès aux réseaux souterrains est difficile au point qu'il devient nécessaire de faire appel à des spéléologues. Cela complique énormément la pose des réseaux de communications filaires d'autant

1. Projet soutenu par le Fonds d'Incitation au Transfert de Technologie (FITT) de la région Rhône-Alpes

plus que la structure de ces réseaux hydrographiques est très souvent fort chaotique.

– Dans le cas d'un réseau de communication hertzien, le fait d'être sous terre rend la propagation des ondes difficile et les techniques ne sont pas encore totalement maîtrisées.

Depuis quelques années, des systèmes de communication radio ont vu le jour. Ils sont généralement utilisés par les secours spéléologiques. Ces systèmes analogiques fonctionnent à basses fréquences et très souvent en phonie.

Il est donc nécessaire d'étudier la faisabilité d'un réseau maillé de capteurs à partir de la couche physique existante. Il permettra à terme l'instrumentation non filaire d'un réseau hydrographique souterrain. Ce système présentera un grand intérêt dans bien des domaines: étude des écoulements souterrains, surveillance de captages profonds, gestion des risques de crues, détection en amont de réseaux hydrographiques des risques de pollution...

Nous ne souhaitons pas procéder à cette instrumentation avec un réseau filaire (comme à l'accoutumé) pour des raisons évidentes de commodité. Une illustration du réseau sans fil attendu figure en 9.1. Nous utiliserons donc les ondes électromagnétiques à basse fréquence comme porteuses. Ces ondes ont la propriété de pouvoir traverser, entre autres, les parois rocheuses.

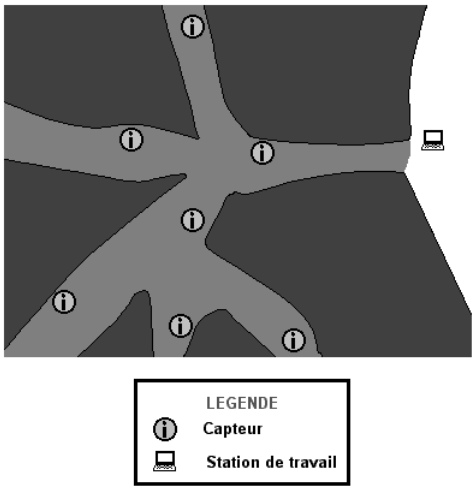


FIG. 9.1 – *Le sous-terrain instrumenté*

9.2 Traitement du problème avec DIAMOND

Dans cette section, nous traitons le problème avec la méthode DIAMOND. La première occurrence d'un mot écrit en police `typewriter` signifie qu'il figure dans le glossaire (fig. 9.13, page 182).

9.2.1 Avant propos

Nous avons présenté, précédemment, les caractéristiques des réseaux ad-hoc dont les réseaux de capteurs (WSN[2]) sont une famille particulière. Un état de l'art des difficultés que posent les réseaux de capteurs sans fil, aujourd'hui collectivement acceptées, est exposé dans [Zhang et al., 2002]. Ces difficultés sont liées aux particularités de cette famille de réseaux ad-hoc.

2. Wireless Sensor Network

En effet, dans un réseaux de capteurs sans fil, les éléments du réseau sont indépendants du point de vue de leur énergie et ne peuvent être rechargés de façon simple comme dans les réseaux cellulaires. Cela a de nombreuses répercussions sur la manière de gérer le système et même de le concevoir. Cette contrainte forte doit être prise en compte à tous les niveaux de la conception et de la réalisation. Une attention toute particulière est nécessaire lorsque l'on s'intéresse au problème du routage de l'information (l'énergie dépensée lors des communications est une part importante des dépenses énergétiques globales). Le fait d'avoir peu d'énergie peut se traduire au niveau du capteur par l'apparition de fautes internes et peut influer sur divers paramètres tels que la portée des messages que l'on envoie, la qualité et la précision des mesures effectuées etc. Par contre, les noeuds sont généralement stationnaires après une période de déploiement exceptés pour quelques uns d'entre eux. L'environnement hostile (en terme de température, pression, humidité...) dans lequel sont généralement plongés les dispositifs peut aussi contribuer à créer des fautes inattendues et à requérir des re-configuration automatiques et dynamiques. L'ensemble peut aussi être soumis à des contraintes temporelles.

9.2.2 Définition des besoins (A)

9.2.2.1 Approche préliminaire (A.1.)

L'objectif du projet EnvSys est de procéder à l'`instrumentation` d'un réseau hydrographique souterrain (voir figure 9.1) à l'aide de `capteurs intelligents` utilisant une technologie de transmission sans fil afin de faire parvenir des mesures vers une station de collecte située en sortie du réseau hydrographique.

La station de collecte. Cette station de travail type PC permet de collecter les mesures qui parviennent des différents capteurs. Elles sont stockées dans un fichier texte contenant des enregistrements du type :

$$< id\_capteurs\_source >< id\_mesure_1 >< mesure_1 > ... < id\_mesure_n >< valeur_n >< RC > \quad (9.1)$$

où id_capteur est l'identifiant du capteur sur 2 octets, id_mesure est l'identifiant du type de mesure sur un octet (1:température, 2:pression, 3:ph), valeur est la mesure représentée par un réel codé suivant la norme IEEE 754 (simple précision soit 4 octets). Ces mesures sont utilisées par un `analyste` (visualisable sur site ou à distance). Cette station permet aussi de connaître les identifiants de tous les capteurs joignables. Cela permet aux `spéléologues` de vérifier que les capteurs qu'ils ont posés sont toujours présents.

Les capteurs. Les capteurs sont plongés dans le réseau hydrographique souterrain. Ils sont autonomes d'un point de vue énergétique (batterie). Ils doivent alors assurer plusieurs fonctions :

- Une fonction mesure : c'est le rôle premier d'un capteur. Il consiste à interagir avec l'environnement pour saisir l'information.
- Une fonction relais : chaque capteur a une portée de transmission limitée. Il est donc nécessaire que chaque capteur aide les autres à faire parvenir leurs mesures à la station de travail.

La communication sans fil. La communication sans fil dans ce type d'environnement présente des particularités fortes notamment sur la portée des capteurs. En effet, de nombreux facteurs jouent sur cette portée maximale des capteurs. Ces limitations proviennent tout d'abord des solutions technologiques qui ont été utilisées pour réaliser les modules d'émission/réception (fréquence, puissance, antenne) et de

la manière dont ces solutions ont été implantées. L'environnement du capteur a aussi un fort impact : l'onde électromagnétique, selon les obstacles qu'elle aura à traverser, ne sera pas exploitable à une distance unique selon chacune de ses directions. La zone de transmission ne sera pas modélisable par une sphère. Le schéma référencé 9.2 peut nous aider à comprendre ce phénomène. La distance d_1 qui sépare les capteurs i_1 et i_2 est inférieure à la distance d_2 qui sépare les capteurs i_1 et i_3. Cependant le bloc de roche séparant le couple de capteurs (i_1, i_2) dégénèrera le signal et fera que le capteur i_2 ne pourra pas recevoir correctement le message contrairement au capteur i_3.

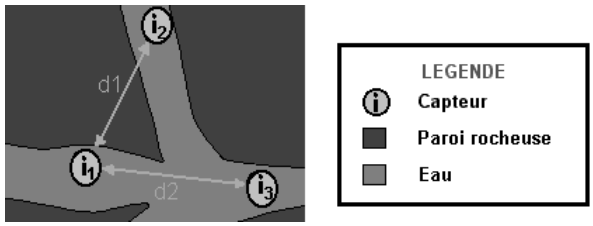


FIG. 9.2 – *Propagation anisotropique du signal*

La tolérance aux pannes. Le maintien de cohérence est un des problèmes centraux du projet ENVSYS. Il convient, pour pouvoir traiter cette question, de trouver les sources possibles d'une violation de l'intégrité fonctionnelle. De ces perturbations, nous pourrons dégager les contraintes auxquelles notre solution devra se soumettre. Elles constitueront nos critères de classification des systèmes existants ainsi que de recommandation pour le système que nous proposerons.

Guy Pujolle, lorsqu'il aborde la tolérance aux pannes, identifie deux types de défauts [Pujolle, 2000]. Les défauts internes issus de la défaillance d'un composant et les défauts externes qui dépendent de l'environnement du système. Cela peut nous permettre de classer brièvement ces perturbations.

Voici donc un aperçu des principales causes de dysfonctionnement possibles pour ENVSYS. Il peut subir :

- Un dysfonctionnement des capteurs (défaut interne): un capteur peut être défaillant et, par exemple, émettre en continu, monopolisant ainsi le réseau.
- La perte éventuelle d'un des capteurs. En effet, le capteur peut être emporté par le courant (défaut externe) ou ne plus avoir d'énergie pour émettre (défaut interne). Ce cas de figure donne naissance à trois éventualités (illustrés par la figure 9.3):
  - Cas 1: Le capteur était un capteur terminal (il ne servait de relais à aucun autre agent). L'intégrité fonctionnelle du système n'est pas vraiment remise en cause : on ne perd a priori qu'une information.
  - Cas 2 : Le capteur n'est pas un agent terminal mais un autre capteur peut prendre le relais. La cohérence peut ainsi être rétablie.
  - Cas 3 : Le capteur était un agent non terminal et le réseau s'est scindé en deux réseaux autonomes. Ce cas est le plus préjudiciable et doit être anticipé lors du placement des capteurs : c'est pourquoi le système doit permettre un mode configuration afin que le spéléologue sache à tout instant combien de capteurs sont dans l'aire d'écoute de celui qui est placé.

    Remarques : Deux capteurs sont dits en accointance s'ils peuvent communiquer. Nous appelons ici accointance "utilisée" une accointance qui sert pour le transport de l'information. L'accointance "non utilisée" est une accointance qui peut être utilisée mais que le système

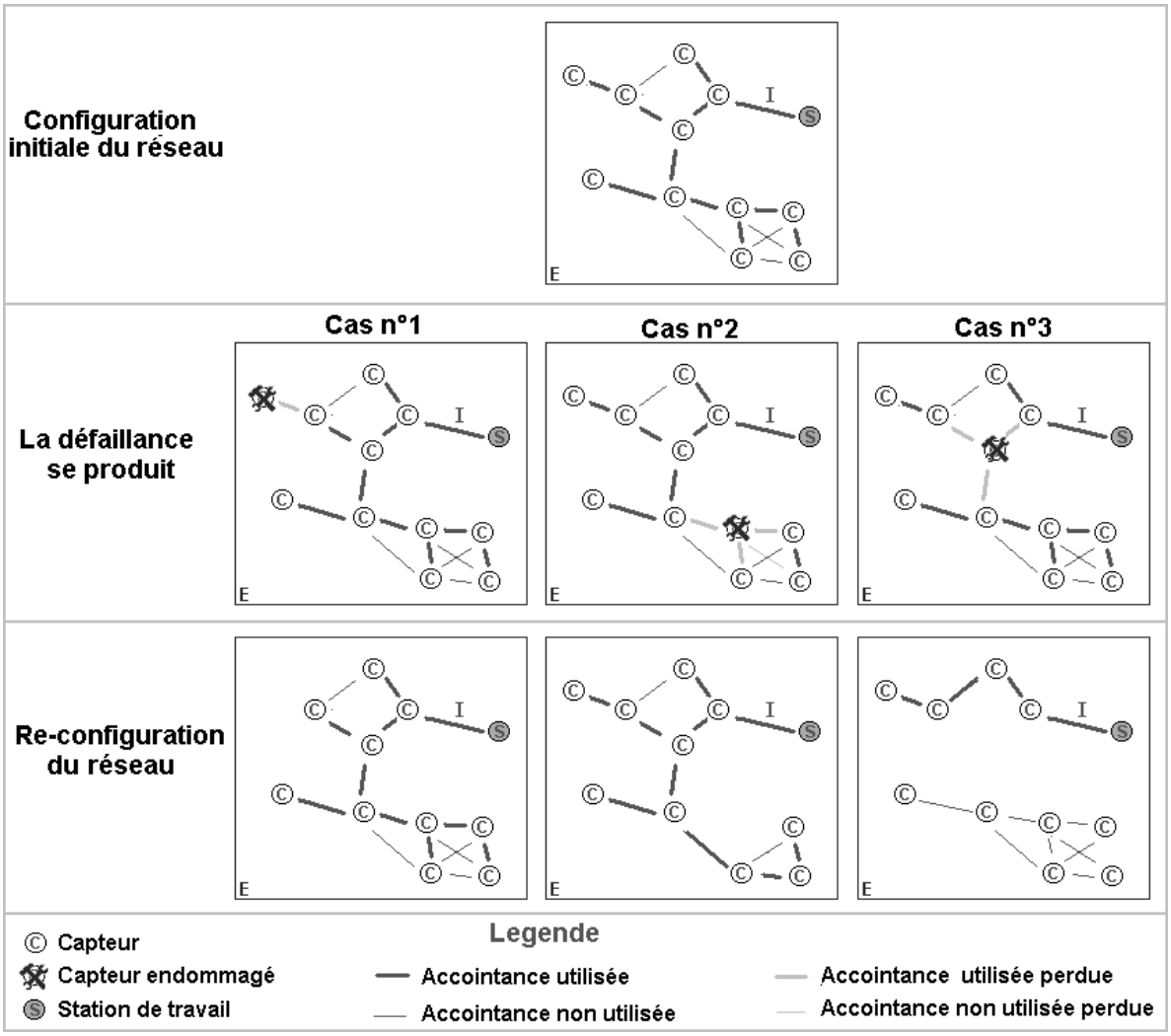


FIG. 9.3 – *Le réseau suite au disfonctionnement d'un des capteurs*

n'exploite pas. Une accointance est dite perdue lorsque que l'un des capteurs en accointance est devenu défaillant.

– Un nombre de capteurs trop important (défaut externe): le nombre de capteurs peut nuire au fonctionnement du système si, par exemple, le nombre de communications servant au contrôle de la cohérence ou de la configuration surcharge le réseau.

– Limitation du débit : ce défaut externe provient du fait que l'on utilise des ondes à basse fréquence qui limitent le débit maximum que l'on peut obtenir. Il peut nuire au bon fonctionnement du système s'il y a de nombreuses informations à échanger.

9.2.2.2 Etude des acteurs A.2

Le système est constitué d'un ensemble de capteurs et d'une station de collecte. Les acteurs agissant sur ce système sont :

- le spéléologue qui place les capteurs dans la grotte et vérifie périodiquement si des capteurs n'effectuent plus leurs mesures et nécessitent donc d'être remplacé (ou qu'un autre capteur soit ajouté).
- l'analyste qui analyse les données collectées.

Quand il place un capteur, le spéléologue doit savoir combien d'autres capteurs sont à la portée de celui qu'il place. Quand il a fini cette opération (et lorsqu'il désire vérifier que le système fonctionne bien c'est-à-dire que toutes les données nécessaires à l'analyste sont bien prélevées par les capteurs) il peut lancer une requête de contrôle.

L'analyste peut visualiser les données qui arrivent aux poste de collectes, les stocker et les détruire.

9.2.2.3 Etudier les cas d'utilisation A.3.

On identifie différents cas d'utilisation :

- Le cas d'utilisation "configuration" qui permet la création du réseau et l'ajout de capteurs,
- La cas d'utilisation "mesure" qui permet la collecte des mesures par la station de collecte. Durant ce cas d'utilisation, l'utilisateur peut :
  - gérer les données (analyste),
  - vérifier la topologie c'est-à-dire les capteurs qui sont encore en état de fonctionnement et participe à la collecte les données (spéléologue).

La figure 9.4 illustre ces cas d'utilisation et leur organisation (l'application étant de ce point de vue simple, la présence du déclenche résume l'organisation des cas d'utilisation).

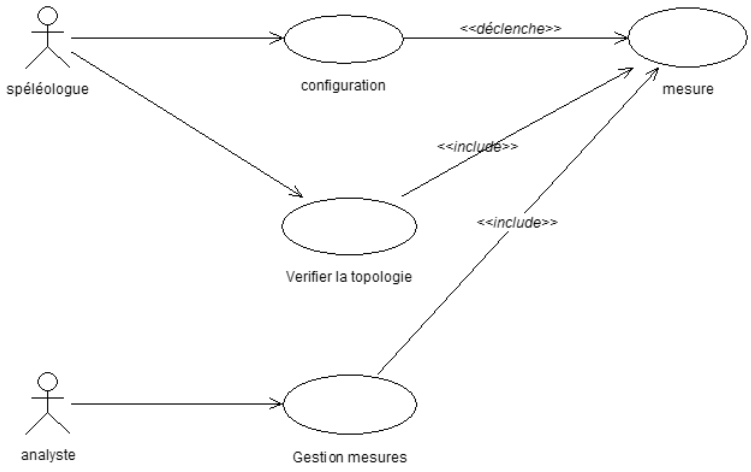


FIG. 9.4 – *Diagramme de cas d'utilisation du projet EnvSys*

9.2.2.4 Etudier les besoins en services (A.4.)

Nous présentons, dans un premier diagramme de séquence (voir figure 9.5), les besoins en service de l'acteur spéléologue lorsqu'il désire placer les capteurs.

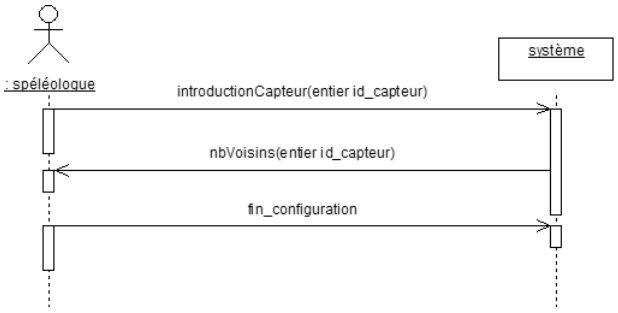


FIG. 9.5 – *Diagramme de séquence lié à la configuration de la topologie du réseau de capteurs*

Le diagramme, présenté en figure 9.6, introduit les besoins en services des acteurs "spéléologue" et "analyste" lorsque le système est utilisé pour instrumenter le réseau hydrographique souterrain.

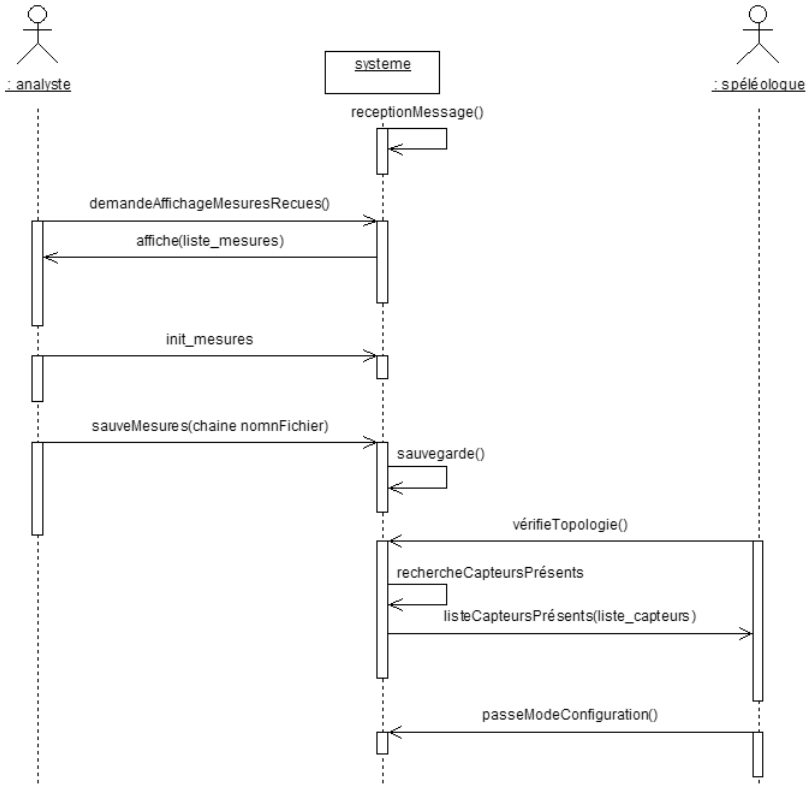


FIG. 9.6 – *Diagramme de séquence du système (les capteurs étant positionnés)*

9.2.2.5 Etude des modes de marche et d'arrêt (A.5.)

En marche normale les mesures arrivent à la station de collecte. Si f_i est la fréquence d'envoi des mesures par le capteur i alors si durant une période $T_{min_i} = 1/f_i + t_{acheminement}$ il n'y a pas de message, on s'est écarté du mode de fonctionnement nominal. Il est peut être nécessaire de rajouter des capteurs (l'analyste doit déterminer si les mesures qui manque biaiseraient l'analyse des résultats et éventuelles

décisions qui en découleraient).

Un mode de préparation a été identifié et est lié au cas d'utilisation "configuration". Durant ce laps de temps, pour pouvoir placer les capteurs, le spéléologue doit savoir, pour chaque capteur qu'il place, le nombre de ses voisins.

9.2.3 Phase d'analyse (B)

9.2.3.1 Phase de situation

L'environnement est le réseau hydrographique souterrain. L'environnement est fait de toutes les informations mesurables par les agents : dans notre cas on s'intéressera, dans un premier temps, uniquement à la température. Nos agents sont situés mais ne peuvent percevoir leur position géographique. Notre environnement, comme la majorité des environnements physiques, est déterministe, non épisodique, dynamique et continu.

Les constituants de notre système sont :

- La station de collecte : cette station reçoit les messages des capteurs qui contiennent les données. Elle permet à l'utilisateur de gérer les données ou de prendre connaissance de la topologie du réseau.
  $\Rightarrow$ Cette entité active n'est pas autonome, la station de collecte ne deviendra pas un agent.
- Les capteurs : Comme vu précédemment, le capteur doit interagir avec l'environnement pour en extraire des données. Il doit ensuite les faire parvenir à une station de collecte. Le problème majeur réside dans le fait qu'il ne dispose que d'une quantité d'énergie limitée. Cette quantité d'énergie doit lui permettre d'effectuer des mesures et d'acheminer les messages au destinataire. Pour ce faire, il nécessite l'aide des autres capteurs et doit donc lui même aider les autres capteurs à faire parvenir leur données à la station de collecte. Il doit déterminer lui même si il peut encore aider les autres capteurs ou les laisser se débrouiller eux-même.
  $\Rightarrow$ Ce sont des entités actives. Ils sont autonomes et mettent en jeu de la coopération.
  $\Rightarrow$ Ils deviendront des agents.

Le problème traité se prête particulièrement bien à l'utilisation du modèle que nous avons développé pour la gestion des communications dans les systèmes complexes physiques autonomes.

9.2.3.2 Phase individuelle (B.2.)

L'arbre des tâches (B.2.1). L'agent doit exécuter des primitives de mesures et faire parvenir les mesures à la station de collecte. Il n'est actuellement pas capable de faire parvenir sa mesure à la station de collecte.

Etablir la représentation de soi et de l'environnement (B.2.2.). L'agent possède un identifiant et connaît son niveau d'énergie. L'agent ne nécessite cependant pas d'avoir une représentation de son environnement. En effet, l'information pertinente aurait été sa position géographique, mais il n'est pas possible de l'acquérir.

Les données de l'environnement qu'il faudra acquérir et qui entraîneront des besoins en capteurs, se résument donc à la température. Ces informations ne sont pas des données de l'agent (elle ne sont pas mémorisées). Cependant, l'agent doit savoir ce dont il dispose comme capteurs (liste des perceptions possibles) : ils s'agit bien de données car sans ces informations il ne sait pas ce qu'il est capable de faire.

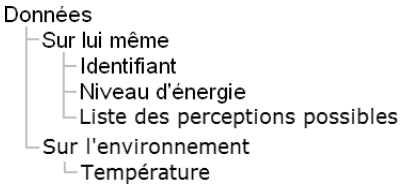


FIG. 9.7 – *Données de l'agent capteur*

Diagramme de contexte (B.2.3.). Quand un spéléologue place un capteur il doit savoir combien d'autres capteurs sont à sa portée : il est nécessaire d'avoir un périphérique d'affichage.

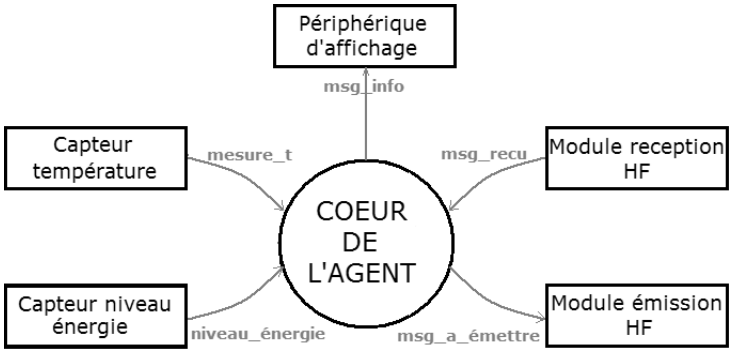


FIG. 9.8 – *Diagramme de contexte de l'agent capteur*

Etablir le comportement des agents (B.2.4.). Le comportement de l'agent est en fait très simple. Il va périodiquement faire appel à ses capteurs pour établir des mesures et les émettre via un message à destination de la station de collecte.

9.2.3.3 Phase sociale (B.3.)

Définir la connaissance des autres (B.3.1). On a besoin des autres pour acheminer les messages à la station de collecte. Il est impossible de connaître sa position géographique. On va s'orienter vers le modèle MWAC. La connaissance des autres est prise en compte au niveau de ce modèle. Ainsi connaître un autre capteur c'est avoir connaissance de son triplet $< id,rôle,groupe >$ sachant que le rôle est soit $REPRESENTANT$ soit $SIMPLE\_MEMBRE$ soit $AGENT\_DE\_LIAISON$. Un groupe est identifié par le propre identifiant de son représentant.

Cette analyse est en fait le résultat de plusieurs itérations. En effet, dans un premier temps l'organisation n'est pas définie, on ne connaît donc de nos voisins que leurs identifiants. Une fois le modèle MWAC choisi et donc l'organisation définie (voir B.3.4), un agent sera identifié par le triplet $< id,rôle,groupe >$*. Lors de l'itération*[3] *suivante sur ce processus B, l'activité B.2.2 introduira l'ajout de données sur les autres (liste des voisins, liste des groupes adjacents et liste des squelettes) ainsi que l'ajout d'une donnée sur soi (le rôle).*

Définir les primitives de communication (B.3.2). Les primitives de communication sont celle proposée par le modèle MWAC. Il s'agit d'une primitive d'envoi de messages à un destinataire et d'une

3. Un arbre de synthèse comprenant les résultats de toutes les itérations de tous processus est visible en figure 9.11

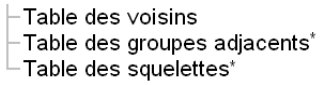


FIG. 9.9 – *Arbre des données relative à la représentation des autres*

primitive de réception de message.

Définir les actions et les perceptions collaboratives (B.3.3). Aucune action/perception collaborative en plus de ce qui a été défini dans le modèle MWAC n'est nécessaire.

Aspect organisationnel (B.3.4). Une organisation est héritée du modèle MWAC (voir §8.2.2).

Quand le système est en phase d'instrumentation, il n'y a aucun besoin supplémentaire en organisation/interaction. Par contre, en phase de configuration, il est nécessaire de définir un protocole de communication entre un agent et ses voisins. En effet, ce mode nécessite qu'un agent sache combien d'autres agents capteurs sont à sa portée. Nous définissons un protocole de communication.

La vue centralisée, qui est généralement celle du demandeur (raisonnement plus naturel pour l'homme), laisse à penser qu'il est nécessaire que le système global soit dans un mode particulier pour supporter l'ajout de capteurs. En fait, ce problème peut être complètement déporté sur l'agent : il suffit qu'il procède à une mise à jour de son voisinage et affiche sur l'écran le nombre de voisins. Avec l'utilisation du modèle MWAC il suffit que l'agent, périodiquement, supprime sa table des voisins et envoie le message $QuiSontMesVoisins$ pour mettre à jour sa table des voisins et procède à l'affichage.

A ce stade, il est donc nécessaire de procéder une nouvelle itération du processus précédent (phase individuelle (B.2.)) :

- *on dote l'agent d'un bouton afin que le spéléologue lui signale qu'il est en mode positionnement ou non (Le diagramme de contexte complet sera visible en phase de conception en figure 9.12).*
- *on ajoute aux données de l'agent une information pour mémoriser s'il est en mode positionnement ou mesure (l'arbre complet des tâches et données est visible en figure 9.11).*

9.2.3.4 Phase d'intégration (B.4)

Identifier l'influence sociale (B.4.1.). Dans un mode positionnement, l'agent doit refuser d'entrer dans un groupe car l'utilisation de $QuiSontMesVoisins$ qui parait assez naturelle, va engendrer de nombreuses ré-organisations. Ces ré-organisations entraîneront elles même des pertes d'énergie inutiles.

Définir le comportement social de l'agent (B.4.2.). On va donc, dans ce cas, dégrader (chez l'agent en positionnement) le protocole d'interaction issu de MWAC. Avec le modèle MWAC, l'agent que l'on positionne envoie $QuiSontMesVoisins$, il reçoit des messages $JeSuisUnDeTesVoisins$, il choisit un rôle et en informe ses voisins. Il nous suffit, pour corriger ce problème, de modifier le module de décision de l'agent pour qu'il ne choisisse aucun rôle : il avorte le bon déroulement du protocole. Le diagramme figurant en 9.10 montre le comportement (raisonnement et envoi de message) de l'agent. L'action décision permet d'avorter le protocole et d'afficher les informations en destination du spéléologue.

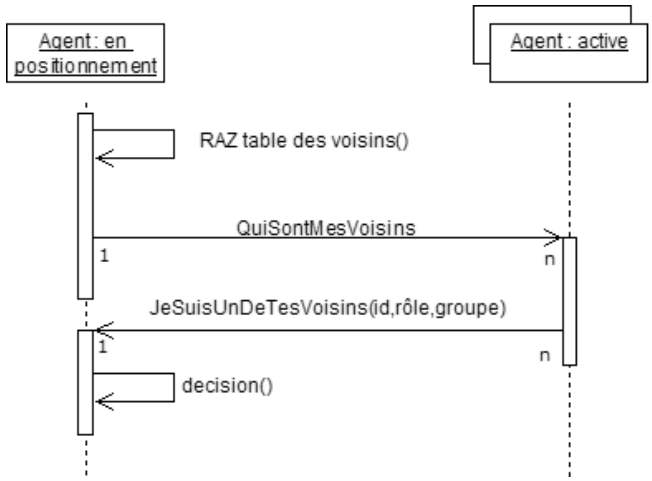


FIG. 9.10 – *Diagramme d'interaction pour le positionnement de l'agent.*

9.2.3.5 Avant-propos : synthèse des tâches et données de l'agents

La figure 9.11 rappelle l'ensemble des tâches et des données de l'agent.

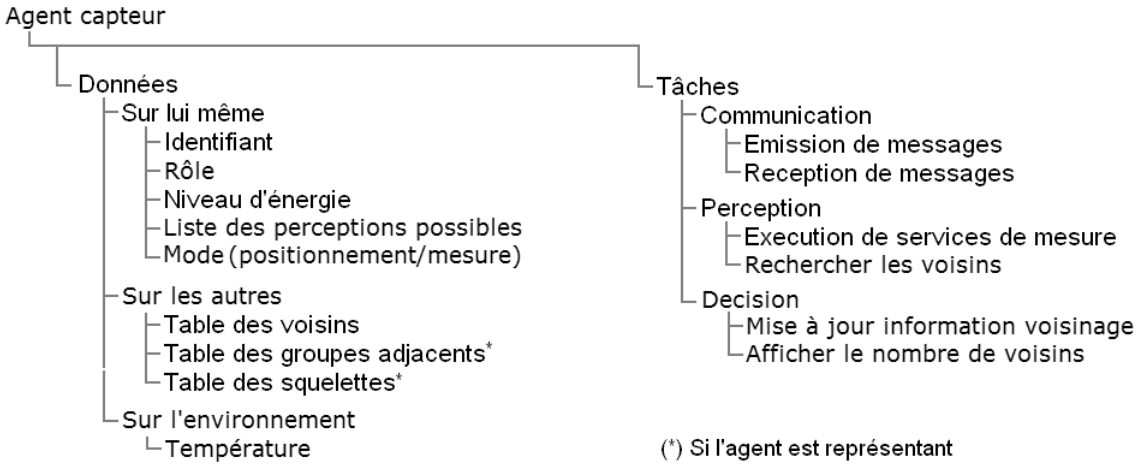


FIG. 9.11 – *Synthèse des tâches est données de l'agent*

9.2.4 Phase de conception (C)

Nous préparons ici la construction des agents issus de l'analyse. Il est à noter que l'implémentation a été réalisée de façon manuelle. Les phases $C_2 - C_4$ décrites dans la méthode prévoient l'utilisation de l'outil de développement, non encore opérationnel lors du développement de l'application. Dans cette section, nous allons donc présenter les éléments sur la conception du système qui ont un intérêt pour l'implémentation présentée dans le chapitre suivant.

9.2.4.1 Définition du contexte (C.1.)

Choisir une architecture pour les différents agents (C.1.1.). Nous utilisons l'archétype d'agent de l'intergiciel MWAC (voir §8.4.2). En effet, nous ne pouvons pas nous contenter d'utiliser la déclinaison "composant" de l'intergiciel: le composant intergiciel ne permet ni d'accéder aux données internes au composant (rôle, liste de voisins etc.) ni de modifier les protocoles d'interactions pour les spécialiser à cette application (voir §8.4.2). L'utilisation de la déclinaison "archétype d'agent" est bien plus pertinente.

Etablir un diagramme de contexte par type d'agent (C.1.2.). On reprend le diagramme de contexte et on procède aux choix technologiques.

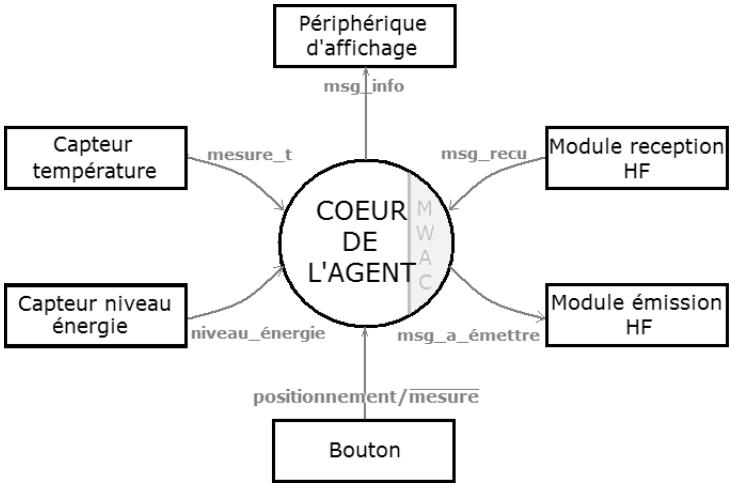


FIG. 9.12 – *Diagramme de contexte (utilisation de l'archétype d'agent MWAC)*

On choisit chacun des organes physiques :

- Le capteur de température est une sonde PT100,
- L'afficheur un écran LCD 2 lignes / 16 colonnes,
- Le bouton sera du type "bouton pressoir".

TAB. 9.1 – *Spécification des interfaces*

Information	Spécification
msg_info	Message à afficher : chaîne de 32 caractères codés sur 8 bits
mesure_t	Température en degré Celsius codée en flottant IEEE simple précision (4 octets)
niveau_énergie	Entier codé sur 8 octets (pourcentage)
positionnement/mesure	Détection du changement d'état sur front montant, nécessite la mémorisation de l'état précédent
msg_recu	Suite de bits respectant le format suivant: émetteur (1 octet), destinataire (1 octet), taille_données (1 octet), données (1-256octets).
msg_emi	Suite de bits respectant le format suivant : émetteur (1 octet), destinataire (1 octet), taille_données (1 octet), données (1-256octets).

9.2.5 Conclusion

L'application EnvSys a permis l'utilisation de la méthode DIAMOND (présentée en chapitre 5) pour développer une solution au problème de la gestion adaptative de la communication pour l'instrumentation sans fil d'un réseau hydrographique souterrain. Elle a permis, de plus, l'utilisation du modèle MWAC (présenté en chapitre 8) pour assurer une gestion adaptative de la communication entre les capteurs, modélisés par des agents.

Les étapes de *recueil des besoins* et d'*analyse* ont permis l'élaboration d'une solution multi-agents. Du fait d'une implémentation manuelle, une partie des activités de l'étape de *conception générique* et l'étape d'*implémentation* qui prévoient l'utilisation d'un outil spécifique de développement n'ont pas été présentés. Cependant, l'implémentation réelle de l'application est donnée dans la section suivante.

Date création: 23/01/2005 **Date de mise-à-jour:** 01/01/2005 **Responsable:** Jean-Paul Jamont **Version:** 1.0

GLOSSAIRE CONTEXTUEL

TERME	DESCRIPTION
Analyste *Voir A.1, acteur analyste*	Dans EnvSys, l'analyste est chargé d'exploiter les mesures. C'est donc lui qui va décider des paramètres a mesurer.
Bus de terrain	Terme employé dans l'industrie pour qualifier des systèmes d'interconnexion d'appareils de mesure, de capteurs, d'actionneurs etc. Il est utilisé en opposition a un bus informatique car il est beaucoup plus simple du fait des faibles ressources numériques embarquées.
Capteur *Voir A.1.*	Dans notre contexte nous appelons capteurs le dispositif complet de mesures et non seulement la partie qui s'occupe de l'acquisition d'une grandeur de l'environnement.
Capteur intelligent	Un capteur est un dispositif qui transforme l'état d'une grandeur physique observée en une grandeur utilisable. Il est dit intelligent si il intègre des fonctionnalités tels que : sortie sur `bus de terrain`, étalonnage automatique, fonctions d'auto-contrôle, fonctions configurable de traitement du signal etc.
Instrumentation	L'instrumentation est une technique de mise en œuvre d'appareils de mesures, d'actionneurs, de capteurs, de contrôleurs en vue de créer un système d'acquisition de données ou de commande. Dans le contexte d'EnvSys, il s'agit d'acquérir des données dans un réseau hydrographique souterrain.
Spéléologue *Voir A.1, acteur spéléologue*	La spéléologue est la personne dont l'activité qui consiste à repérer, explorer, étudier et cartographier les cavités souterraines. Dans EnvSys, c'est la personne habilitée à placer les capteurs dans le réseau hydrographique souterrain.
Station de collecte *Voir A.1*	Dispositif type PC qui a pour objectif de recueillir l'intégralité des mesures et de les stoker. Il permet aussi de d'identifier quels sont les capteurs qui compose le réseau (pour le spéléologue).

FIG. 9.13 – *Fiche de glossaire contextuel*

Chapitre 10

Simulation et implémentation

Comme c'est fréquent en science, j'ai trouvé, grâce à ces données, des réponses à des questions que je ne m'étais pas posées initialement.

André Langaney

[A propos de pollution atmosphérique]
C'est plus parlant de voir une plante en train de crever qu'un chiffre s'affichant sur un écran. On fait tout de suite le rapport avec ses poumons.

Jean-Pierre Le Garrec

Nous introduisons dans ce chapitre la plate-forme de simulation qui nous a permis d'évaluer notre solution. Nous présentons ensuite notre implémentation des agents ainsi que les résultats de simulation.

10.1 La plate-forme de simulation

Nous allons dans ce chapitre décrire la phase de simulation et introduire l'embarquement des agents sur des cartes autonomes.

10.1.1 Raisons d'être de cette plate-forme

La plate-forme de simulation (voir figure 10.1) va nous permettre d'expérimenter notre approche et les solutions logicielles que nous avons choisies pour implémenter ces agents.

Le passage par la simulation est indispensable car elle permet de mettre à jour d'éventuels problèmes qui seront plus faciles à traiter, en terme de programmation, sur le simulateur que directement sur les systèmes embarqués. De plus, la simulation permet de mettre en place facilement des mesures. Les performances de notre système pourront ainsi être comparées avec d'autres solutions. Elle permet aussi

de quantifier l'émergence inférée par l'approche système multi-agents : mieux apprécier le phénomène d'auto-organisation.

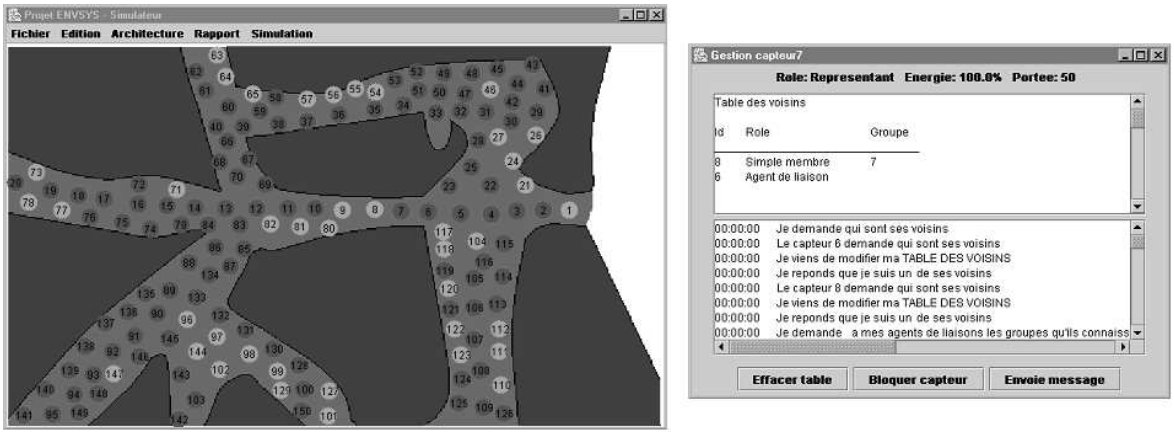


FIG. 10.1 – *Interface graphique (fenêtre principale) du simulateur*

Les fonctionnalités que doit fournir la plate-forme sont :

- créer l'environnement c'est-à-dire définir la position géographique des capteurs,
- mesurer et visualiser les performances des architectures implémentées et ce à travers divers critères comme le rendement ou l'encombrement mémoire des tables et autres listes,
- créer des fautes effaçant les données d'un agent sur ses voisins ou en le bloquant afin d'étudier la tolérance aux fautes de notre système,
- créer des déplacements afin d'étudier l'impact de la mobilité sur les performances
- espionner un capteur c'est-à-dire avoir accès à ses attributs (niveau d'énergie, rôle, portée), à ses diverses tables de routage et avoir un historique des messages émis et reçus ainsi que des actions menées.

10.1.2 Structure logicielle

Notre outil peut être vu de manière simple comme composé de deux types d'objets (voir figure 10.2). Le simulateur de réseau (SimReseau) simule le comportement. Il n'est pas nécessaire à l'utilisateur de connaître son fonctionnement précis. Le simulateur de capteur (SimCapteur) simule le comportement d'un capteur et possède donc ses propres modèles et architectures. Tous les simulateurs de capteurs sont dotés des mêmes fonctionnalités de communication. Ils envoient leurs requêtes au composant SimRéseau qui y répondra suivant l'environnement simulé. La figure illustre cette architecture.

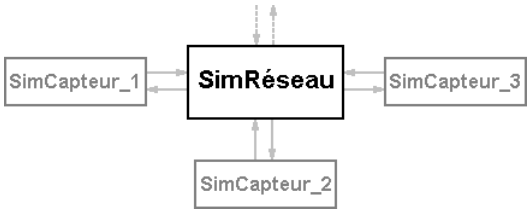


FIG. 10.2 – *Plugging des capteurs sur le simulateur de réseau*

Le simulateur a été réalisé en Java et conçu de manière à ce qu'une personne qui désire ajouter une nouvelle architecture d'agent n'ait pas à connaître le fonctionnement du simulateur de réseau. Il suffit, tout d'abord, de comprendre le mécanisme de notification d'événements. Ces événements permettent le rafraîchissement des informations affichées sur le capteur et de mesurer les performances de l'implantation de l'architecture. Enfin, il est nécessaire de comprendre le fonctionnement de la classe abstraite appelée *Capteur* qui propose les primitives minimales d'envoi de trames et d'affichage d'informations (voir figure 10.3).

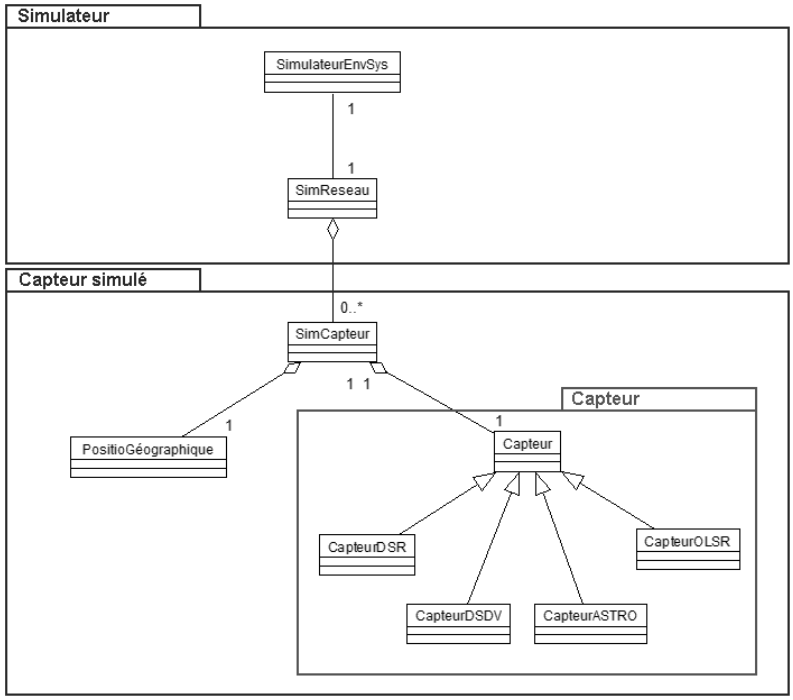


FIG. 10.3 – *Diagramme UML de la plate-forme de simulation*

10.1.3 Création d'un nouveau capteur

Pour créer un nouveau type de capteur, il suffit donc de créer une nouvelle classe qui hérite de la classe capteur. Le constructeur de l'objet doit au moins contenir une référence vers un objet de type SimEnvSys (pour l'interface graphique), une référence vers un objet SimRéseau et un identifiant du capteur (un entier). Après avoir spécialisé les différentes fonctions héritées de la classe Capteur (voir figure 10.4) on crée le corps du capteur (méthode run car on réalise des threads).

10.1.4 Edition de scénario

Afin de produire des comparaisons exploitables, il est nécessaire de s'assurer que, quel que soit le type de capteur testé, les mêmes événements ont lieu aux mêmes instants. De plus, il peut être intéressant de programmer le déclenchement d'évènements particuliers comme les pannes ou les déplacements. Ainsi nous avons développé un langage de script pour créer les scénarios. En réalité il y a deux niveaux de scripts.

```
abstract public class Capteur extends Thread implements Serializable

// Informations utiles sur le capteur
protected int       typeCapteur;    // architecture
protected int       id;             // l'identifiant
protected float     energie;        // le niveau d'energie
protected int       portee;         // Portée en pixels
protected int       role;           // Le role

// Gestion du thread
public void requetePause()          // Stand bye
public void requeteReveil()         // Reveil du capteur
public void requeteStop()           // Arret du capteur
protected boolean dort(int duree)   // Endort le capteur pour une certaine durée

// Accesseurs
public int getId()
public float getEnergie()
public int getPortee()
public int getRole()
public static String roleVersChaine(int i)
public boolean getEstBloque()
public int getTypeArchitecture()
public void setPortee(int nouvPortee)

// Primitive d'émission d'une trame
public void emetTrame(int dest,Object msg)

// A implementer
public int identifieMessage(Object msg)
public String donneesVersChaine()
public long encombrementTableRoutage()

// Pour les fautes
public void fauteBloqueDebloqueCapteur(){}
public void fauteEffaceTableRoutage(){}
```

FIG. 10.4 – *La classe abstraite Capteur*

Le script de haut niveau contient des instructions évoluées comme des boucles, le déplacement d'un capteur d'un point A à un point B en une durée t. Lors de la compilation ces instructions sont décomposées en une suite de primitives datées ne nécessitant que peu de temps d'exécution (le script de bas niveau). Nous avons défini une syntaxe inspirée des langages orientées objets. Une illustration de ces scripts figure en 10.5. Ce scénario simule un fonctionnement uni-directionnel pour lequel un des capteurs 2,3,4 et 5 émet un message au capteur 1.

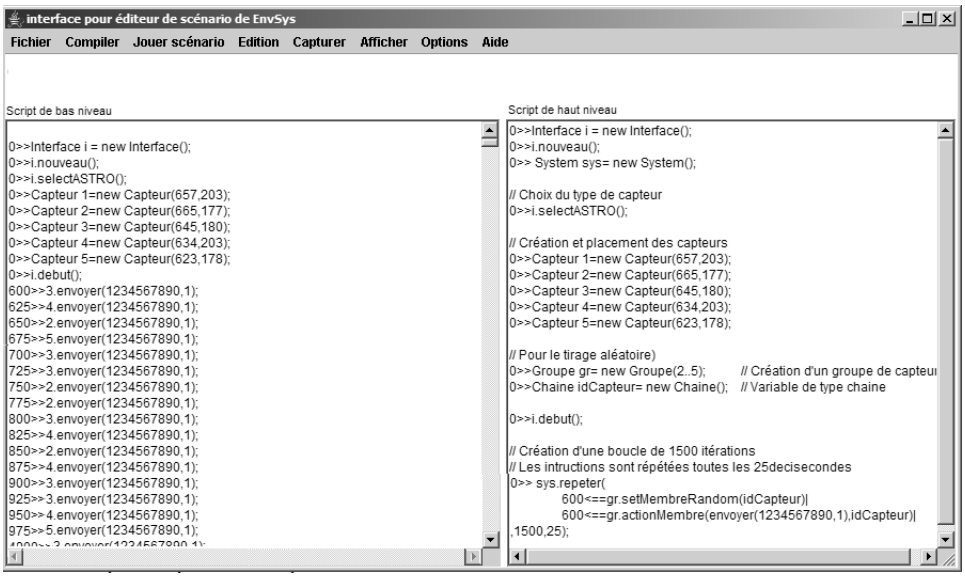


FIG. 10.5 – *Exemple de création d'un scénario illustrant un fonctionnement uni-directionnel*

Une interface graphique a été créée pour assister l'utilisateur dans la création de scénario notamment pour ce qui concerne les déplacements.

10.2 L'implémentation réelle

10.2.1 L'architecture d'agent embarqué

A ce jour, nous avons implémenté un système minimal afin de démontrer la faisabilité de notre approche. Une station de travail collecte les informations émises par les divers capteurs. L'architecture que nous avons choisie pour les capteurs est une version dégradée du modèle OSI (Open Systems Interconnection). Elle est souvent utilisée dans l'informatique embarquée et est organisée en trois couches : une couche physique, une couche liaison et une couche applicative (voir figure 10.6).

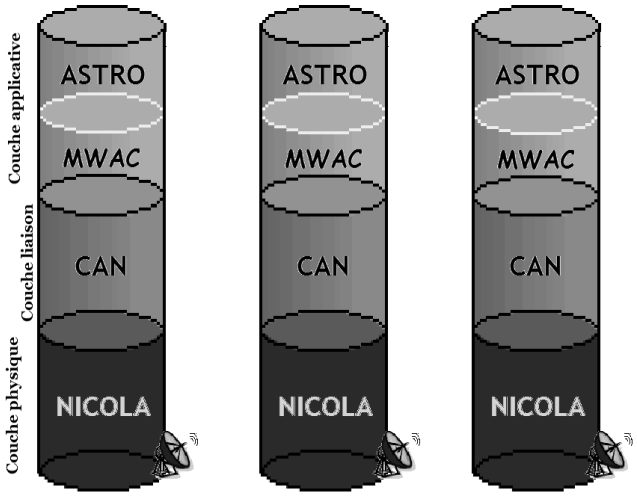


FIG. 10.6 – *Agent capteur*

La couche physique que nous utilisons est celle du système NICOLA [Graham, 1999]. Ce système de communication est utilisé par les secours spéléologiques français. Cette couche est implémentée grâce à un DSP (Digital Signal Processor). L'intérêt d'utiliser ce processeur réside dans la flexibilité qu'il offre quant à son implémentation.
Notre couche liaison utilise la version wireless du protocole CAN (Controller Area Network) qui a été éprouvée depuis longtemps dans l'industrie automobile.
La couche applicative est construite avec les agents qui constituent le système.

10.2.2 Les plates-formes cibles

Les agents ont été portés sur deux plates-formes différentes (voir figure 10.7) : une carte basée sur un siemens C515C et une carte à base d'un Microchip PIC18F452.

Dans un premier temps, nous nous sommes orientés vers des cartes à bases de Siemens C515C afin de procéder à une étude de faisabilité. Ces cartes fournissent un noyau temps réel. Le noyau temps réel KR-51 permet de créer une application multitâches utilisant le microcontrôleur C515C. Il nous était alors

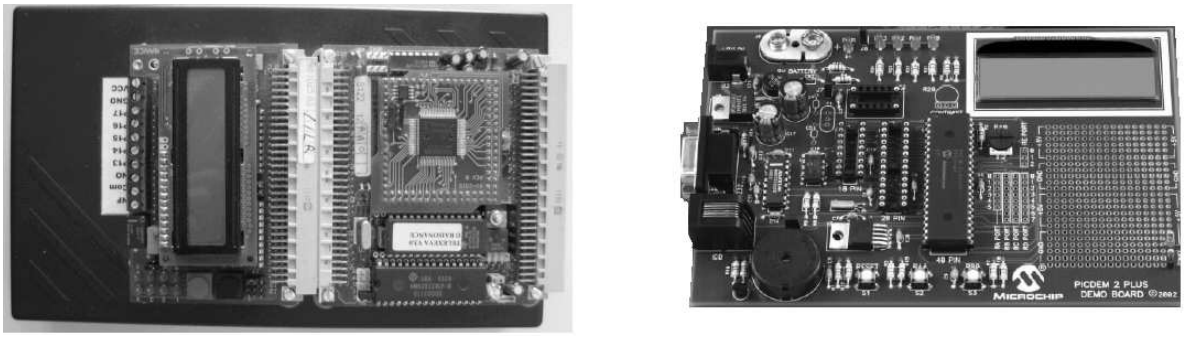


FIG. 10.7 – *Plateformes cibles : carte à base de C515C et carte à base de 18F452*

facile d'associer à une tâche une compétence et ainsi implémenter facilement le parallélisme inhérents aux agents pour satisfaire les contraintes temps réel.

Dans un second temps, la faisabilité établie sur des cartes à base de Siemens C515C (voir figure 10.8), nous nous sommes orientés sur une carte à base d'un PIC. Cette grande famille est constituée de microcontrôleurs à faibles coûts. Des modifications mineures de programme permettent d'adapter l'application d'une cible à une autre d'une même famille de microcontrôleur. L'intérêt d'implémenter notre solution sur ce composant réside en un choix technologique établi avec les contraintes de traitement de signal : la cible finale [1] sera une carte dsPIC qui intègre un microcontrôleur et un DSP.

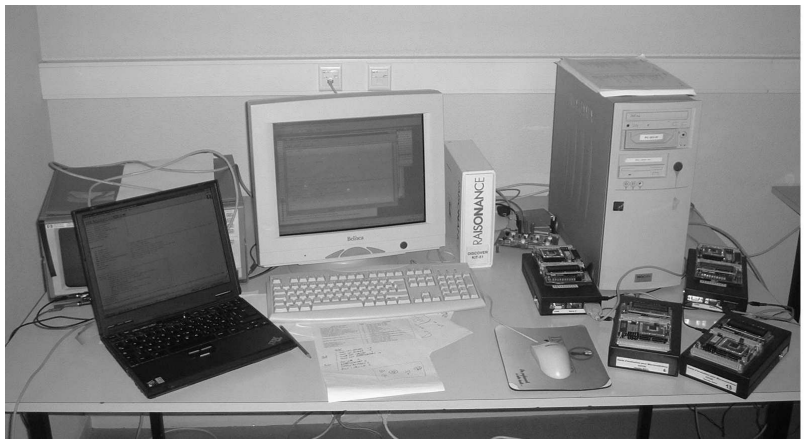


FIG. 10.8 – *Manipulation avec le matériel*

10.2.3 Utilisation de l'intergiciel.

Dans cette application, les agents doivent transmettre périodiquement des mesures à une station de collecte d'informations. Pour ce faire les agents doivent utiliser un package `WCommunication`, écrit en langage Java et traduit en C++ car un grand nombre de plate-formes physiques utilisent ce langage. Ce package contient essentiellement deux classes abstraites (`Identifier` et `Message`) et les classes `Communication` et `BitField`. Dans la classe abstraite `Message` le concepteur doit

1. Carte en cours de conception.

implémenter les primitives pour convertir un message en un champ de bits. (`BitField MessageToBitField(Message m)` et la fonction réciproque `Message BitFieldToMessage(BitField b)`). Dans la classe abstraite `Identifier`, le concepteur du système doit préciser le type de l'identifiant et implémenter les primitives `BitField IdentifierToBitField` et `Message BitFieldToMessage(BitField b)`. Les primitives pour convertir un identifiant en un champ de bits sont aussi à implémenter. La classe `Communication` contient une liste de couples `(Identifier,Message)` pour l'émission et la réception des messages. Cette liste est privée et doit être accédée via les méthodes `Bool SendMessage(Identifiant, Message)` et `CoupleIdentifierMessage ReceiveMessage()`.
Le package doit être connecté au système d'exploitation. Ce dernier doit fournir le niveau d'énergie disponible (primitive `SetBatteryLevel(Float l)` à la classe communication `Communication` et les champs de bit qui arrivent. D'un autre coté, l'intergiciel donne au système d'exploitation des champs de bits à émettre. Cela se fait par un appel à la méthode `BitField GetBitFieldToSend()`.

10.2.4 Organisation du code

Les environnements de programmation disponibles pour ces plates-formes cibles ont imposé le choix du langage C pour créer les agents. Le code produit (dans le cas du microcontrôleur PIC) est organisé comme l'illustre la figure 10.9.

On retrouve dans ce digramme l'intergiciel MWAC et l'utilisation de la bibliothèque CAN qui permet d'utiliser le protocole de niveau liaison du même nom. Le module lcdpd2en permet l'utilisation de l'afficheur. Le module AGENT regroupe la représentation du monde, la partie décisionnelle.

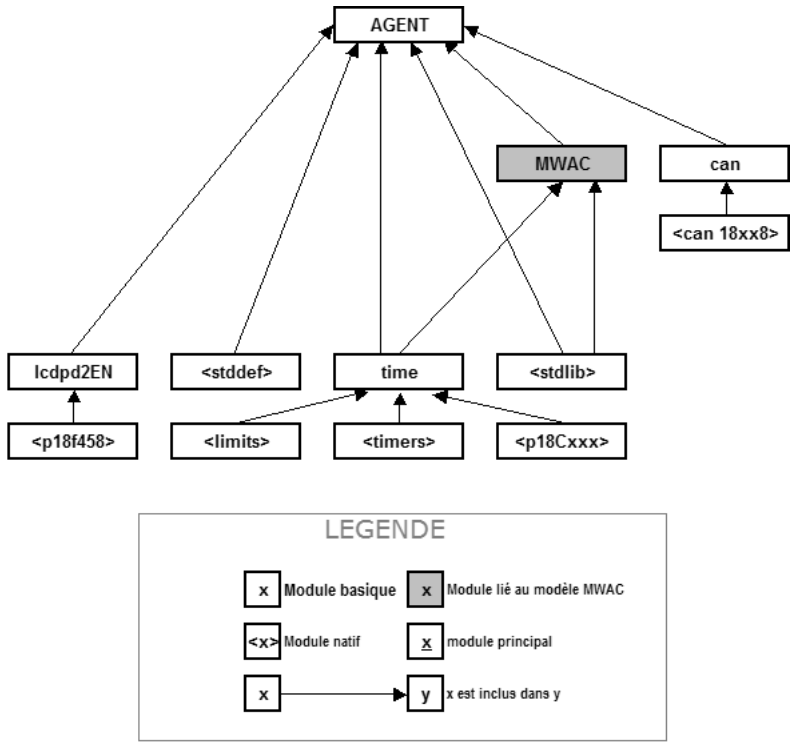


FIG. 10.9 – *Graphe des inclusions entre modules*

10.3 Analyse des performances de notre solution

Nous avons comparé notre système multi-agents avec trois protocoles ad-hoc. Ces protocoles sont DSDV et deux variantes de DSR. La première de ces variantes est la version naturelle et première de ce protocole (nous l'appellerons DSR) et la seconde la version avec maintenance des routes (notée DSR-ROUTAGE). Cependant, les résultats obtenus avec DSDV étant les plus mauvais, ils ne seront pas présentés.

10.3.1 Définition du rendement.

Nous présentons dans cette section la définition du rendement que nous avons adoptée puis une illustration du calcul de ce rendement.

10.3.1.1 Définition.

Nous appellerons dans la suite *rendement du protocole* à instant t le rapport entre le volume utile théorique des différents messages envoyés (en supposant qu'ils aient pris les routes optimales) que divise le volume total transmis par le protocole. Le système qui aura le plus haut rendement aura donc transmis moins de données et donc économisé plus d'énergie. Cette définition du rendement permet de pénaliser correctement les protocoles qui n'utilisent pas les routes optimales. Il est à noter que les protocoles DSR et DSR-ROUTAGE utilisent des routes généralement optimales. Le routage effectué par l'auto-organisation, obtenue avec nos agents capteurs, utilise quand à lui souvent des routes non optimales : en effet, dans notre approche c'est la succession de groupes qui est optimale et non la succession de capteurs.

Il est important de prendre en compte qu'avec notre approche tous les messages sont acquittés contrairement au protocole DSR. L'acquittement des messages intervient dans le maintien de la cohérence de l'auto-organisation au même titre que l'écoute indiscrète.

10.3.1.2 Illustration du calcul du rendement.

Définition: On appelle *volume utile* d'un message le volume qu'occupe les données transportées et *volume de configuration* d'un message le volume des informations ajoutées pour l'acheminement du message (identifiant destinataire, identifiant de type de message etc.).

Notations: Soit v_u le volume utile d'une trame, V_u le volume utile des messages échangés pour l'acheminement du message de l'émetteur au destinataire. De la même manière nous utiliserons les notations v_c et V_c pour exprimer le volume de configuration ainsi que v_t et V_t pour exprimer le volume total. On a v_t = v_u + v_c et V_t = V_u + V_c + V_o. Soit r le rendement. On a défini r = V_u(RouteOptimale) / V_t(Protocole).

Cas pratique : Le capteur s_1 veut transmettre une température à s_4. La température est codée sur 4 octets mais la trame envoyée prends 10 octets. On a v_u=4 octets et v_c=6 octets soit v_u=10 octets.

Le protocole DSR: La route proposée est la route optimale (c'est quasiment toujours le cas) qui correspond à la succession de capteurs (s_1,s_2,s_3,s_4). DSR a perdu 50 octets pour trouver cette route. Son rendement après l'envoi d'un message sera r_{dsr} = (4 * 4) / (50 + 4*(4+6)) = 17.7%. Si on envoie deux messages on aura pour le protocole DSR un r_{dsr} = (2*(4 * 4)) / [2*50 + 2*(4*(4+6))] = 17.7% tandis

que pour le protocole DSR_ROUTAGE nous obtiendrons $r_{dsrRoutage}$ = (2*(4 * 4)) / [50 + 2*(4*(4+6))] = 22.8%.

Avec MWAC: La succession de capteurs proposée est (s_1,s_5,s_6,s_3,s_4). Admettons que l'organisation a nécessité 40 octets pour se mettre en place et 20 octets pour trouver le destinataire (le volume de la requête de recherche prends moins de place). Après l'émission du message on aura transmis V_t = (4*4) / [40 + 20 + (5*(4+6)] soit 14.5%. Apres deux messages on aura V_t = 2*(4 * 4) / [40 + 20 + 2*(5*(4+6)] = 20%.

10.3.2 Résultats

Pour ce cas d'étude nous nous placerons dans le contexte du projet EnvSys c'est-à-dire celui d'un protocole uni-directionnel. En effet, les capteurs veulent effectuer des mesures et les faire parvenir à la station de travail qui collecte les informations. Pour cette expérience toutes les cinq secondes deux capteurs transmettent leurs données. Ces messages contiennent un octet qui décrit le type de mesure et quatre octets qui codent l'information mesurée. Les mêmes scénarios sont appliqués aux protocoles. Ces mesures ont été effectuées pour 50, 100, 150, 200, 250 et 300 capteurs. La figure 10.10 donne les rendements mesurés pour ces différentes configurations.

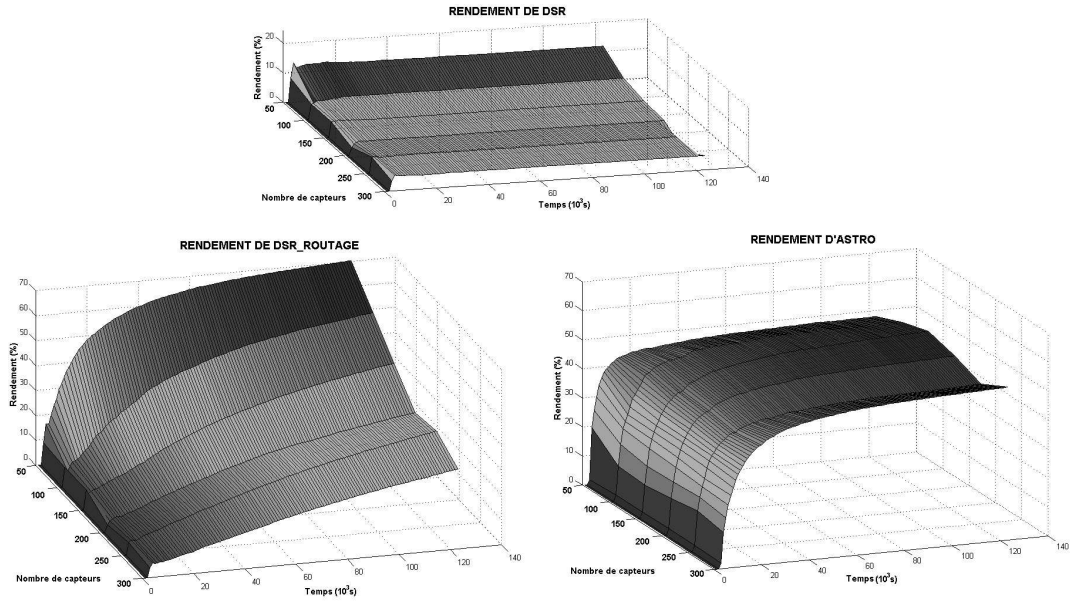


FIG. 10.10 – *Performances de DSR, DSR-ROUTAGE et de notre approche multi-agents*

Nous pouvons voir que DSR a bien un moins bon rendement que DSR-ROUTAGE. Dans ce cas d'utilisation DSR-ROUTAGE se comporte bien car ou bout d'un certain temps, aux perturbations près, toutes les routes auront été mémorisées étant donné que nous sommes dans un cas uni-directionnel. L'avantage de l'approche multi-agents dans ce cas d'utilisation est qu'elle permet rapidement d'atteindre un bon rendement. On peut à juste titre supposer que dans un cas d'utilisation réelle (fréquence d'envoi des mesures beaucoup plus faible) les capteurs auraient épuisé leur énergie avant nos capteurs utilisant l'intergiciel MWAC. Le rendement de DSR n'aurait d'ailleurs ainsi jamais été supérieur à celui de l'ap-

proche multi-agents que nous avons menée. Quand la simulation commence les agents s'auto-organisent. Pour auto-organiser (50,100,150,200,250,300) capteurs il aura fallu échanger aux capteurs un total cumulé de (3800,6950,13900,16220,23870,24790) octets. Ce volume n'est donné qu'à titre indicatif car d'une auto-organisation à l'autre les volumes sont rarement les mêmes.

Dans ce cas de figure nous constatons que notre approche multi-agents a un bien meilleur rendement et ce malgré le biais de volume de configuration utilisé pour auto-organiser les agents. En effet, contrairement à DSR, l'inondation dans avec notre approche est très localisée (au plus tout les représentants et un des agents de liaison adjacent).

10.3.2.1 Synthèse de nos comparaisons

Nous avons vu que notre approche était particulièrement adaptée pour notre problème. Nous avons dressé un comparatif (tableau 10.1) de notre protocole avec DSR, DSR-Routage et DSDV dans le contexte d'EnvSys en modulant les notes attribuées par des remarques critiques de notre propre notation.

TAB. 10.1 – *Tableau comparatif des performances des différentes solutions testées pour le routage des communications dans EnvSys*

Critère	DSDV	DSR	DSR-ROUTAGE	MWAC
Optimalité des routes (1)	* * * * *	* * * * *	* * * * *	* * *
Tolérance aux pannes (2)	*	* * * * *	* * * * *	* * * *
Rendement du protocole (3)	Lié à la fréquence de rafraîchissement des tables. Mauvais rendement dans tous les cas.	Moyen	Moyen, bon dans le cas de protocoles unidirectionnels	Bon en général, très bon dans le cas de grand réseaux en grappes.
Adéquation pour EnvSys (4)	*	* *	* * *	* * * *

Remarques :

1. L'optimalité de notre solution est en terme de groupes : elle ne propose pas la meilleure succession de stations mais la meilleure succession de groupes. Cependant, l'impact est négligeable sur le rendement (voir figure 10.10). Le gain en énergie est donc bien réel : la perte énergétique causée par la non optimalité de la route prise lors de l'acheminement de la mesure est moins important que le gain obtenu lors de la recherche de routes.
2. La tolérance aux pannes ne peut jamais être meilleure que dans le cas de DSR car ce protocole est purement réactif : il n'utilise aucune table de mémorisation. Cependant, la présence de panne ne nuit pas à la performance de notre système.
3. Il est difficile d'apprécier le rendement car il est fonction de divers paramètres tel que le volume utile des messages transportés (étant donné le caractère non optimal des routes proposées par notre approche, l'envoi d'une collection de mesures pourrait par exemple être plus pénalisant), la fréquence d'émission des messages, du nombre moyen de stations dans l'aire d'émission/réception etc. Cependant, si on se réfère au cahier des charge de EnvSys notre protocole a un très bon rendement au vue de la solution préalablement retenue.

4. Ce critère consiste à mesurer l'adéquation entre les attentes du projet EnvSys (gestion des dépenses énergétiques) : il consiste en un classement de ces quatre solutions.

10.3.3 Conclusion

Nous avons présenté, dans un premier temps, notre plate-forme de simulation qui nous a permis de présenter une évaluation comparative de notre solution. Notre approche a en effet été comparée a deux protocoles de routage traditionnels : DSR et DSDV. Les résultats permettent d'entrevoir le gain en rendement de notre approche et donc un gain en énergie.

Les deux implémentations réelles à base de microcontrôleurs siemens C515C et d'un Microchip PIC18F452 ont été présentés. Chacune des cartes est un agent de notre système conformément à l'analyse faite dans le chapitre précédent.

Conclusion

Dans un premier chapitre nous avons présenté le projet EnvSys et son analyse selon la méthode DIAMOND.

Dans un second chapitre nous avons présenté notre solution basée sur le modèle MWAC. Notre solution est beaucoup plus performante que le protocole DSDV (solution similaire à RIP (Routing Internet Protocol) qui était un des candidats initiaux) quel que soit le critère choisi, même s'il est énergétique (l'énergie dépensée pour émettre un message étant quasiment proportionnelle à son volume). Les résultats sont d'autant plus positifs que si on avait demandé aux agents utilisant DSDV d'acquitter les messages reçus (ce qui semble logique dans une application industrielle), ce dernier aurait encore plus souffert de la comparaison. Cependant, on sait de DSDV qu'il a été un des premiers protocoles ad-hoc et qu'il a été, de nos jours, abandonné. Cependant, bon nombre de protocoles sont issus ou se sont inspirés de ce dernier (AODV, CGSR, GSR). Nous avons donc décidé, au vu de ces résultats, d'implanter DSR qui sert de comparaisons à de nombreux protocoles[2]. La comparaison des performances de notre solution et du protocole a été discuté et il en sort que notre solution, dans le cadre du projet EnvSys, est plus adaptée.

Cependant, une limite subsiste! D'autres analyses que nous avons menées nous ont montré que nos performance, dans le cas d'un réseau où tous les capteurs sont en ligne droite avec pour uniques voisins deux autres capteurs, peuvent être mauvaises : on perd beaucoup de temps à former l'organisation (dans le cas ou tout les stations démarrent à un même instant t). En ce sens, le protocole doit être amélioré pour traiter ce cas particulier pouvant survenir dans d'autres applications.

2. Nous n'avions pas, à l'époque de l'élaboration de l'architecture de capteur qui allait nous servir comparatif de performances, toutes les données sur CGSR

cinquième partie

EXTENSIONS ET PERSPECTIVES

Introduction

Notre travail a donné d'ores et déjà lieu à des extensions et des développements qui ouvrent des perspectives riches tant au niveau méthodologique qu'applicatif.

Nous présentons dans un premier chapitre les perspectives en terme d'outils. L'outil MASC (MultiAgent System Codesign) qui est l'outil associé à la méthode DIAMOND sera présenté. Cet outil pour l'instant ne couvre que la phase de conception détaillée et une partie de la phase d'implémentation. Le second outil abordé est le simulateur pour les systèmes sans fil communicants qui a permis d'évaluer le modèle MWAC et notre solution dans le cadre d'EnvSys. Cet outil, comme nous le verrons, pourra lui aussi évoluer pour permettre de mieux simuler les systèmes complexes physiques.

Le second chapitre présentera une nouvelle application que nous allons traiter avec la méthode DIAMOND. Cette application servira, comme nous le verrons, à assister les arbitres de compétition de danse sportive à gérer ces évènements sportifs. Elle ouvre des perspectives pour l'utilisation de DIAMOND dans le domaine des systèmes d'informations coopératifs.

Chapitre 11

Les outils

Ce chapitre traite des perspectives en terme d'outil. Nous commençons par aborder l'outil MASC (MultiAgent Systems Construction) que nous avons développé. Cet outil n'ayant jamais été présenté, nous insistons tout d'abord sur les fonctionnalités qu'il doit offrir, puis sur la gestion documentaire qu'il doit intégrer dans une perspective de démarche qualité. Nous traitons d'abord la phase de conception détaillée puis la génération de code (parties les plus abouties).

Dans une seconde partie nous présentons les extensions que nous souhaitons apporter pour notre simulateur de systèmes complexes physiques ouverts : un simulateur hybride logiciel/matériel.

11.1 L'outil MASC

11.1.1 Avant-propos

Cette section présente les objectifs de notre outil qui doit entre autres permettre de:

- Couvrir toutes les phases de la méthode. Afin d'avoir un tout homogène, il est nécessaire de fournir un guide complet de conception. Ainsi les informations saisies lors de la phase d'analyse devront être rappelées dans les différentes phases sous la forme de notes.
- Produire automatiquement les documentations nécessaires lors des différentes activités du processus d'élaboration du système,
- Permettre la création visuelle du système,
- Permettre la production du code.

Etat d'avancement. Actuellement l'outil couvre les phases finales du cycle de vie à savoir la conception générique et la génération de code en Java et de description matérielle VHDL.

11.1.2 La démarche qualité

11.1.2.1 La gestion documentaire

Principe général. La gestion documentaire consiste à assurer la couverture du cycle de vie des documents [Pinet, 2002] comme l'illustre la figure 11.1. Nous nous attacherons particulièrement aux parties concernant la préparation/développement et l'exploitation.

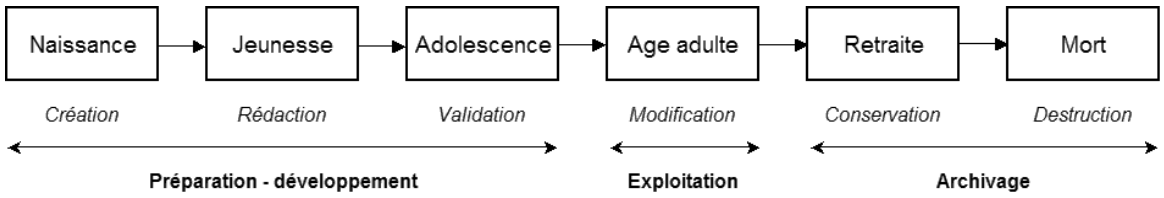


FIG. 11.1 – *Cycle de vie du document*

Pour les différentes phases l'outil va permettre la saisie des différentes fiches associées à leurs activités. Ces fiches seront consultables à tout moment et durant toutes les phases. Toute modification du contenu ne sera autorisée que par l'analyste responsable du document.

Cas du glossaire. En parallèle avec ces saisies, on pourra mettre à jour les différents *glossaires*. Pour chacun des termes, défini dans un glossaire, il sera possible de saisir des fiches détaillées explicatives afin de faire par exemple des schémas. On pourra préciser les activités pour lesquelles ces fiches se doivent d'être connues afin de programmer leur affichage automatique. Les analystes et les concepteurs étant généralement différents, il est nécessaire d'assister le passage de relais. Même si ces fiches sont incomplètes elle présenteront l'avantage de lever des interrogations au niveau du concepteur et ainsi relancer des discussions avec les analystes.

Dès qu'un mot clé est défini, toute occurrence de celui-ci sera un lien qui permettra d'accéder à sa définition et/ou ses fiches détaillées (quelle que soit l'activité).

Autres pointeurs. Pour permettre une consultation rapide et efficace des documents, des pointeurs sont mis en place dans les documents. Prenons le cas de la fiche de description des acteurs. De cette fiche, on pourra accéder automatiquement aux documents qui lui sont liés, à savoir les diagrammes de cas d'utilisation auquel participe l' acteur et le glossaire pour lequel on filtrera les termes (seuls ceux apparaissant sur la fiche initiale seront directement accessibles). Ces pointeurs sont automatiquement créés suivant leurs dépendances. Comme sous-entendu précédemment, il sera possible d'ajouter des pointeurs manuellement. Cependant, pour ne pas nuire à la lisibilité, ces pointeurs devront être commentés.

Génération automatique de documents. Certains documents sont générés automatiquement car issus directement du résultat du processus de conception. C'est le cas par exemple des diagrammes de composants. Un composant, quand il est créé, se voit préciser son interface (ses entrées/sorties) et son architecture interne est spécifiée. La généricité étant un des intérêts des composants ils sont abondamment commentés par leur concepteur et la génération automatique de fiches composants sera donc possible.

11.1.2.2 La démarche

La gestion documentaire intervient dans la démarche qualité. Elle permet d'assurer la traçabilité du processus créatif qu'est la conception d'un système. Il faudra mettre en oeuvre, dès le recueil des besoins, la définition de critères de mesures, le système de mesure (sur les activités) et un système d'appréciation des mesures (afin d'estimer la qualité du travail).

Des normes et des démarches qualités existent déjà [Pinet, 2002] :

- Normes internationales sur la gestion de la qualité et l'assurance de la qualité (ISO 9000, 9001, 9004, 10005, 10007 pour le management de la qualité et l'assurance qualité), l'audit (ISO 10011-{1,2,3}) ou l'élaboration de manuels qualité (ISO 10013).
- Normes sur l'ingénierie et la qualité du logiciel pour les systèmes d'informations documentaires (normes Z61-1xx, Z67-1xx et Z67-90x et normes NF ISO/IEC 9126 / 12119 / 12207 et ISO/IEC TR 9294)
- Normes portant sur l'évaluation et l'amélioration du processus logiciels (ISO/IEC 15504-x),
- Evaluation et amélioration des produits logiciels (ISO/IEC 12598-x)

11.1.3 Phase de conception détaillée

La partie de l'outil couvrant cette phase est en cours de finition. L'utilisation d'une architecture comme patron n'a pas encore été implémentée.

Définition du contexte. On précise le choix de l'architecture pour les différents agents (on peut éventuellement en créer une). On complète la documentation de l'agent en ajoutant les informations sur le diagramme de contexte résultant des choix technologiques.

Construction des tâches applicatives de l'agent, **construction des modules de communication et des structures d'organisation** et **construction du contrôle de l'agent**. Dans cette phase on s'intéresse à la création des composants de la coquille externe puis interne des agents. Pour chaque composant une documentation doit être créée. L'outil permet de construire des composants (voir figure 11.2) et de les entrer dans une base de données (via le placement dans un arbre qui les trie par thème).

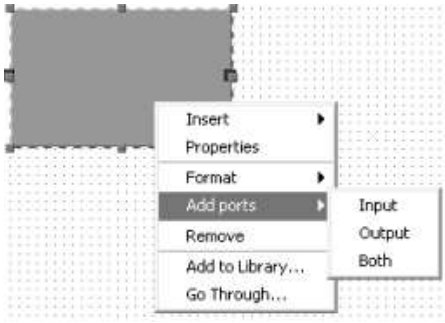


FIG. 11.2 – *Création d'un composant*

D'une manière générale, notre outil est inspiré de LabView[1] (Laboratory Virtual Instrument Engineering WorkBench) un outil de National Instrument [Cottet, 2001]. Notre outil de développement devra

1. http://www.ni.com/labview/

permettre de construire des applications multi-agents en utilisant un langage de programmation essentiellement graphique : l'objectif n'est pas d'écrire des lignes de programmes dans un langage textuel : on manipulera des objets graphiques (les composants). Ces objets graphiques représentent à la fois les variables du programme, ainsi que des fonctions qui vont réaliser des actions portant sur ces variables. La programmation consistera simplement à concevoir le traitement de l'information, organiser et relier les variables avec les fonctions au moyen de fils. Il sera évidemment possible de créer de nouveaux composants. Chaque composant, pour que la méthode soit pleinement exploitable, devra pouvoir se décliner en une version matérielle ou logicielle. Ainsi, l'outil devra fournir :

– un ensemble de composants de base (les opérateurs primitifs) qui eux-mêmes pourront être logiciels/matériels,
– un ensemble de structures de contrôle afin de réaliser des tests et autres boucles.

La gestion des communications sans fil dans un système complexe n'est pas simple. Aussi l'outil intègre l'intergiciel MWAC que nous avons créé. Pour le moment il n'existe qu'une version purement logicielle de l'intergiciel. Une version à base de brique pouvant faire l'objet d'un partitionnement est en cours de construction.
L'application est construite pas assemblage de composants (voir figure 11.3).

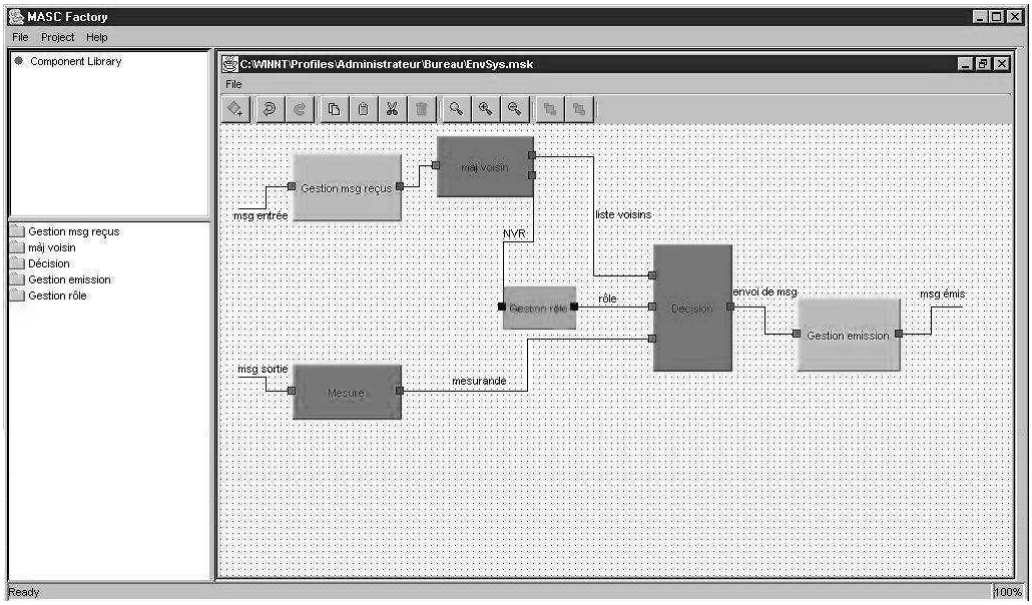


FIG. 11.3 – *Assemblage de composants*

Un composant peut contenir un arbre (voir figure 11.4), une spécification VHDL ou du code Java[2].

11.1.4 Phase d'implantation

Partitionnement. Le partitionnement est pour le moment manuel. On le configure en accédant aux propriétés du composants (voir figure 11.5).

2. L'inconvénient d'un composant contenant du code java est qu'il ne peux être décliné sous une forme matérielle

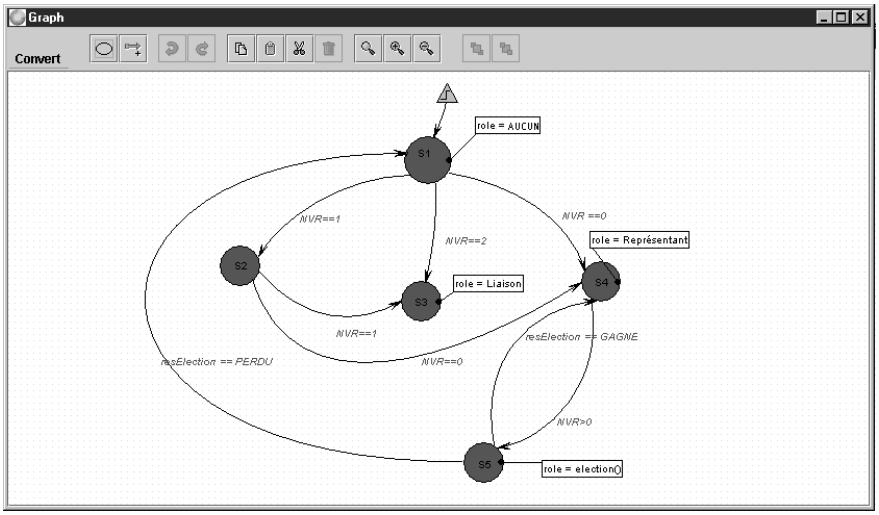


FIG. 11.4 – *Arbre*

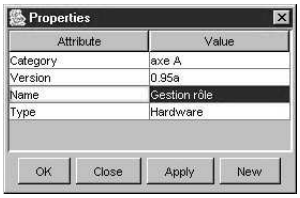


FIG. 11.5 – *Partitionnement manuel d'un composant*

Phase de co-simulation et de co-validation. Cette phase n'est pas couverte par l'outil.

Phase de réalisation. Dans cette phase, le code des parties logicielles et les spécifications matérielles sont générées. La figure 11.6 présente un extrait de code java et spécification VHDL générée avec notre outil.

La spécification matérielle est actuellement traitée par l'outil WARP de Cypress, qui prend, dans notre cas, la description matérielle VHDL comme entrée. Le flot de traitement de WARP est présenté en figure 11.7.

On peut constater que l'outil MASC permet les deux types de partitionnement possibles identifiés dans [Adams and Thomas, 1996]. On peut partitionner des composants distincts et les faire collaborer (voir figure 11.8, configuration A) ou partitionner un même composant en une partie logicielle qui collabore avec une partie matérielle (voir figure 11.8, configuration A).

A ce niveau de l'outil, un travail important reste à effectuer : la co-synthèse des interfaces. Il existe trois niveaux d'abstraction pour les interfaces logicielles/matérielles (voir figure 11.9).

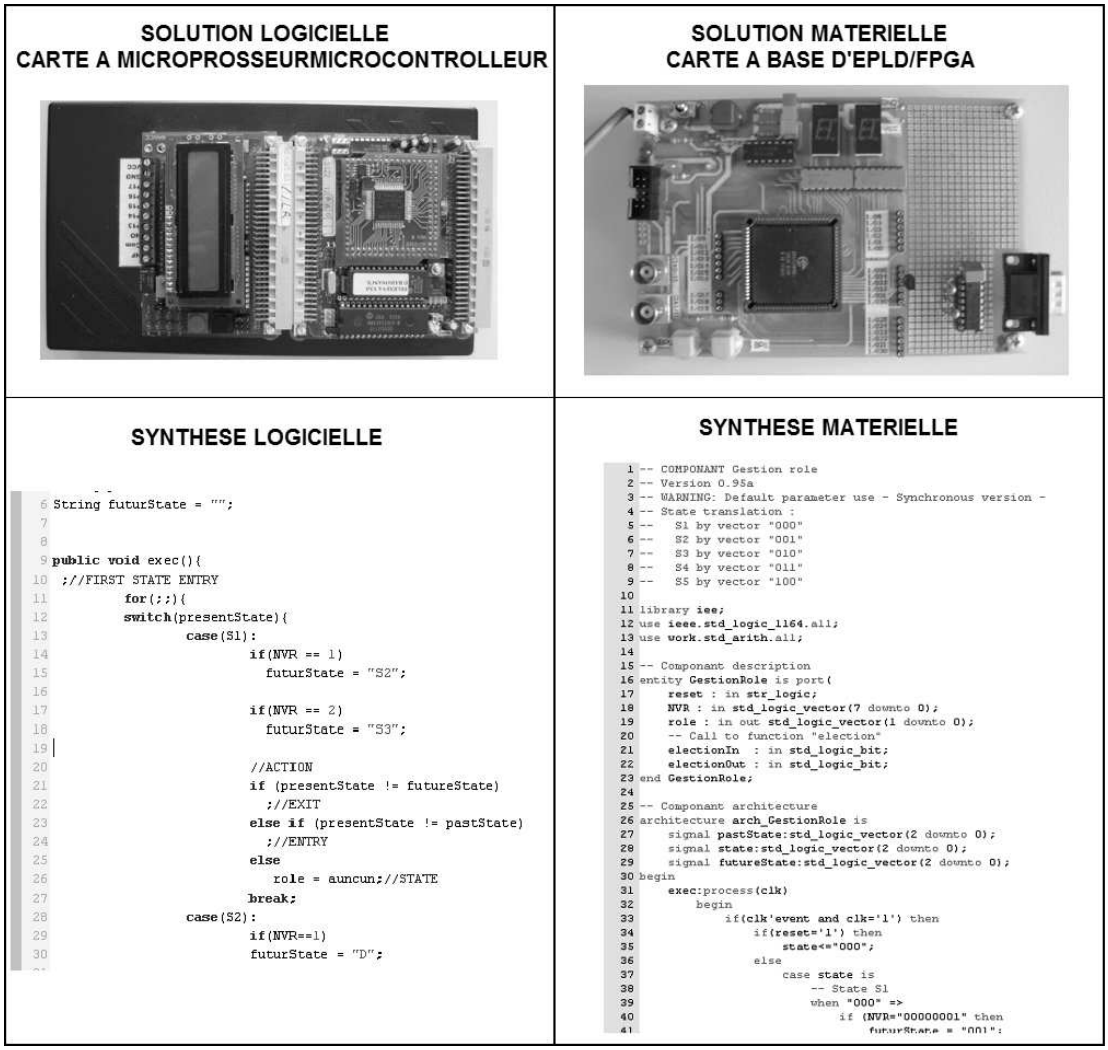


FIG. 11.6 – *Extrait de code java et spécification matérielle générés à partir d'un même composant*

11.2 Le simulateur pour les systèmes complexes ouverts sans fil

Un de nos objectifs est d'améliorer le simulateur qui nous a servi pour la simulation des systèmes complexes physiques ouverts. Cet outil est générique : il pourrait tout aussi bien simuler des applications de la robotique collective.

Comme nous l'avons vu l'un des intérêt du passage à la simulation est de vérifier simplement et efficacement le fonctionnement d'une solution. Ainsi, on peut corriger d'éventuels problèmes avant de passer sur les systèmes physiques.

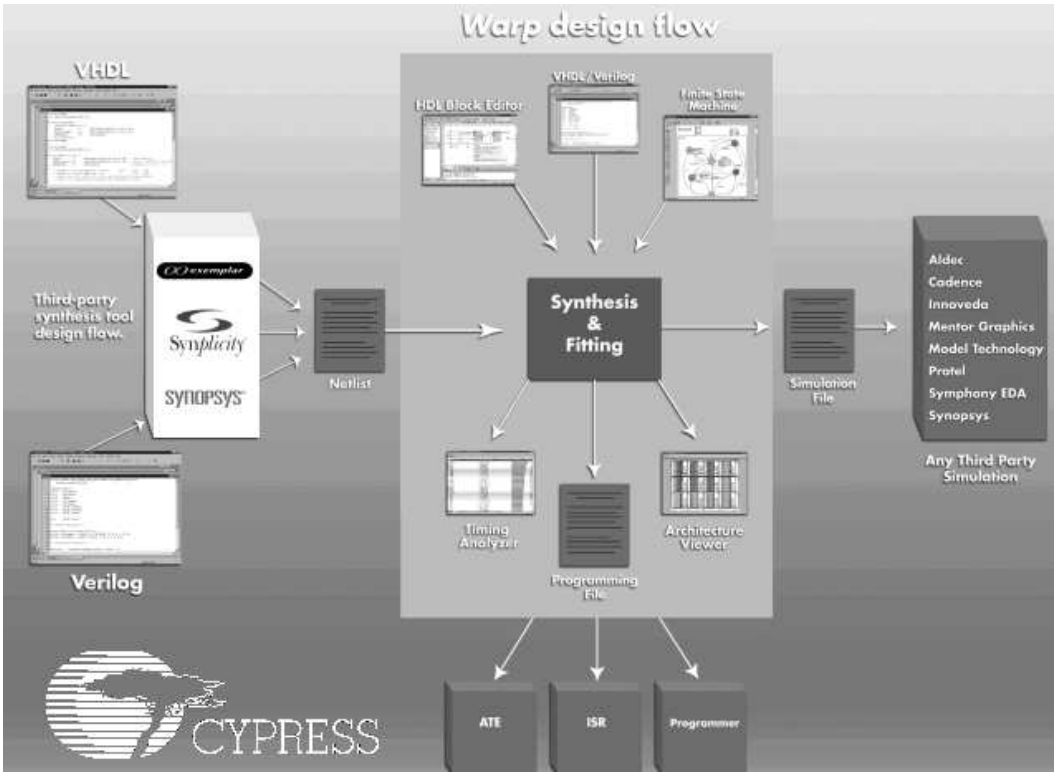


FIG. 11.7 – *Flot de traitement de l'outil WARP*

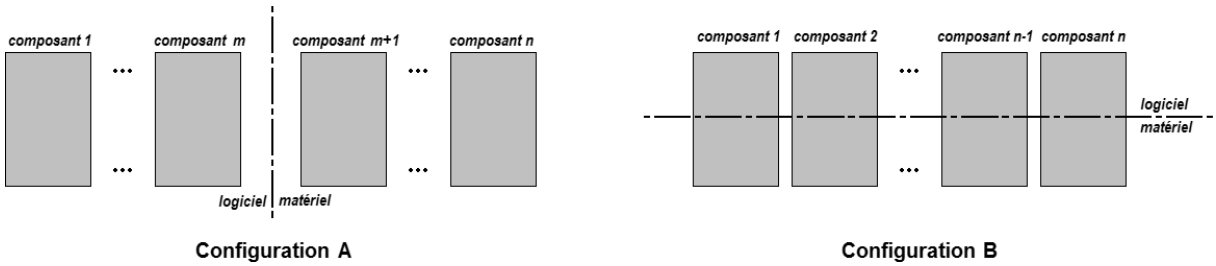


FIG. 11.8 – *Les deux types de co-design*

Un autre avantage que l'on pourrait envisager serait la limitation du coût financier de test de systèmes physiques. En effet, un des inconvénients de ces tests est lié au passage à l'échelle. Plonger dans un même environnement un grand nombre d'entités physiques est coûteux. Une solution à ce problème peut être un simulateur hybride logiciel/matériel.

Ainsi si l'on reprend le cas de la simulation pour EnvSys, on peut imaginer faire collaborer, au sein d'une même simulation, quelques capteurs physiques et une majorité de capteurs simulés. Pour cela, il suffirait de les interfacer avec le simulateur déjà existant (voir figure 11.10).

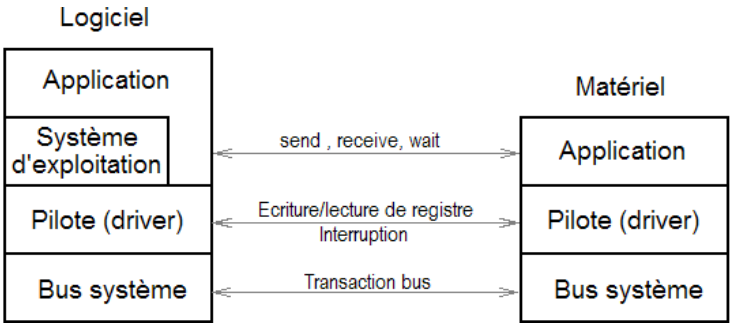


FIG. 11.9 – *Abstraction des interfaces logicielles/matérielles*

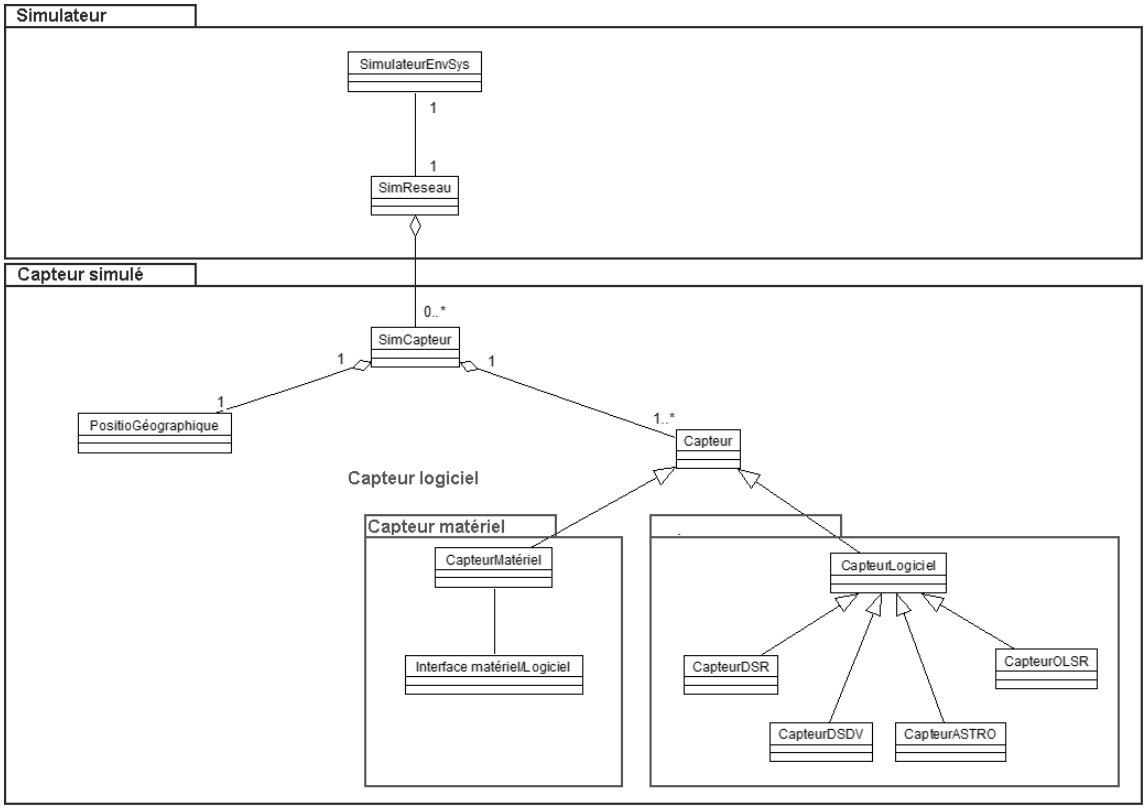


FIG. 11.10 – *Architecture du simulateur logiciel/matériel pressentie*

Chapitre 12

Application à la gestion de bases de données réparties : gestion d'une compétition sportive de danse

Dans ce chapitre, nous présentons une application du domaine des systèmes d'information embarqués que nous allons traiter avec la méthode DIAMOND. Nous commençons, dans une première section, par introduire la problématique de la gestion des notes dans une compétition de danse sportive. Dans un deuxième temps, nous abordons l'analyse du problème selon DIAMOND.

12.1 Les compétitions sportive de danse

Généralités. La danse sportive est un sport de compétition qui se pratique en couple. Cette activité se décline en deux styles totalement différents: les danses appelées standards [1] et les danses dites latines.

Chacun des styles comporte cinq danses interprétées par les couples sur des chorégraphies basées sur une technique pointue propre à chaque danse. Comme dans toute discipline sportive, il existe différentes classes d'âges allant des Poussins (moins de 9 ans) jusqu'aux vétérans (plus de 45 ans). Chaque catégorie est divisée en séries, partant de la série 5 (débutants) pour finir en première série (niveau international).

Lors d'une compétition les couples appartenant à une même catégorie dansent simultanément, sauf s'ils sont trop nombreux, et dans ce cas là, plusieurs passages sont nécessaires. Les juges parcourrant la piste de danse doivent alors sélectionner les couples qui iront en finale ou, si c'est une finale, classer les couples s'exécutant sur la piste. Toute compétition aboutit à un classement qui reflète le résultat de l'appréciation technique et artistique d'un jury pour un couple. Ce classement se fait (souvent) de manière manuscrite.

L'objectif de ce projet est d'informatiser le système de jugement des couples dans une telle compétition. Le règlement d'une compétition est soumis à un règlement qu'il est possible de consulter dans [Comité National De Danse Sportive, 1999]).

1. On les nomme ainsi car elles furent les premières à faire l'objet d'une standardisation au début du XX ème siècle

Jugement. Lors d'une compétition, ce sont les couples (et non les individus) qui sont identifiés via un numéro de dossard. Le classement des couples se fait par un jury composé de trois à onze juges. Ils doivent sélectionner (éliminatoires) ou classer (finale) les danseurs présents sur la piste. Pour chaque danse, le juge doit remplir une fiche contenant son jugement.

Les deux composantes du jugement. Comme il a été sous-entendu précédemment, il existe deux sortes de jugements :

- Les éliminatoires ou les juges sélectionnent les couples, dans une catégorie et un style précis, les autorisant à poursuivre la compétition. Pour cela, ils cochent les numéros de dossard qu'ils souhaitent voir accéder à l'étape suivante.
- La finale où chaque couple est classé (généralement il y a 6 couples sélectionnés pour une finale) du premier au dernier. Le chiffre qu'il attribue à un couple (1 pour le premier) correspond a sa place.

Les deux grilles de notations minimales sont représentées en figure 12.1.

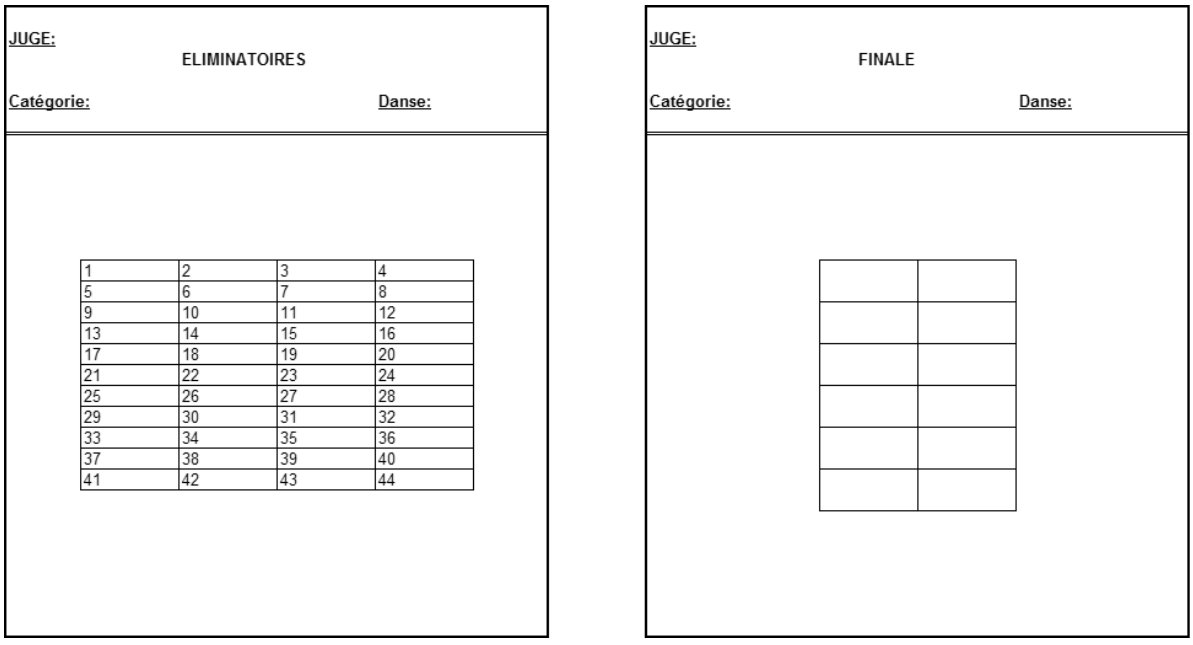


FIG. 12.1 – *Différentes grilles de notations (Eliminatoires / Finale)*

Déroulement d'une compétition. Un jury est composé de simples juges et d'un président qui à pour rôle de gérer l'après jugement. Avant chaque danse et après chaque phase (éliminatoire ou finale), un ramasseur récupère les grilles de notation qu'il avait distribué auparavant. Un annonceur introduit les étapes de la compétition, le nombre de couples à sélectionner (ou classer), le nombre de passages. Un logiciel[2] permet d'établir le classement.

2. Le logiciel Skating System de l'IDSF (International Dance Sport Federation)

12.2 Un nouveau système de gestion des notes

12.2.1 Intérêt de l'automatisation de la gestion des résultats

Il existe plusieurs raisons qui ont poussé l'ISDF (International Dance Sport Federation) à s'intéresser à l'informatique. Nous présentons, ci-dessous, les critères mettant en évidence l'intérêt d'une automatisation du jugement d'une compétition. Ils sont issus d'observations faites par des danseurs et membres de jury.

Rapidité. Le gain en temps durant la collecte des résultats et l'élaboration des classements est un point important. Les danseurs apprécieront que les danses s'enchaînent plus rapidement et les spectateurs apprécierons cette densité plus importante de spectacle.

Convivialité. Les juges mettent en avant l'austérité du procédé actuel. Actuellement, la présentation des informations est faite de manière à faciliter le traitement automatique des notes mais en aucun cas pour simplifier le travail des juges.

Sécurité. Une collecte automatique des informations augmentera la confiance des danseurs dans les résultats (aucune feuille ne circule entre les divers intervenants, aucune feuille ne sera plus égarée).

Synthétique. Le juge est souvent surchargé d'informations (verbales et écrites). Les grilles d'évaluations doivent devenir dynamique, s'adapter aux éliminations. De plus il est nécessaire de pouvoir adresser à un juge (ou diffuser) des messages.

Ouverture. Il est important que le système soit ouvert car l'utilisation du système permettra d'entrevoir de nouvelles fonctionnalités (accès à l'historique d'un couple, statistiques en temps réel etc.), de nouvelles exigences.

12.2.2 Critique d'un système actuel de gestion des notes.

Le système existant. Il existe déjà un système qui utilise des PDA mais qui présente de nombreux inconvénients. Son coût d'utilisation est très élevé (il faut payer pour chaque compétition la location des PDA et du système de recueil des notes). Les PDA n'ont aucun module sans fil : ce ne sont que des feuilles électroniques qu'il faut amener à une station qui collecte des notes. La convivialité de ce produit est très critiquée (ce système est un produit générique qui n'a pas été spécialement conçu pour la danse).

Après chaque étape de la compétition, les PDA sont collectés puis connectés à un ordinateur qui recueille les notes. Ensuite, ils doivent être reparamétrés pour la suite de la compétition puis sont redistribués aux juges. La lenteur de ce procédé fait que les organisateurs de compétition louent généralement deux jeux de PDA afin de paralléliser les tâches.

L'évolution que nous allons apporter. Pour notre part, nous équiperons les PDA d'un module de communication sans fil. Cette solution technologique permettra d'avoir un système totalement dynamique et de concevoir un système qui permettra de pallier ce problème. L'inconvénient de l'utilisation d'un tel module est la nécessité de sécuriser les échanges de communication.

12.3 Traitement du problème avec DIAMOND

Nous n'allons présenter ici que le début du traitement du problème avec DIAMOND. En effet, à cette date le projet est toujours dans la phase de recueil des besoins. Nous travaillons essentiellement avec un danseur actif qui a mis en évidence le besoin d'un système.

12.3.1 Approche préliminaire du problème

Inscription. Dans le système que nous proposons, les danseurs peuvent s'inscrire à la compétition via internet (il en est de même pour les juges qui acceptent d'arbitrer les compétitions). Pour ceux qui n'ont pas la possibilité d'accéder à internet, un poste est disponible sur le lieu de la compétition. Il en est de même pour les compétitions disposant de moyens moins importants: le poste situé sur le site de la compétition suffit. Cette inscription alimentera une base de données: les données traditionnellement demandées en début de compétition seront saisies à ce niveau.

Un compétiteur renseigne son nom, son prénom, sa date de naissance, son adresse, son club de provenance. Cependant, ce ne sont pas des individus qui s'inscrivent mais bien des couples.

Les juges qui acceptent vont saisir leur nom, leur prénom, leur adresse et surtout un mot de passe qui leur permettra de s'identifier sur internet (pour consulter la liste des inscrits) ou s'authentifier sur les PDA qu'ils utiliseront pour le jugement.

La préparation des étapes. Le contrôleur paramètre la compétition comme le type des danses. Il organise les étapes (finale, demi-finale, etc.) en préparant les passages. Un passage correspond à l'audition d'un groupe de danseurs par un groupe d'arbitres. Le nombre de passages (et le nombre de couples par passage n'est pas calculable: c'est un consensus entre les différents juges selon leurs appréciations de critères ad-hoc (taille de la piste, éventuels retards, type de compétition etc.)

La notation. Actuellement la notation est faite de manière centralisée car le contrôleur travaillait jusque là sur un poste de travail de type PC tout comme les arbitres travaillaient avec des fiches. Le poste de travail pourrait être remplacé par un simple PDA. Le contrôleur est celui qui utilise le plus d'informations (visualisation des résultats intermédiaires, préparation des prochains passages, édition de messages à destination des juges etc.). Il n'est donc pas judicieux de centraliser le processus de notation sur un seul PDA.

Assister l'attaché de presse. Le travail de l'attaché de presse peut être simplifié car une grande partie des informations du document de synthèse "type" peut être créée automatiquement.

12.3.2 Etude des acteurs

Le *couple de danseur* s'inscrit pour pouvoir participer à la compétition.

Les *juge*s acceptent de participer à la compétition en s'inscrivant. Ils notent ou classent les différents couples qui se présentent aux différents passages. Dans un premier temps, ils sélectionnent les couples qui pourront passer à l'étape suivante. En finale, chaque juge propose un classement des compétiteurs qui sont encore en course (une finale oppose entre 2 et 7 couples).

Le *présentateur* présente les couples, les phases de la compétition. Pour cela il se base sur les informations affichées sur son PDA :

– Etat d'avancement de la compétition (Quelle est l'étape en cours? Quelle est l'étape suivante? Combien de passages y a t'il? Qui sont les concurrents?)
– Des messages à diffuser et qui sont donnés par l'accueil.

Le *contrôleur* est la personne qui va gérer les différentes étapes de la compétition. Les paramètres qu'il saisit sont le nombre et le nom des danses, le nombre de couples voulus en finale, nombre de passages. Il valide les numéros de dossard associés aux couples etc.

L'*attaché de presse* qui à l'aide des informations qu'il visualise sur la compétition (résultats provisoires, notations etc.) rédige un compte-rendu détaillé de la compétition.

12.3.3 Etude des cas d'utilisation

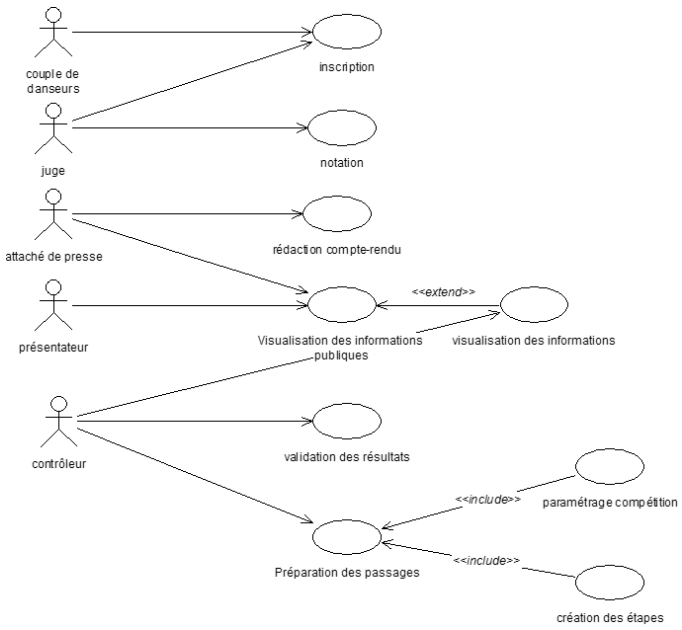


FIG. 12.2 – *Cas d'utilisation du système de gestion de danse*

Les différents cas d'utilisation sont :

– L'*inscription* qui permet aux danseurs et aux juges de s'inscrire,
– La *notation* qui permet aux juges de noter ou classer les couples,
– La *validation des résultats* qui permet au contrôleur d'appliquer les éventuelles pénalités votées par les juges et d'autoriser la diffusion des résultats finaux,
– La *préparation des passages* durant laquelle le contrôleur définit pour une phase le nombre de passages et la liste des couples associés aux passages,
– La *visualisation des informations* qui permet au contrôleur de visualiser toutes les données dont les grilles que sont en train de saisir les arbitres.

– La *visualisation des publiques* qui permet de visualiser les informations relatives à la compétition (hors résultats de l'étape en cours).
– La *rédaction du compte-rendu* permet d'assister le travail de l'attaché de presse en produisant des documents type de synthèse (tableaux et graphiques relatifs aux notations et constitution des passages et des couples)

12.3.4 Etude des besoins en services

Les diagrammes suivants (figure 12.3) présentent les besoins en services des différents acteurs du système.

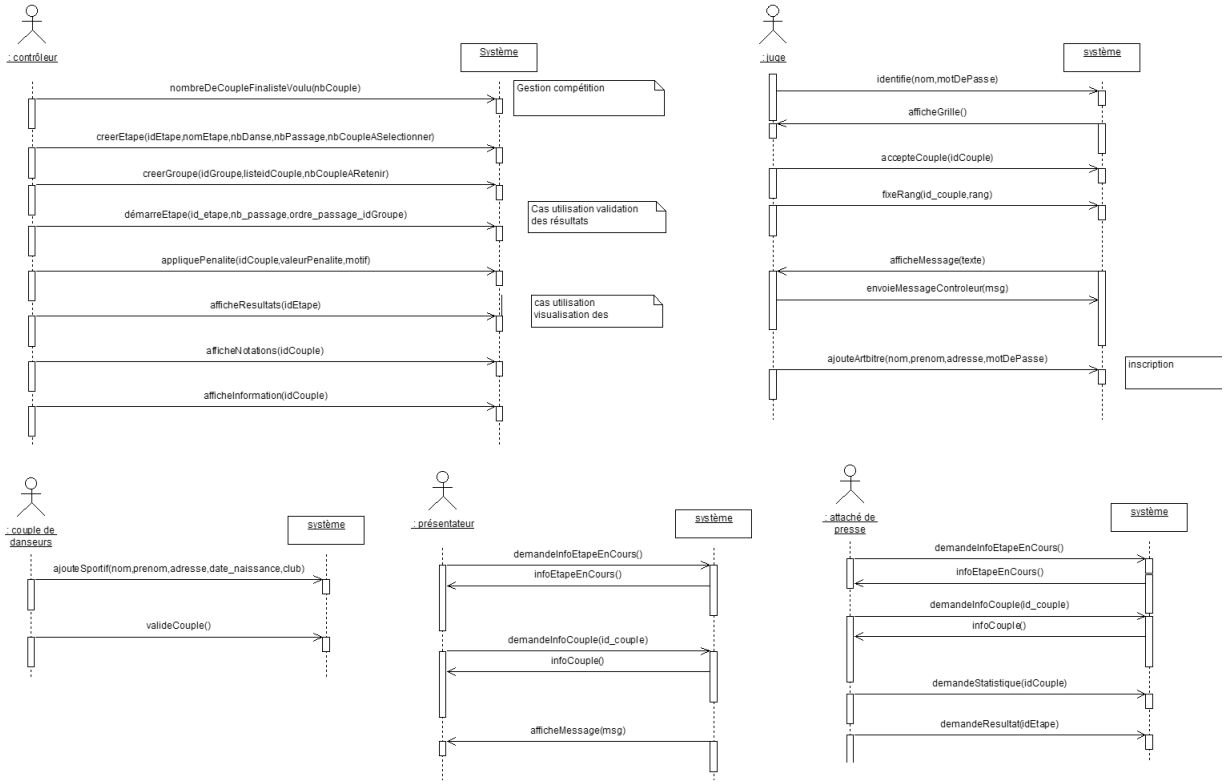


FIG. 12.3 – *Besoins en services du système de gestion de danse*

12.3.5 Etablir les modes de fonctionnement

Cette activité est intéressante lors de l'étude de systèmes physiques plus critiques tandis que ce système est essentiellement logiciel sans contrainte matérielle spécifique.

12.3.6 Eléments pour l'analyse multi-agents du problème

Les agents. Nous nous orientons vers une agentification du type "agent personnel". C'est à dire que chaque intervenant humain sera assisté par un agent.

Les agents *juge* seront chargés de mettre à la disposition des juges les grilles de notations et les informations sur l'étape en cours. Ils automatiseront tous les transferts afin qu'après chaque évaluation, le juge soit informé de l'avancement de la compétition (temps dont il dispose avant sa prochaine sollicitation etc.). Ces agents procéderons à la fusion des notes et à l'établissement des classements.

Les agents *attaché de presse* procéderont à de l'écoute flottante afin de prélever les informations "publiques" qui circulent sur le réseau afin qu'il puisse assister l'attaché de presse pour établir son rapport.

L'agent *contrôleur* aura pour objectif d'assister le contrôleur dans sa gestion du planning. De plus, il ordonnancera les étapes de la compétition et informera les autres agents d'éventuels changements sur le programme initial. Il veillera au bon respect des contraintes temporelles et enverra automatiquement des messages d'avertissement aux autres agents. Cet agent peut recevoir des demandes d'informations des autres mais ne répond que si ces dernières peuvent être "publiées".

L'agent *présentateur* a une liaison particulière avec l'agent contrôleur car il nécessite d'avoir accès au planning afin de pouvoir procéder aux différentes annonces (retard, délibération etc.).

Un agent *sécurité* sera mis en place au niveau du poste d'inscription afin de veiller à ce que les autres agents du système soient autorisés à participer au processus d'évaluation (vérification des mots de passe, des adresses MAC des machines connectées etc.).

L'organisation. Le modèle MWAC sera utilisé afin que la couverture du réseau soit plus grande (multiplicité des points d'accès). De plus, lors des délibérations, l'agent qui aura le rôle de représentant sera celui qui sera chargé d'établir la synthèse des notations et de donner à l'agent contrôleur le résultat de l'étape. Ce résultat sera d'ailleurs mémorisé au niveau de l'agent.

12.3.7 Eléments pour l'implémentation

Dans cette application, le problème est de nature très logicielle. On utilise une plate-forme matérielle figée (le PDA). Ce PDA disposera d'un module Wifi ou Bluetooth. L'agent sera donc constitué du logiciel créé et du PDA. Le partitionnement sera tout logiciel. Le code généré sera en Java.

Conclusion

Dans une première partie nous avons présenté les perspectives en terme d'outils. Un effort important sera à faire dans le cadre de l'outil accompagnant la méthode. Dans l'état actuel, seul la conception détaillée et la génération de code sont dans un état avancé.

Un second effort sera à porter sur l'outil de simulation afin de permettre de mettre en jeu des entités logicielles simulées et des entités matérielles.

Dans une seconde partie, nous avons présenté nos perspectives en terme d'application. Un système permettant d'assister les juges dans leur travail de notation d'une compétition de danse sportive est à concevoir, montrant la pertinence de notre approche sur des applications embarquées sur des architectures mobiles banalisées.

CONCLUSION GENERALE

CONTEXTE DE TRAVAIL

A l'origine de cette thèse est un intérêt pour les problèmes relevant des systèmes complexes physiques ouverts et des systèmes multi-agents. Parmi ces motivations se trouvent des travaux dans le contexte de la robotique collective et de l'instrumentation ou des domaines en plein essor comme l'informatique diffuse ou les systèmes d'informations embarqués coopératifs.

De la confrontation de ces deux domaines émergent deux importantes problématiques. La première concerne la méthode qu'il faut adopter pour s'intéresser à ces problèmes complexes physiques ouverts. Les systèmes multi-agents, qui sont une approche pertinente, ne fournissent pas de méthodes complètes pour traiter ces systèmes dont la nature est physique: elles se prêtent essentiellement à traiter la nature logicielle du système. La seconde problématique concerne des besoins en architectures spécifiques. Considérer les composants des systèmes complexes physiques comme des noeuds coopérants d'un réseau sans fil est une démarche attrayante qui peut-être vue comme la traduction physique extrême et la décentralisation. Il nous est alors nécessaire de proposer un modèle afin d'assurer une communication adaptative à l'environnement, à la portée d'émission des modules de communication des agents et leur déplacement. Ce problème qui traite de la localisation de destinataire est lui même un problème complexe.

CONTRIBUTIONS EN TERMES DE MODÈLES

Une importante contribution de cette thèse est associée à la première problématique soulevée. Elle réside dans la méthode complète d'analyse et de conception DIAMOND (Decentralized Iterative Approach for Multiagent Open Networks Design). Cette méthode possède plusieurs spécificités que nous allons résumer.

En terme de cycle de vie, notre méthode présente déjà plusieurs originalités.
Tout d'abord, dans les méthodes multi-agents, la prise en compte de la partie matérielle du système se limite au choix de la plate-forme sur laquelle seront déployés les agents. Ces méthodes discriminent les parties logicielles et matérielles d'une manière précoce dans le cycle de vie. Notre méthode, qui se veut de type co-design, permet d'unifier en un seul et même cycle de vie, la conception des *parties*[3] logicielles et matérielles.
Ensuite, toujours au niveau des cycles de vie, le choix d'un cycle en spirale est original. Nous utilisons

3. On ne peut pas parler réellement de parties car avant le processus de partitionnement rien n'est logiciel et rien n'est matériel (hormis les capteurs et effecteurs)

ce cycle afin de renforcer les liens entre les processus, de permettre des itérations aux niveaux des phases et parce qu'il se prête fortement à l'intégration de démarches qualité.

Au niveau de la phase de recueil des besoins, deux spécificités sont à noter.
La première concerne le diagramme de cas d'utilisation qui, de part la nature physique du système, met en jeu des acteurs physiques qui peuvent ne pas nécessiter de services de la part du système que l'on conçoit: en génie logiciel avoir un acteur qui ne participe à aucun diagramme de séquences mettrait en exergue une erreur d'analyse.
La seconde spécificité réside dans la structuration du fonctionnement global du système à concevoir via les modes de marche et d'arrêt.

Au niveau de la phase d'analyse d'autres originalités sont présentes.
Tout d'abord, le travail itératif (Phase de situation $\rightarrow$ Phase individuelle $\rightarrow$ Phase sociale $\rightarrow$ Phase d'intégration $\hookleftarrow$) est une démarche structurée adaptée à l'analyse des systèmes physiques organisés. Il permet de traiter de manière progressive l'organisation en intégrant les diverses interactions possibles entre les différents modèles comportementaux individualistes et collectifs.
L'utilisation de diagramme de contexte pour décrire les différentes interfaces matérielles de l'agent et mettre en évidence les besoins en senseurs et effecteurs est là aussi originale.

La phase de conception générique commence par compléter le diagramme de séquence. A chaque interface matérielle doit correspondre le choix d'une solution technologique pour les différents capteurs et effecteurs. Nous fixons un ordre temporel pour la construction des agents (dans lequel la dimension collective du système a été intégrée): nous commençons par construire la partie externe de l'agent et finissons par son coeur.
La construction des agents se fait à l'aide de composants graphiques agencés dans un pattern provenant d'une architecture d'agent. Cela permet de guider le concepteur dans sa construction. Les composants sont eux-mêmes construits à l'aide d'autres composants ou avec des formalismes de spécification i.e. machines à états finis.

La phase d'implémentation est originale dans le sens ou elle propose des critères de partitionnement de chacune des entités du système. De ce partitionnement découle une génération automatique du code (pour les parties logicielles) ou des spécifications matérielles de type VHDL.

La seconde problématique concerne l'architecture spécifique pour une gestion adaptative des communications dans les systèmes complexes physiques dont les constituants sont vus comme des noeuds d'un réseau sans fil. Le routage est un compromis entre des contraintes locales des noeuds telles que le niveau d'énergie et des contraintes globales (la nécessité de trouver des routes).

La deuxième contribution importante de cette thèse réside donc dans une résolution de ce problème. Le modèle MWAC (Multi-Wireless-Agent Communication) que nous proposons donne aux agents les moyens de s'organiser eux-mêmes en fonction de leurs contraintes. Les groupes qu'ils forment via des mécanismes d'auto-organisation basés sur l'attribution de rôle permettent de réduire significativement la perte énergétique induite par l'inondation. La dépense énergétique occasionnée par les messages que s'échangent les agents pour s'auto-organiser nuit peu à l'efficacité de MWAC pour nos applications. Ce modèle a donné naissance à un intergiciel qui se décline, suivant les problèmes, en un package, un composant ou un archétype d'agent.

CONTRIBUTIONS EN TERME D'APPLICATION

Une application importante de cette thèse, est l'utilisation de la méthode DIAMOND et du modèle MWAC pour construire le système au sein du projet EnvSys. L'objectif de cette application est l'instrumentation d'un réseau hydrographique souterrain. Les capteurs, qui sont déployés, sont modélisés par des agents suivant les spécifications du modèle MWAC. Ces agents ont été embarqués sur deux types de cartes à microcontrôleurs et sont opérationnels. L'implémentation réelle dans la grotte n'a pas encore été possible car il reste un important travail à faire au niveau de la propagation des ondes dans les milieux souterrains [Schira, 2003].
Cette thèse a donné lieu à la création de plusieurs outils.
Une méthode d'analyse et de conception, qui se veut complète, nécessite des concepts, une démarche et des outils. Aussi l'outil MASC (MuliAgent System Codesign) a été construit dans cette optique. Dans un premier temps, nous avons porté nos efforts sur la validation de la partie conception générique/implémentation. Ainsi, cet outil permet de construire des composants et de les agencer de manière à créer des agents. De cette conception, peut être généré du code (langage Java ou C) ou en un langage de description matérielle (VHDL).
La création du modèle MWAC (et son utilisation pour EnvSys) a nécessité l'utilisation d'un simulateur. Ce simulateur recrée l'environnement physique des capteurs et permet d'évaluer différents critères dont essentiellement les volumes de configuration et les volumes utiles transmis. Ces mesures permettent d'évaluer le rendement. La définition du rendement des réseaux traditionnels dit point à point n'étant pas applicable à notre problème, nous avons défini un critère de rendement mettant en avant la dépense énergétique.

PERSPECTIVES

Notre travail a donné d'ores et déjà lieu à des extensions et des développements qui ouvrent des perspectives riches tant au niveau méthodologique qu'applicatif.

En terme de perspectives méthodologiques, la méthode ne sera réellement complète que lorsque l'outil couvrira toutes les phases de la méthode. A court terme, l'outil doit couvrir les phases de recueil des besoins et d'analyse. De plus, la méthode doit être confrontée à d'autres problèmes afin que les enseignements que nous en tirerions permette son amélioration. C'est dans cet objectif, que nous travaillons sur le projet qui consiste à assister des arbitres dans la notation qu'ils effectuent dans le cadre de compétition de danse sportive. Cette application plutôt logicielle (bien qu'embarquée sur PDA) va nous permettre d'évaluer notre méthode pour la conception de système sur "smart device" de ce type.

Suite à cela, une importante réflexion doit être amorcée afin de réfléchir à l'intégration à l'outil d'une démarche de gestion de projet et de gestion de la qualité. La démarche devra permettre plus que la traçabilité complète de l'avancement du projet et la mesure de la satisfaction du demandeur. Nous pensons que la mise en place de mesures qualité associées à des critères) peut permettre d'aider à l'agentification du problème, grâce par exemple à des méthodes de méta-learning.

Une autre perspective en terme d'outil réside dans le simulateur que nous souhaitons rendre hybride afin de pouvoir simuler dans un même scénario des entités physiques et des entités purement logicielle (simulant les entités physiques). Les intérêts principaux que nous voyons sont:

– les systèmes multi-agents mettant en jeu de nombreuses entités, elle permettra de diminuer significativement le coût des simulations sans perdre de vue le fonctionnement réel,

– une amélioration du test du système produit en instanciant des fautes complexes aux niveaux des entités simulées et de voir l'impact au niveau des entités matérielles,
– une extension possible de la démarche qualité en incluant un feedback dans la méthode.

Ce travail avait pour l'objectif de montrer que l'application des systèmes multi-agents à des systèmes physiques pouvaient être très profitables et que le niveau de maturité atteint dans les développements de systèmes multi-agents permettait d'envisager d'aborder les problèmes de ces nouvelles architectures mobiles en environnements ouverts d'ordinaire très déterministe.

Le potentiel des systèmes multi-agents dans ce champ d'application peu étudié est très grand et que la voie est ouverte aujourd'hui pour des réalisations mêlant logiciel et matériel mobile dans les applications de l'informatique embarquées et des systèmes diffus.

sixième partie

ANNEXES

Annexe A

Notations de la méthode DIAMOND

Cette annexe présente les principales notations de la méthode DIAMOND. Quand une notation est inspirée d'un existant, on précisera le nom de cette notation ou méthode ainsi que les modifications que nous y avons apportées (partie 'notre utilisation'). Si cette rubrique n'apparaît pas, c'est que nous utilisons cette notation comme elle a été prévue.

A.1 Notations utilisées dans le recueil des besoins

Le recueil des besoins peut faire intervenir différentes notations informelles permettant durant l'*approche préliminaire du problème (A.1)* . L'*étude des acteurs (A.2)* et l'*étude des cas d'utilisation (A.3)* permettront l'établissement des ***diagrammes de cas d'utilisation***. L'*étude des besoins en services (A.4)* utilise les ***diagrammes de séquence*** et les ***diagrammes de collaboration***. L'*études des modes de marche et d'arrêt (A.5)* met en oeuvre des descriptions textuelles.

A.1.1 Les diagrammes de cas d'utilisation

Source: UML

Description: Les diagrammes de cas d'utilisation représentent la structure des fonctionnalités nécessaires aux différents utilisateurs du système. Ils mettent en jeu les *acteurs* du système (symbolisé par un "homme" pour les acteurs humains, un rectangle pour les autres non-humains), des *cas d'utilisation* (symbolisés par des "bulles") et les diverses relations entre les acteurs/cas et cas/cas.

Les relations possibles entre cas d'utilisation sont:

– l'*inclusion*, représentée par une flèche continue pleine, si un cas d'utilisation incorpore explicitement un autre (le cas d'utilisation inclus n'est jamais exécuté seul mais seulement en tant que partie d'un cas de base plus vaste),

– l'*extension*, représentée par une flèche pointillée , si un cas incorpore implicitement un autre (le cas peut fonctionner seul mais peut éventuellement être complété par un autre mais uniquement sous certaines conditions et à certains points particuliers de son flôt d'événement),

– la *généralisation*, représentée par une flèche continue non pleine, si un cas peut être généralisé ou spécialisé (le cas d'utilisation descendants héritent de la sémantique de leurs parents, mais peuvent éventuellement les modifier).

La relation possible entre un acteur et un cas est la participation (trait continu).

La seule relation possible entre acteurs est la généralisation (flèche continue non pleine).

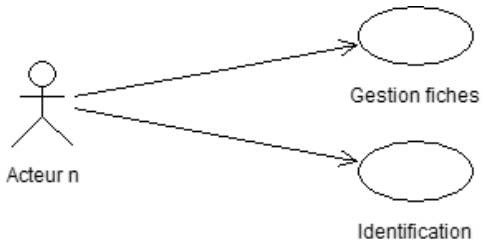


FIG. A.1 – *Diagramme de cas d'utilisation*

A.1.2 Les diagrammes de séquences

Source: UML

Description: Le diagramme de séquences modélise les interactions. Il s'utilise essentiellement pour décrire les scénarios d'un cas d'utilisation (les entités sont les acteurs et le système) ou décrire des échanges de messages entre objets.

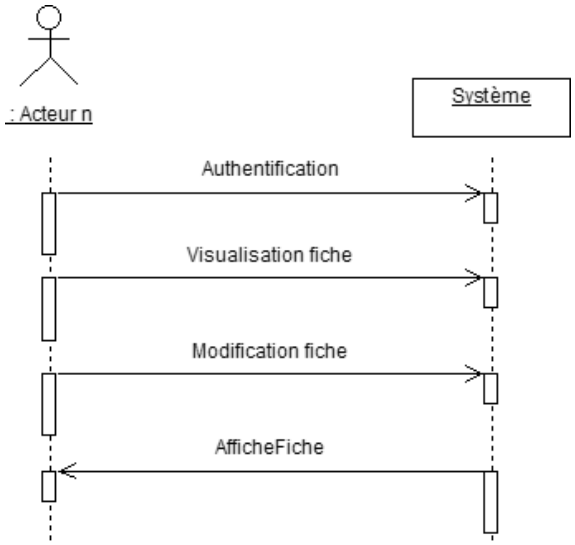


FIG. A.2 – *Diagramme de séquences*

Notre utilisation: A ce stade du cycle de vie du système, nous utilisons les diagrammes de séquences uniquement pour modéliser ce que les acteurs attendent du système.
Il est possible qu'un acteur figurant sur le diagramme de cas d'utilisation n'apparaisse pas sur le diagramme de séquence, ce qui traduit généralement une erreur d'analyse en utilisant (RUP,UML). Dans un système physique, il se peut qu'une entité physique passive intervienne dans un cas d'utilisation mais n'apparaisse pas dans le diagramme de séquence. La nature de l'interaction est en fait différente.

A.1.3 Les diagrammes de collaboration

Source: UML

Description: Le diagramme de collaboration est une mise en contexte des interactions des entités du système: on peut y préciser les états des entités qui interagissent.

Sur ce diagramme, on peut indiquer pour chaque message les clauses qui conditionnent son envoi, son rang (son numéro d'ordre par rapport aux autres messages), sa récurrence et ses arguments.

La syntaxe d'un message est la suivante est **[pré "/"] [["["cond"]"] [séq] ["*"["||"]["["iter"]"]] ":"] [r ":="] msg"("[par]")"**

- *pré* sont les prédécesseurs (liste de numéros de séquence de messages séparés par une virgule). Un message n'est envoyé que lorsque tous ses prédécesseurs le sont aussi (permet de synchroniser l'envoi de messages).
- *cond* est une garde, une expression booléenne, qui permet de conditionner l'envoi du message, à l'aide d'une clause exprimée en langage naturel.
- *séq* est un numéro de séquence du message qui indique le rang du message, c'est-à-dire son numéro d'ordre par rapport aux autres messages.
- *iter* qui indique une récurrence du message et permet de spécifier en langage naturel l'envoi séquentiel (ou en parallèle, avec "||") de messages.
- *r* qui est la valeur de retour du message,
- *msg* le nom du message.
- *par* les paramètres (optionnels) du message.

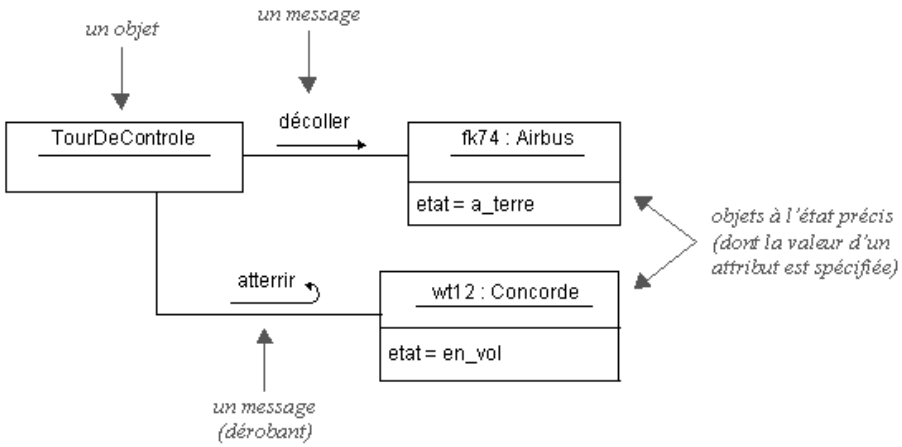


FIG. A.3 – *Diagramme de collaboration*

Notre utilisation: Nous n'utiliserons les diagrammes de collaboration que pour insister sur les interactions attendues par l'utilisateur entre des acteurs physiques (des équipements) d'un système. Il ne sera donc utilisé que si le protocole d'interaction demandé par l'utilisateur, ou les solutions technologiques imposées par le demandeur est précis et contraignant.

A.2 Notations utilisées dans l'étape d'analyse

L'étape d'analyse multi-agents peut, tout comme dans l'étape précédente, faire intervenir différentes notations informelles durant la *phase individuelle (B.1)* (synoptique décrivant l'agent physique, esquisses pour représenter l'environnement de l'agent etc.). Cette phase fait aussi intervenir le ***diagramme de contexte*** pour représenter l'agent comme entité physique située dans un environnement et les ***graphes d'actions/perceptions faisables***. Le comportement individuel de l'agent peut être exprimé de manière textuelle ou plus formelle à l'aide de FSM ou de diagrammes de collaboration. Les protocoles d'interaction sont spécifiés en utilisant les diagrammes de séquences UML ou AUML. La phase sociale est très textuelle mais utilise, pour décrire l'organisation, des ***diagrammes entités/relations*** ou des ***diagrammes de classe***.

A.2.1 Diagramme de contexte

Source: SART

Description: Le diagramme de contexte des données (DCD) est le diagramme de plus haut niveau de SART. Il montre les flots de données circulant entre le système à spécifier et les entités extérieures, les acteurs externes (terminaisons) avec lesquels le système communique.

L'objectif du diagramme de contexte est donc d'exposer le but principal du système.

Il met en oeuvre le système comme un processus unique auquel on adjoint les terminaisons ainsi que les flots de données associés.

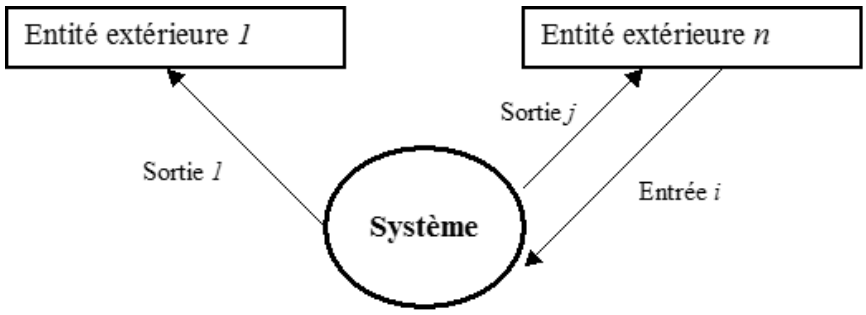


FIG. A.4 – *Diagramme de contexte*

Nous pouvons voir dans la figure A.4, le processus unique, symbolisé par un cercle, portant comme label le nom du processus ou la fonction principale représentative de ce que fait le système (sa mission).

Notre utilisation: Nous utiliserons les diagrammes de contexte pour représenter les agents et non le système (comme avec une analyse avec SART). Ainsi, les terminaisons ne représentent pas les entités extérieures du système mais les interfaces avec le monde extérieur (capteurs/effecteurs) : les terminaisons font donc partie intégrante de l'agent. De plus nos diagrammes SART font apparaître des flôts de contrôle (flèches en pointillées) en plus des flôts de données (fléches continues) comme nous pouvons le voir en (figure 5.12; page 86) ou (figure 9.8, page 177).

A.2.2 Graphe d'actions faisables

L'objectif du graphe des tâches est de montrer les dépendances entre les différentes actions faisables par l'agent ou qu'il a besoin que l'on fasse pour lui: il est en effet possible de définir des actions que l'agent n'est pas capable d'effectuer et qu'il déléguera au moment de l'exécution [Sichman et al., 1994].

Quatre catégories d'actions ont été identifiées:

1. Les *actions primitives* qui sont les tâches non physiquement décomposables.
2. les *actions primitives paramétrées* qui sont des actions créées à partir d'une action primitive que l'on paramètre.
3. Les *actions composées* qui sont des agencements de primitives.
4. Les *actions situées* qui nécessitent une partie de la représentation du monde pour instancier les actions composées ou les primitives.

Comme vu dans le chapitre 5, le qualificatif "physiquement" est utilisé dans la définition des primitives pour écarter certaines ambiguïtés. Si on considère par exemple le cas d'un moteur qui permet faire de avancer un mobile selon deux vitesses, $avancer\_lentement$ et $avancer\_vite$ ne seront pas des actions primitives mais des actions primitives paramétrées. D'un point de vue logique cela peut paraître étonnant mais c'est la nature physique de l'application qui nous oblige à prendre ce point de vue. En effet, sans la primitive $avancer$ on ne peut pas créer $avancer\_lentement$ et $avancer\_vite$. La réciproque n'aurait aucun sens.

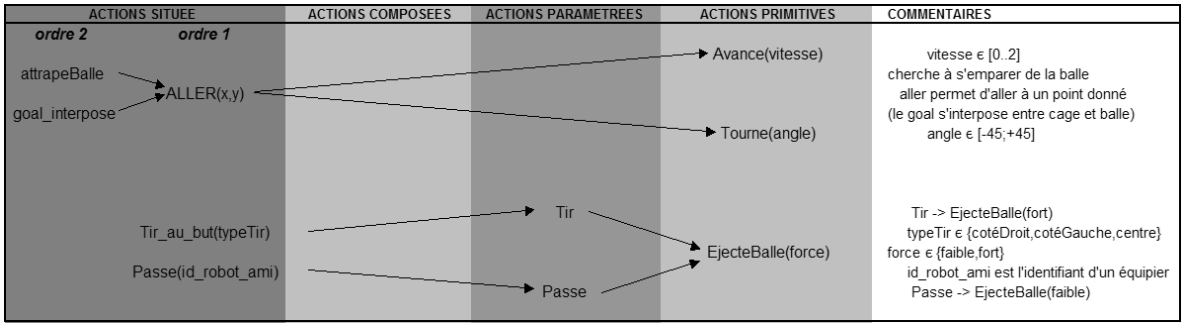


FIG. A.5 – *Graphe d'action faisables*

A.2.3 Graphs des perceptions faisables

L'ojectif du graphe des perceptions faisables est de montrer les dépendances entre les différentes perceptions faisables par l'agent. Il est possible de définir des perceptions que l'agent n'est pas capable d'effectuer (besoins en données) et qu'il déléguera au moment de l'exécution.

1. Les *perceptions primitives* qui sont les perceptions non décomposables.
2. Les *perceptions composées* qui sont construites à partir d'autres perceptions c'est-à-dire des éléments de la représentation du monde de l'agent (ils peuvent donc nécessiter l'interaction avec d'autres agents).

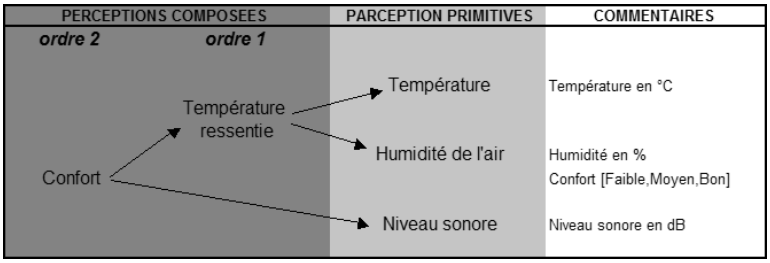


FIG. A.6 – *Graphe des perceptions faisables*

A.2.4 Diagrammes de séquence AUML

Source: AUML

Description: Les diagrammes AUML (Agent Unified Modeling Langage) sont des adaptations au contexte des systèmes multi-agents des diagrammes UML. Le diagramme de séquence AUML (figure A.7) est donc très semblable au diagramme de séquence UML (voir A.1.2). Des opérateurs spécifiques ont été ajoutés notament pour exprimer le choix etc. Il permet de décrire un protocole d'interaction mettant en jeux divers agents de différents rôles.

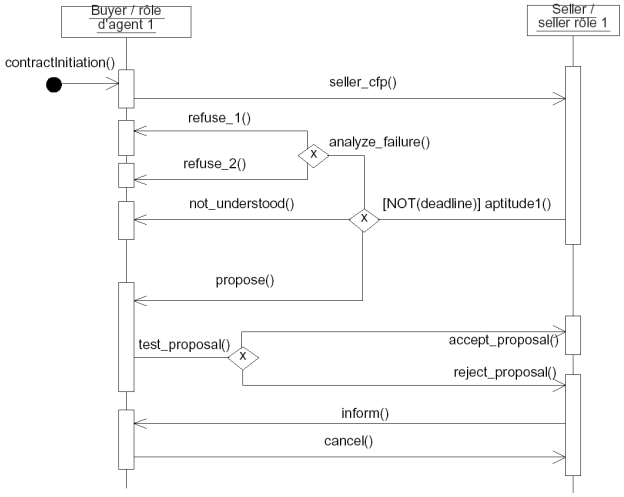


FIG. A.7 – *Diagramme de séquence AUML tiré de [Picard, 2004]*

A.2.5 Diagramme entités/relation

Source: MERISE - Modèle entité/relation

Description: Le diagramme entité-relation permet la définition du contenu d'une base de données.

Il met en jeu des *ensembles d'entités*. Les entités sont des objets (abstraits ou concrets) au sujet duquel on conserve de l'information dans la base de données. Une entité doit être identifiable c'est-à-dire qu'elle puisse être distinguée d'une autre entité. Les ensembles d'entités regroupent des entités semblables (de

même type), c'est-à-dire au sujet desquelles on veut conserver la même information.

Une relation entre ensembles d'entités $\{E_1,E_2,...,E_k\}$ est un sous-ensemble du produit cartésien $E_1 \times E_2 \times ... \times E_k$. C'est donc un ensemble de k-tuples $\{e_1,...,e_k\}/e_1 \epsilon E_1, e_2 \epsilon E_2,...,e_k \epsilon E_k$. k est le degré de la relation ainsi, si k = 2, la relation est dite binaire et si k = 3, elle est dite ternaire.

La participation d'un ensemble d'entités à une relation est son rôle. On donne un nom au rôle d'un ensemble d'entités dans une relation.

Ainsi dans la relation $PATERNITE : PERSONNE\,PERSONNE$ (voir figure A.8), l'ensemble d'entités PERSONNE à deux rôles distincts : "père" et "enfant".

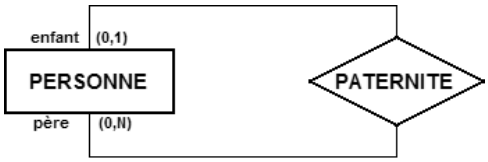


FIG. A.8 – *Schéma entité/relation*

Pour chaque ensemble d'entités participant à une relation, c'est-à-dire pour chaque rôle d'un ensemble d'entités, on précise dans combien de t-uples de la relation chaque entité peut apparaître: c'est la cardinalité.

Notre utilisation: Nous utilisons cette notation lorsque nous portons notre attention sur les liens entre les différents type d'agents : si il est plus pertinent, pour l'application, de focaliser sur les rôles en faisant abstraction des données. Cette notation nous parait plus contraignante, que la notation basée sur les diagrammes de classe, pour ce qui est la description de la structure organisationnelle (les liens de dépendance).

A.2.6 Diagramme de classes

Source: UML

Description: Les diagrammes de classes permettent de développer la structure des entités utilisées par les utilisateurs, la structure des classes, la structure des modules d'un langage de développement.

La figure A.9 illustre une description d'organisation via l'utilisation d'un diagramme de classe. Pour que cette description soit complète il faudrait évidement joindre une description textuelle afin de documenter les rôles et les liens.

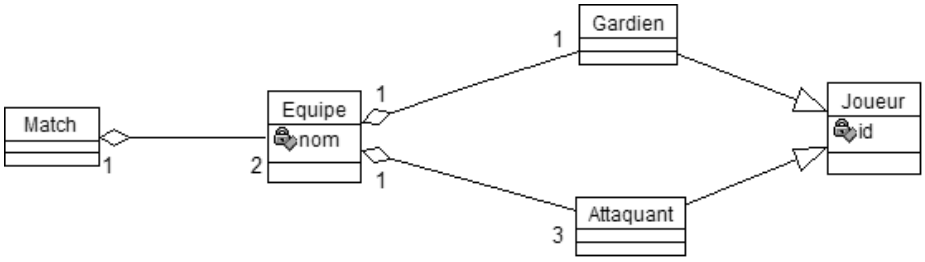


FIG. A.9 – *Diagramme de classe*

Notre utilisation: Nous utilisons cette notation lorsque nous portons notre attention sur le lien entre les rôles et les données.

A.3 Notations utilisées dans l'étape de conception générique

Dans l'étape de conception générique, après avoir choisi une architecture pour les agents, on précise le diagramme de contexte en spécifiant la nature de chacun des signaux/informations/contrôle: pour cela un tableau est utilisé (voir le tableau 5.3, page 93). Les autres activités de la conception générique font intervenir des composants. Chaque composant est décrit de manière externe (***Description externe de composant***) et de manière interne (***Description interne de composant)***.

A.3.1 Description externe de composant

Un composant, dans sa vue externe (voir 5.14, page 94) , se résume à un ensemble de ports d'entrée, de sortie et d'entrée/sortie. Nous représentons donc le composant sous la forme classique d'une boite noire dans laquelle figure son nom et/ou son symbole. Les ports en entrée figurent à gauche et les ports en sorties à droite. Chacun des ports est nommé. Un port d'entrée/sortie sera représenté par un port d'entrée et un port de sortie de même nom. Des tableaux permettent de spécifier les triplets (nom de port en entrée, type de donnée, nom du composant source) ou les triplets (nom de port en sortie, type de donnée, nom du composant destination).

A.3.2 Description interne de composant

Un composant, dans sa vue interne, doit permettre de modéliser l'évolution des informations sur les ports de sortie en fonction des informations en entrées et, éventuellement, de son état interne. Pour cela on peut utiliser un agencement de composants (voir figure 5.14, page 94), de FSM (voir figure 11.4, page 205) ou d'algorithme pour décrire le comportement du composant.

A.4 Notations utilisées dans l'étape d'implémentation

L'étape d'implémentation est une suite d'activité ne faisant pas intervenir de notations particulières : il s'agit d'un guide permettant d'aboutir à une implémentation de la solution conçue dans les étapes précédentes.

Annexe B

Introduction à VHDL

Cette annexe présente une rapide introduction au langage de description VHDL. Il ne s'agit en aucun cas d'un tutorial à VHDL mais bien d'une présentation ayant pour but de permettre à des personnes non initiées de se faire une idée de ce langage de description.

B.1 Introduction

Le VHDL est un langage de description développé dans les années 1980 aux États-Unis. En 1987, il est ensuite devenu la norme IEEE 1076 et fut révisé en 1993 pour supprimer quelques ambiguïtés et améliorer sa portabilité. Cette norme est vite devenue un standard en matière d'outils de description matériel. A ce jour, on utilise le langage VHDL pour concevoir des ASIC, programmer des composants programmables du type PLD, CPLD et FPGA, concevoir des modèles de simulations numériques etc.

Les électroniciens ont représenté les structures logiques ou analogiques des notations, outil de description graphique, mais la complexité des fonctions numériques à réaliser a imposé l'utilisation d'autres outils de description. C'est dans ce contexte, que le VHDL est apparu : il est en effet plus aisé de décrire un compteur ou un additionneur 64 bits en utilisant l'outil de description VHDL plutôt qu'un schéma!

Des travaux ont été effectués afin de procéder à des traductions de spécifications faites avec des langages issus des systèmes distribués tels que SDL [Glunz et al., 1993], LOTOS [Carreras et al., 1995] ou ESTELLE [Wytrebwicz and Budkowski, 1995].

B.2 Structure d'une description VHDL

D'une manière générale, tout composant matériel peut être vu comme une boite noire (l'entité) cachant un comportement permettant de faire évoluer des sorties en fonction des entrées.

FIG. B.1 – *Composant matériel*

La description interne d'une entité peut être élémentaire (figure B.2a) ou composée d'autres entités (figure B.2b).

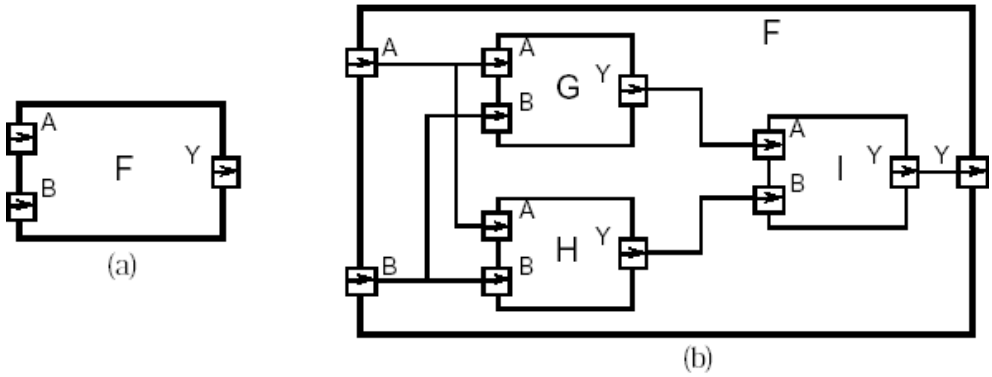


FIG. B.2 – *Entité élémentaire et entité composée (figure tirée de [Ashenden, 1990])*

Nous n'allons nous intèrésser uniquement à la description d'une entité élémentaire. Une description VHDL est constitué de deux parties distinctes:

1. La description externe de l'entité,
2. La description de l'architecture interne de l'entité.

On s'attache donc, dans un premier temps, à décrire l'entité d'un point de vue externe.

```
 ENTITY <nom_entité> IS
Description des entrées et des sorties de la structure en explicitant,
pour chacune d'entre elles, le nom, la direction, et le type.
<nom>:<direction ϵ {IN,OUT,INOUT}> <TYPE>;
END <nom_entité> ;
```

Dans un second temps, on va décrire l'architecture interne de l'entité.

```
 ARCHITECTURE <nom_architecture> OF <nom_entité> IS
Zone de déclaration.
BEGIN
Description de la structure logique.
END <nom_architecture> ;
```

B.3 Exemple de spécification VHDL

B.3.1 Cas d'un composant matériel : le registre 4 bits

Pour illustrer le langage de description VHDL, nous proposons la description d'un registre 4 bits (figure B.3).

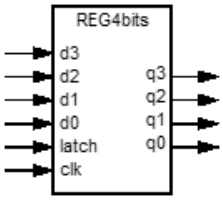


FIG. B.3 – *Registre 4bits*

```
-- Description externe
entity REG4bits is
  port (
     d0, d1, d2, d3: in bit; -- le mot de 4 bits en entrée
     latch: in bit;                  -- le signal de commande
     clk: in bit;                -- le signal d'horloge
     q0, q1, q2, q3:  out bit -- données de sortie
      );
end entity REG4bits;


\noindent
-- Description de l'architecture
architecture reg4bits of REG4bits is
begin
  process is
    variable d0_reg, d1_reg, d2_reg, d3_reg: bit;
  begin
    if en = '1' and clk = '1' then
      -- mémorisation des entrées
      d0_reg := d0;
      d1_reg := d1;
      d2_reg := d2;
      d3_reg := d3;
    end if;
    -- modification des signaux de sortie
    q0 <= d0_reg after 5 ns;
    q1 <= d1_reg after 5 ns;
    q2 <= d2_reg after 5 ns;
    q3 <= d3_reg after 5 ns;
    -- attente du prochain événement sur l'un des signaux d'entrée
    wait on d0, d1, d2, d3, en, clk;
  end process;
end architecture reg4bits;
```

B.3.2 Traduction d'un graphe d'état en VHDL

Dans cet exemple tiré de [Lagreze, 2003], nous allons maintenant nous intéresser à la traduction d'un graphe à état fini en VHDL (sujet abordé lors de l'exposé de la méthode DIAMOND). Nous prenons en considération le graphe (figure B.4) décrivant le fonctionnement d'une gâche électrique.

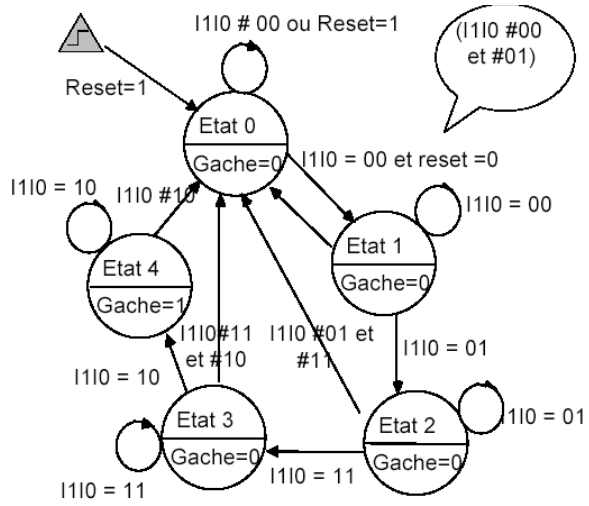


FIG. B.4 – *Graphe d'état décrivant la gestion d'une serrure électrique*

La traduction en VHDL de ce graphe d'état est:

```
entity serrure is port(
  clk, reset: in std_logic ;
  Inter: in std_logic_vector(1 downto 0);
  Gache: out std_logic );
end serrure;
architecture arch_serrure of serrure is
  type states is (state0, state1, state2, state3, state4);
  attribute state_encoding of states:type is one_hot_one;
  signal state: states;
begin
  etats: process (clk)
  begin
    if (clk'event and clk ='1') then
    if (reset = '1') then
      state <= state0;
    else
      case state is
        when state0 =>
          if (Inter="00") then
            state <= state1;
          else
            state <= state0;
          end if;

        when state1 =>
          if (inter="00") then
            state <= state1;
```

```
          elsif (inter="01") then
            state <= state2;
          else
            state <= state0;
          end if;

        when state2=>
          if (inter="01") then
            state <= state2;
          elsif (inter="11") then
            state <= state3;
          else
            state <= state0;
          end if;

        when state3 =>
          if (inter="11") then
            state <= state3;
          elsif (inter="10") then
            state <= state4;
          else
            state <= state0;
          end if;
          end if;

        when state4 =>
          if(inter="10") then
            state <= state4;
          else
            state <= state0;
          end if;
      end case;
    end if;
  end if;
end process;

  sorties: process (state)
  begin
    case state is
      when state0 =>
        Gache<=’0’;
      when state1 =>
        Gache<=’0’;
      when state2 =>
        Gache<=’0’;
      when state3 =>
        Gache<=’0’;
      when state4 =>
        Gache<=’1’; -- ouvre porte
    end case;
  end process;

end arch_serrure;
```

B.3.3 Assemblage de composants

Dans cet exemple tiré de [Lagreze, 2003], on spécifié une porte ET à trois entrées à partir de portes ET à deux entrées (voir figure B.5).

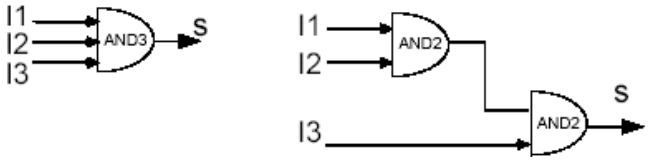


FIG. B.5 – *Construire une porte ET à quatre entrées à partir de deux portes ET à deux entrées*

On commence par définir un composant AND2 (pour porte ET à deux entrées).

```
entity AND2 is
  port(
    E1,E2: in std_logic;
    S: out std_logic);
end AND2;
architecture arch_AND2 of AND2 is
begin
  S <= E1 and E2;
end arch_AND2;
```

On construit dès lors la porte à trois entrées avec deux portes à deux entrées comme précisé en figure B.5.

```
library ieee;
use ieee.std_logic_1164.all;
entity AND3 is
  port(
    I1,I2,I3: in std_logic;
    S: out std_logic);
end AND3;
architecture arch_AND3 of AND3 is
component AND2
  port(E1,E2: in std_logic; S: out std_logic);
end component;
  signal Y: std_logic;
begin
  -- instanciation du composant AND2
  Porte1:AND2 port map (I1,I2,Y);
  Porte2:AND2 port map (I3,Y,S) ;
end arch_AND3;
```

Annexe C

Synthèse des activités de la méthode DIAMOND

Nous présentons un rappel des activités et un rappel des documents produits:

- Un rappel de l'objectif,
- Les sorties en terme de documents (quel document est produit?).

LEGENDE:
▶ : document produit (ces documents peuvent être ré-utilisés dans le processus).
⋆ : objectif du processus ou de l'activité selon les cas.

A. Définition des besoins

▶*Glossaire contextuel*

A.1 Approche préliminaire du problème

⋆*Comprendre le probléme*

▶*Fiches de présentation du projet*

▶*Fiches de recueil des besoins fonctionnels*

▶*Fiches des contraintes techniques*

▶*Fiches des contraintes technologiques*

A.2 Etudier les acteurs

⋆*Comprendre qui sont les utilisateurs*

A.2.1 Identifier les acteurs

A.2.2 Décrire les acteurs

▶*Fiches de documentation des acteurs*

A.3 Etudier les cas d'utilisation

⋆*Comprendre ce que l'on attend du système*

A.3.1 Identifier les cas d'utilisation

▶*Fiches de cas d'utilisation*

A.3.2 Décrire les cas d'utilisation

A.3.3 Organiser les cas d'utilisation

▶*Fiches de structuration des cas d'utilisation*

A.3.4 Documenter les cas d'utilisation

▶*Fiches de documentation des cas d'utilisation*

A.3.5 Illustrer les cas d'utilisation

▶*Fiches de synthèse des cas d'utilisation*

A.4 Etudier les besoins en services

⋆*Quels sont les échanges de messages?*

▶*Fiches de diagrammes de séquence*

▶*Fiches d'IHM*

A.5 Identifier les modes de fonctionnement

⋆*Structurer le fonctionnement global du système*

A.5.1 Déterminer les modes d'arrêt

▶*Fiches d'étude des modes d'arrêts*

A.5.2 Déterminer les modes de marche

▶*Fiches d'étude des modes de marches*

A.5.3 Déterminer les modes défaillants

▶*Fiches d'étude des modes de défaillants*

B. Analyse

▶*Glossaire d'analyse*

B.1 Phase de situation

B.1.1 Caractériser l'environnement

⋆*Quel est l'environnement du système?*

▶*Fiches de description de l'environnement*

B.1.2 Déterminer les entités actives et passives

⋆*Quels sont les entités baignées dans cet environnement?*

▶*Fiches d'identification des entités actives et passives*

B.1.3 Déterminer les agents

⋆*Que sont les agents?*

▶*Fiches de description de l'entité passive/active*

B.2 Phase individuelle

⋆*Permettre la conception future des agents autistes?*

▶*Glossaire d'analyse de l'agent*

B.2.1 Identifier l'aspect externe des agents

⋆*Idenfier les perceptions possibles de l'agent*

⋆*Idenfier les actions possibles de l'agent*

B.2.1.1 Etablissement du graphe des tâches

B.2.1.2 Etablir la représentation de l'environnement

B.2.1.3 Diagramme de contexte

▶*Fiches de contexte*

B.2.2 Identifier l'aspect interne des agents

⋆*Identifier les capteurs de l'agent*

⋆*Identifier les effecteurs de l'agent*

⋆*Analyser la réprésentation du monde de l'agent.*

▶*Fiches de schéma d'action*

▶*Fiches de schéma de perception*

B.2.3 Etablir le comportement des agents

⋆*Créer un comportement autiste de l'agent*

▶*Fiches de comportement individuel*

B.3 Phase sociale

⋆*Affecter une position sociale à l'agent*

B.3.1 Définir la connaissance des autres

⋆*Identifier ce que l'on percoit (et comment) des autres agents*

▶*Fiches de connaissance des autres*

B.3.2 Définir les primitives de communications

⋆*Identifier comment on peut communiquer avec les autres agents*

▶*Fiches des besoins en communication*

B.3.3 Définir les actions et perceptions collaboratives

⋆*Identifier les actions collectives*

⋆*Mettre à jour la représentation du monde de l'agent*

▶*Fiches des actions et perceptions collaboratives*

B.3.4 Définir l'aspect organisationnel

⋆*Identifier l'aspect social du système*

▶*Fiches d'aspect organisationnel*

▶*Fiches des comportements collectifs*

B.4 Phase d'intégration

⋆*Socialiser les agents : adapter comportement individuel et social*

B.4.1 Identifier l'influence sociale

⋆*Identifier l'influence social sur le comportement individuel*

▶*Fiches d'identification de l'influence sociale*

B.4.2 Définir le comportement social de l'agent

⋆*Etudier la possibilité de contrats, formation de coalition*

▶*Fiches de comportement social*

B.4.3 Assembler/adapter les comportements individuels et sociaux

⋆*Rendre sociable le comportement de l'agent autiste*

▶*Fiches d'adaptation des comportements individuels et sociaux*

C. Conception générique

▶*Glossaire de conception*

C.1 Définition du contexte

⋆*Faire des choix technologiques*

C.1.1 Choisir une architecture pour les différents agents

⋆*Choisir l'architecture adaptée au contraintes d'analyse*

▶*Fiches de choix architecturale*

C.1.2 Préciser le diagramme de contexte pour chaque type agent

⋆*Choisir les capteurs et effecteurs*

▶*Fiches de diagramme de contexte*

C.2 Construction des tâches applicatives de l'agent

⋆*Construire la coquille de l'agent*

C.2.1 Construire la coquille externe de l'agent

C.2.1 Construire la coquille interne de l'agent

▶*Fiches de composants*

C.3 Construction des modules de communication et des structures d'organisation

⋆*Construire les modules de communication*

⋆*Construite la structure organisationnelle de l'agent*

C.3.1 Construire les modules de communication

C.3.2 Construire la structure organisationnelle

▶*Fiches de composants*

C.4 Construction du contrôle de l'agent

⋆*Construire le controle de l'agent*

C.4.1 Construire le contrôle (partie environnementale)

C.4.2 Construire le contrôle sociale de l'agent

▶*Fiches de composants*

D. L'étape d'implantation

D.1 La phase de partitionnement

D.1.1 Déterminer si la question du partitionnement se pose

⋆*Identifier les parties qui nécéssitent un partitionnement*

D.1.2 Hiérarchiser les critères

⋆*Identifier ce qui doit être optimisé*

D.1.3 Déterminer le choix du partitionnement

⋆*Choisir ce qui deviendra logiciel et matériel*

D.1.4 Définir les interfaces logicielles/matérielles

▶*Fiches de critère de partitionnement*

▶*Fiches de partitionnement*

⋆*Choisir les solutions pour faire communiquer des composants*

D.2 La phase de co-simulation

⋆*Simuler le fonctionnement du système hybride*

▶*Fiches de cosimulation*

D.3 La phase de co-validation

⋆*Valider le système hybride*

▶*Fiches de covalidation*

D.4. La phase de réalisation

⋆*Générer les codes sources*

▶*Fiches de code source*

⋆*Générer les descriptions matérielles*

▶*Fiches de description architecturale*

▶*Fiches de co-synthèse*

D.5 Test

⋆*Tester le système créer*

Annexe D

Fiches vierges utilisées dans la méthode DIAMOND

Certaines fiches n'apparaissent pas en annexes car elles sont vierges (seul le titre et le cartouche sont présents). Il s'agit des fiches :

- une fiche de présentation du projet,
- une fiche de recueil des besoins fonctionnels,
- une fiche des contraintes techniques,
- une fiche des contraintes technologiques,
- une fiche de diagramme de séquences,
- une fiche IHM,
- une fiche de description de l'environnement,
- une fiche de caractérisation des entités passives et actives,

D'autre part certaines fiches sont similaires à d'autre :

- Fiche d'étude des modes de marches :
  - une fiche d'étude des modes d'arrêts (similaire à la figure D.7)
  - une fiche d'étude des modes défaillants (similaire à la figure D.7)
- Glossaire contextuel :
  - un glossaire contextuel.

Les fiches présentes sont :

- une fiche de glossaire contextuel (figure D.1),
- une fiche de recueil des besoins fonctionnels (figure D.2),
- une fiche de documentation des acteurs (figure D.3)
- une fiche de documentation de cas d'utilisation (figure D.4),
- une fiche de synthèse des cas d'utilisation (figure D.5).
- une fiche de structuration des cas d'utilisation (figure D.6).

– une fiche d'étude des modes de marche D.7

Enfin, il est possible que des fiches soient disponibles en blancs c'est-à-dire sans contenu. Par exemple, il est possible d'avoir une fiche de recueil des besoins fonctionnels différentes (le probléme peut nécéssiter une autre grille).

Date création:	Date de mise-à-jour:	Responsable:	Version:

GLOSSAIRE CONTEXTUEL

TERME	DESCRIPTION

FIG. D.1 – *Fiche de glossaire contextuel*

Date création:	Date de mise-à-jour:	Responsable:	Version:

FICHE DE RECUEIL DES BESOINS FONCTIONNELS

QUI	FAIT QUOI	COMMENT

FIG. D.2 – *Fiche de recueil des besoins fonctionnels*

Date création:	Date de mise-à-jour:	Responsable:	Version:

FICHE DE DOCUMENTATION D'ACTEUR

ACTEUR	ROLE

FIG. D.3 – *Fiche de documentation des acteurs*

Date création:	Date de mise-à-jour:	Responsable:	Version:

FICHE DE CAS D'UTILISATION

Titre:

But:

Résumé:

Acteurs:

Préconditions:

Enchaînements nominaux:

Enchaînements alternatifs:

Exeptions:

Postconditions:

FIG. D.4 – *Fiche de documentation de cas d'utilisation*

Date création:	Date de mise-à-jour:	Responsable:	Version:

FICHE DE SYNTHESE DES CAS D'UTILISATION

CAS D'UTILISATION	ACTEURS	MESSAGES EMIS/RECUS PAR LES ACTEURS

FIG. D.5 – *Fiche de synthèse des cas d'utilisation*

Date création:	Date de mise-à-jour:	Responsable:	Version:

FICHE DE STRUCTURATION DES CAS D'UTILISATION

CAS D'UTILISATION	ACTEURS	PACKAGE

FIG. D.6 – *Fiche de structuration des cas d'utilisation*

Date création:	Date de mise-à-jour:	Responsable:	Version:

FICHE DE MODE DE MARCHE

MODE DE MARCHE	DESCRIPTION

FIG. D.7 – *Fiche d'étude des modes de marches*

Annexe E

Publications

ARTICLES DE REVUE

Jean-Paul JAMONT, Michel OCCELLO, *Une approche multi-agents pour la gestion de la communication dans les réseaux de capteurs sans fil*, Revue Technique et Science Informatique, Hermès sciences, 2005, à paraître.

PUBLICATIONS INTERNATIONALES

Jean-Paul JAMONT, Michel OCCELLO, *DIAMOND: A physical multiagent systems codesign approach*, 2005 International Workshop on Multi-Agent Robotic Systems (MARS), INSTICC Press, 2005.

Jean-Paul JAMONT, Michel OCCELLO, André LAGREZE, *An energy efficient multiagent middleware for physical complex system*, 2005 International Workshop on Articificial Neural Networks and Intelligent Information Processing (ANNIIP), INSTICC Press, 2005.

Jean-Paul JAMONT, Michel OCCELLO, *Designing cooperative embedded systems using a multiagent approach: The DIAMOND method*, 2005 IFIP Artificial Intelligence Applications and Innovations - IFIP/AIAI'2005, Springer Verlag, Shi Z. and Li D. (Eds.), Beijin, China, 2005.

Jean-Paul JAMONT, Michel OCCELLO, André LAGREZE, *A decentralized self-organized approach for wireless sensor networks*, 2004 IFIP Design Methods and Applications for Distributed Embedded Systems - IFIP/DIPES'2004, Kluwer academic publisher, Kleinjohann B. and Gao G.R. and Kopetz H. and Kleinjohann L. and Rettberg A. (Eds.), Vol. 150, ISBN 1-4020-8148-0, Toulouse, France, 2004.

Jean-Paul JAMONT, Michel OCCELLO, *Using organizational structures emergence for maintaining functional integrity in embedded systems networks*, 2004 IFIP Artificial Intelligence Applications and Innovations - IFIP/AIAI'2004, Kluwer academic publisher, Bramer M. and Devedzic V. (Eds.), Vol. 154, ISBN 1-4020-8150-2, Toulouse, France, 2004.

Jean-Paul JAMONT, Michel OCCELLO, *An adaptive multiagent infrastructure for self-organized physical embodied systems : An application to wireless communication management*, 2004 Third IEEE In-

ternational Symposium on Advance Distributed Systems - ISADS'2003, Springer Verlag, LNCS n°Vol. 3061, ISBN 3-540-22172-7, Guadalajara, Mexique, 2003.

Jean-Paul JAMONT, Michel OCCELLO, *Using Self-Organization for Functional Integrity Maintenance of Wireless Sensors Networks*, 2003 IEEE/WIC/ACM International Conference on Intelligent Agent Technology - IAT'2003, IEEE Computer society, ISBN 0-7695-1931-8, Alifax, Canada, p.535-539, 13-16 oct. 2003.

Jean-Paul JAMONT, Michel OCCELLO, André LAGREZE, *A Multiagent System for the instrumentation of an underground hydrographic system*, 2002 IEEE International Symposium on Virtual and Intelligent Measurement Systems - VIMS'2002, IEEE Instumentation and Measurement Society, ISBN 0-7803-7344-8, Mt Alyeska Resort, AK, USA, p20-25,19-20 may 2002.

PUBLICATIONS NATIONALES AVEC ACTES

Jean-Paul JAMONT, Michel OCCELLO, *Un intergiciel par envoi de message économe en énergie basé sur une approche multi-agents*, 2004 Journées Francophones Ubiquité et Mobilité , ACM digital library.

Jean-Paul JAMONT, Michel OCCELLO, *Gestion adaptative de la communication dans les réseaux sans fil de systèmes physiques*, 2004 Journées Francophones sur les Systèmes Multi-Agents, Hermès sciences, ISBN 2-7462-1021-5.

Jean-Paul JAMONT, Michel OCCELLO, André LAGREZE, *Simulation du maintien d'intégrité fonctionnelle par système multi-agents pour un réseau de capteurs sans fil*, 2004 Journées Francophones sur les Systèmes Multi-Agents, Hermès sciences,, ISBN 2-7462-1021-5.

COLLOQUES INDUSTRIELS

Jean-Paul JAMONT, *Vers un AGL multi-agents pour la création de systèmes complexes mixtes logiciels/matériels*, Colloque de l'Agence Rhône-Alpes pour la Maîtrise des Technologies de la Mesure - ARATEM'2004, Lyon, France, 2004

Jean-Paul JAMONT, *Projet ENVSYS - Instrumentation non filaire de réseaux hydrographiques souterrains*, Colloque de l'Agence Rhône-Alpes pour la Maîtrise des Technologies de la Mesure - ARATEM'2003, Chambéry, France, 2003

Jean-Paul JAMONT, *Approche co-design pour la conception de systèmes complexes embarqués auto-organisés*, Colloque de l'Agence Rhône-Alpes pour la Maîtrise des Technologies de la Mesure - ARATEM'2002, Valence, France, 28 Novembre 2002

Bibliographie

[Abrial, 1996] Abrial, J.-R. (1996). *The B Book - Assigning Programs to Meanings*. Cambridge University Press.

[Adam, 2000] Adam, E. (2000). *Modèle d'organisation multi-agent pour l'aide au travail coopératif dans les processus d'entreprise : application aux systèmes administratifs complexes*, Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.

[Adams and Thomas, 1996] Adams, A. K. and Thomas, E. (1996). The design of mixed hardware/software systems. In *33rd Design Automation Conference*, pages 515–520.

[AFNOR, 2003] AFNOR (2003). ICT Motorola mise sur Java et J2ME. *Enjeux*, 233:22.

[Aggelou and Tafazolli, 1999] Aggelou, G. and Tafazolli, R. (1999). Relative distance micro-dicovery ad hoc routing (rdmar) protocol. Technical report, Internet Engineering Task Force.

[Agha et al., 2002] Agha, K. A., Pujolle, G., and Vivier, G. (2002). *Réseaux de mobiles et réseaux sans fil*. Eyrolles.

[Agimont, 1996] Agimont, G. (1996). *Modélisation et Simulation des Organisations Mutli-agents*, Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.

[Agresti, 1986] Agresti, W. W. (1986). What are the new paradigms? In Agresti, W. W., editor, *New Paradigms for Software Development*. IEEE Computer Society.

[Alissali, 1998] Alissali, M. (1998). *Introduction au Génie Logiciel*. Support de cours - Université du Maine - Le Mans.

[Allara et al., 2000] Allara, A., Bombana, M., Fornaciari, W., and Salice, F. (2000). A case study in design space exploration: The tosca environment applied to a telecommunication link controller. *IEEE Design & Test of Computers*, 17(2):60–72.

[Antoniu et al., 2004] Antoniu, G., Deverge, J.-F., and Monnet, S. (2004). Building fault-tolerant consistency protocols for an adaptative grid data-sharing service. In *ACM Workshop on Adaptative Grid Middleware*, pages 49–59.

[Arabshahi et al., 2001] Arabshahi, P., Gray, A., Kassabalidis, I., Das, A., and II, R. M. (2001). Adaptive routing in wireless communication networks using swarm intelligence. In *conference of the 9th AIAA Int. Communications Satellite Systems Conference*, pages 1–9.

[Ashcroft and Wadge, 1977] Ashcroft, E. A. and Wadge, W. W. (1977). Lucid, a nonprocedural language with iteration. *Communications of the ACM*, 20(7):519–526.

[Ashenden, 1990] Ashenden, P. J. (1990). *The VHDL Cookbook, First Edition*. Dept. Computer Science, University of Adelaide, South Australia.

[Aumiaux, 2000] Aumiaux, M. (2000). *Initiation au langage VHDL*. Dunod.

[Baeijs and Demazeau, 1996] Baeijs, C. and Demazeau, Y. (1996). Les organisations dans les systèmes multi-agents. In *5éme Journée Nationale du PRC-IA sur les Systèmes Multi-Agents*, pages 35–46.

[Baker and Ong, 2002] Baker, M. and Ong, H. (2002). A java embedded micro-kernel infrastructure. In *conference of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 224–224, New York, NY, USA. ACM Press.

[Banavar et al., 1999] Banavar, G., Chandra, T., Strom, R., and D.Sturman (1999). A case for message oriented middleware. In *DISC*, pages 1–18.

[Bar-Yam, 1997] Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Longman.

[Basagni et al., 1998] Basagni, S., Chlamtac, I., Syrotiuk, V., and Woodward, B. (1998). A distance routing effect algorithm for mobility (dream). In *conference of ACM/IEEE MOBICOM'98*, volume Dallas, Texas, pages 76–84.

[Beck, 1998] Beck, K. (1998). Extreme programming: A humanistic discipline of software development. In Astesiano, E., editor, *Fundamental Approaches to Software Engineering, 1st Internationsl Conference, FASE'98*, volume 1382 of *Lecture Notes in Computer Science*, pages 1–6. Springer.

[Belhadj, 1994] Belhadj, H. (1994). Overview on graphical entities for high-level description. In *IFIP Workshop on Logic and Architecture Synthesis*, Grenoble, France.

[Berger et al., 2003] Berger, M., Watzke, M., and Helin, H. (2003). Towards a fipa approach for mobile ad hoc environments. In *Proceeding of the 8th International Conference on Intelligence in next generation Networks*, Bordeaux, France.

[Berlage and Genau, 1993] Berlage, T. and Genau, A. (1993). A framework for shared applications with a replicated architecture. In *ACM Symposium on User Interface Software and Technology*, pages 249–257.

[Bernon et al., 2002] Bernon, C., Gleizes, M.-P., Peyruqueou, S., and Picard, G. (2002). Adelfe: A methodology for adaptive multi-agent systems engineering. In *Engineering Societies in the Agents World III, Third International Workshop, ESAW 2002*, volume LNCS N°2577, pages 156–169, Spain. Springer.

[Berrebi, 1997] Berrebi, E. (1997). *Méthodologie pour l'application industrielle de la synthèse comportementale*, Thèse de doctorat, Institut National Polytechnique de Grenoble.

[Berry and Gonthier, 1992] Berry, G. and Gonthier, G. (1992). The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152.

[Birman, 1985] Birman, K. P. (1985). Replication and fault-tolerance in the isis system. In *ACM Operating Systems Review*, pages 79–85.

[Birrel and Ould, 1985] Birrel, N. and Ould, M. (1985). *A practical Handbook for Software Development*. Cambridge University Press.

[Bénard et al., 2002] Bénard, J.-L., Bossavit, L., Médina, R., and Williams, D. (2002). *L'Extreme Programming - Avec deux études de cas* -. Eyrolles.

[Boehm, 1976] Boehm, B. W. (1976). Software engineering. *IEEE Trans. Computers*, 25(12):1226–1241.

[Boehm, 1988] Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72.

[Boissier and Demazeau, 1996] Boissier, O. and Demazeau, Y. (1996). Asic: An architecture for social and individual control and its application to computer vision. In John W. Perram and Jean-Pierre Müller, editor, *Distributed Software Agents and Applications, 6th European Workshop on Modelling Autonomous Agents - MAAMAW '94*, volume 1069, pages 1–18, Denmark. Springer.

[Boissier et al., 2004] Boissier, O., Gitton, S., and Glize, P. (2004). Caractéristiques des sytèmes et des applications. In des Techniques Avancées, O. F., editor, *Systèmes Multi-Agents (Série ARAGO)*, volume 29, pages 25–54.

[Bollen and Heylighen, 1996] Bollen, J. and Heylighen, F. (1996). Algorithms for the self-organisation of distributed, multi-user networks. possible application to the future world wide web. In Trappl, R., editor, *Cybernetics and Systems '96*, pages 911–916.

[Bottoni et al., 2000] Bottoni, P., Koch, M., Parisi-Presicce, F., and Taentzer, G. (2000). Consistency checking and visualization of ocl constraints. In *The Unified Modeling Language, Advancing the Standard, Third International Conference*, pages 294–308.

[Boughlam, 1988] Boughlam, A. (1988). Définition d'un modèle, notes de travail sadt. Technical report, IGL Technology.

[Bourdon, 2001] Bourdon, F. (2001). *http://www.iutc3.unicaen.fr/ bourdf/cours/DEA-I3/index.html*.

[Brazier et al., 2002] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2002). Principles of component-based design of intelligent agents. *Data Knowledge Engineering*, 41(1):1–27.

[Bronevetsky et al., 2004] Bronevetsky, G., Marques, D., Pingali, K., Szwed, P. K., and Schulz, M. (2004). Application-level checkpointing for shared memory programs. In *Proceedings of the 11th International Conference on Architectural Support for Programming Langages and Operating Systems - ASPLOS'04*, pages 235–247, Boston, USA.

[Buchenrieder et al., 1993] Buchenrieder, K., Sedlmeier, A., and Veith, C. (1993). Hw/sw co-design with prams using codes. In Agnew, D., Claesen, L. J. M., and Camposano, R., editors, *11th IFIP conference of the International Conference on Computer Hardware Description Languages and their Applications - CHDL '93*, volume A-32, pages 65–78. North-Holland.

[Buck et al., 1994] Buck, J., Ha, S., Lee, E. A., and Messerschmitt, D. G. (1994). Ptolemy: A framework for simulating and prototyping heterogenous systems. *International Journal in Computer Simulation*, 4(2).

[Burrafato and Cossentino, 2002] Burrafato, P. and Cossentino, M. (2002). Designing a multi-agent solution for a bookstore with the passi methodology. In *AOIS '02, Agent-Oriented Information Systems, conference of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002 at CAiSE*02)*, volume 57 of *CEUR Workshop conference*.

[Calderoni et al., 1997] Calderoni, S., Courdier, R., Leman, S., and Marcenac, P. (1997). Construction expérimentale d'un modèle multi-agents. In JFIADSMA, editor, *European Conference on Cognitive Science*, pages 109–123, La Colle-sur-Loup France.

[Calisti et al., 2004] Calisti, M., Lozza, T., and Greenwood, D. (2004). An agent-based middleware for adaptative roaming in wireless networks. In *AAMAS 2004 Worshop on Agents for Ubiquitous Computing*, New York, USA.

[Call, 1994] Call, J. M. (1994). Quality factors. In *Encyclopædia of Software Engineering*, volume 1, pages 958–969. John Wiley & Sons.

[Calvez et al., 1997] Calvez, J., Pasquier, C., and Heller, D. (1997). Hardware/software system design based on the mcse methodology. In J.M. Bergé, O. Levia, J. R., editor, *Current Issues in Electronic Modeling*, volume 9, pages 1–36. Kluwer Academic Publishers.

[Campo et al., 2002] Campo, C., Marin, A., Garcya-Rubion, C., and Breuer, P. (2002). Service discovery in pervasive multiagent systems. In *AAMAS2002 - conference of the Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices*. ACM.

[Camps, 1995] Camps, V. (1995). *Vers une théorie de l'auto-organisation dans les systèmes multi-agents basée sur la coopération: application à la recherche d'information dans un système d'information répartie*, Thèse de doctorat, Université Paul Sabatier.

[Camps, 1998] Camps, V. (1998). Application of a self-organizing method based on cooperation to information retrieval. In *13th European Conference on Artificial Intelligence*, pages 313–314. John Wiley & Sons.

[Camps and Gleizes, 1995] Camps, V. and Gleizes, M. (1995). Principes et évaluation d'une méthode d'auto-organisation. In *Actes des 3émes Journées du GDR-PRC IA/SMA*, pages 337–348, St Baldoph France. Hermès.

[Cannon, 1932] Cannon, W. B. (1932). *Wisdom of the Body*. Norton, London.

[Carabelea and Boissier, 2004] Carabelea, C. and Boissier, O. (2004). Multi-agent platforms on smart devices : Dream or reality ? In *conference of the Smart Objects Conference*.

[Carabelea et al., 2003] Carabelea, C., Boissier, O., and Ramparany, F. (2003). Benefits and requirements of using multi-agent systems on smart devices. In Kosch, H., Böszörményi, L., and Hellwagner, H., editors, *Euro-Par*, volume LNCS n°2790, pages 1091–1098. Springer.

[Carreras et al., 1995] Carreras, C., López, J. C., Lopez, M. L., Sánchez, L., Delgado-Kloos, C., and Martinez, N. (1995). From lotos to vhdl. In Bergé, J.-M., Levia, O., and Rouillard, J., editors, *Series Current Issues in Electronic Modelling*, volume 3. Kluwer Academic Publishers.

[Caspi et al., 1987] Caspi, P., Pilaud, D., Halbwachs, N., and Plaice, J. (1987). Lustre: A declarative language for programming synchronous systems. In *Symposium on Principles of Programming Languages*, pages 178–188.

[Castor et al., 2004] Castor, A., Pinto, R., Silva, C. T. L. L., and Castro, J. (2004). Towards requirement traceability in tropos. In Ridao, M. and Cysneiros, L. M., editors, *Workshop em Engenharia de Requisitos*, pages 189–200.

[Cerri and Loia, 1997] Cerri, S. A. and Loia, V. (1997). A concurrent, distributed architecture for diagnostic reasoning. *User Model. User-Adapt. Interact.*, 7(2):69–105.

[Chen and Talwar, 2003] Chen, D. and Talwar, V. (2003). Extending Java Virtual Machines for networked embedded devices. *Research Disclosure*, 475:475/076.

[Chen et al., 2000] Chen, H., Chakraborty, D., Xu, L., Joshi, A., and Finin, T. (2000). Service discovery in the future electronic market. In *conference of the 17th National Conference on Artificial Intelligence*, pages 1–6, Austin, Texas, USA. AAAI Press/MIT Press.

[Chen and Gerla, 1998] Chen, T. and Gerla, M. (1998). Global state routing: A new routing scheme for ad-hoc wireless networks. In *conference of IEEE International Conference on Communications - ICC'98*, Atlanta.

[Chiang et al., 1997] Chiang, C., Wu, H., Liu, W., and Gerla, M. (1997). Routing in clustered multihop, mobile wireless networks. In *conference of IEEE Singapore International Conference on Networks*, pages 197–211, Singapore.

[Chiozzotto and Lenail, 1998] Chiozzotto, L. and Lenail, B. (1998). Java et l'informatique industrielle. *Jautomatise*, (5).

[Choudhury et al., 2000] Choudhury, R., Paul, K., and Bandyopadhyay, S. (2000). Topology discovery in ad-hoc wireless networks using mobile agents. In *conference of the 2nd ACM International Workshop on Mobile Agents for Telecommunication Applications*, pages 1–16. Springer-Verlag.

[Clausen et al., 2001] Clausen, T., Jacquet, P., Laouiti, A., Muhlethaler, P., Qayyum, A., and L.Viennot (2001). Optimized link state routing protocol. In *IEEE International Multitopic Conference*, Pakistan.

[Clohessy, 2001] Clohessy, K. (2001). Virtual machine technology: Managing complexity and providing protability for embedded systems. *Dedicaced Systems Magazine*, (5):58–60.

[Collinot and Drogoul, 1998] Collinot, A. and Drogoul, A. (1998). Using the cassiopeia method to design a robot soccer team. *Applied Artificial Intelligence*, 12(2-3):127–147.

[Comité National De Danse Sportive, 1999] Comité National De Danse Sportive (1999). *Réglement Technique, http://membres.lycos.fr/pourladanse/rtcnds.doc*. Fédération Française de Danse.

[Cook et al., 1996] Cook, D. J., Gmytrasiewicz, P. J., and Holder, L. B. (1996). Decision-theoretic cooperative sensor planning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(10):1013–1023.

[Cormen et al., 2002] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (2002). *Introduction à l'algorithmique : Cours et exercices*. Dunod.

[Cottet, 2001] Cottet, F. (2001). *LabVIEW : Programmation et applications*. Dunod.

[Das et al., 1997] Das, B., Sivakumar, R., and Bharghavan, V. (1997). Routing in ad hoc networks using a spine. In *conference of the International Conference On Computer Communications and Networks (ICCCN 1997)*, pages 34–41. IEEE.

[De Man et al., 1995] De Man, H., Bolsens, I., Lin, B., Van Rompaey, K., Vercauteren, S., and Verkest, D. (1995). Co-design of dsp systems. In *NATO ASI Hardware/Software Co-Design*.

[De Micheli, 1995] De Micheli, G. (1995). *Hardware/Software Co-Design*. North Atlantic Treaty Organization (NATO).

[DeLoach et al., 2001] DeLoach, S., Wood, M., and Sparkman, C. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258.

[Demazeau, 1993] Demazeau, Y. (1993). La plate-forme paco et ses applications. In *Actes des 2émes Journées du GDR-PRC IA/SMA*, Montpellier France.

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behavior in agent-based systems. In *European Conference on Cognitive Science*, Saint-Malo France.

[Demazeau and Costa, 1996] Demazeau, Y. and Costa, A. R. (1996). Populations and organisations in open multi-agent systems. In *1st Symposium on Parallel and Distributed AI*, Hyderabad, India.

[Défago et al., 1998] Défago, X., Schiper, A., and Sergent, N. (1998). Semi-passive replication. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, page 43, Washington,DC , USA.

[Diaz et al., 1998] Diaz, G., Mammeri, Z., and Thomesse, J.-P. (1998). Communication de groupe dans les applications multimédia coopératives : une synthèse. In *NOTERE'98 - Colloque International sur les NOuvelles TEchnologies de la REpartition*.

[Donnadieu and Karsky, 2002] Donnadieu, G. and Karsky, M. (2002). *La systémique, penser et agir dans la complexité*. LIAISONS.

[Drogoul, 1993] Drogoul, A. (1993). *De la Simulation Multi-agents à la Résolution Collective de Problèmes*, Thèse de doctorat, Université Paris VI.

[Dube et al., 1997] Dube, R., Rais, C., Wang, K.-Y., and Tripathi., S. (1997). Signal stability based adaptive routing for ad hoc mobile networks. In *conference of IEEE Personal Communications*, pages 36–45.

[Durand, 1998] Durand, D. (1998). *La systémique*. Que sais-je ?, 2nd édition, PUF.

[E. Sternheim, 1993] E. Sternheim, R. Singh, Y. T. (1993). *Digital Design With Verilog Hdl*. Chapman & Hall.

[Eker et al., 2003] Eker, J., Janneck, J. W., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, J., and Xiong, Y. (2003). Taming heterogeneity - the ptolemy approach. volume 91, pages 127–144.

[Equipe MAGMA, 2002] Equipe MAGMA (2002). Rapport d'activités 1998-2002. Technical report, Institut IMAG, Laboratoire LEIBNIZ, Equipe MAGMA.

[Ericsson, 2003] Ericsson (2003). Industry upstarts: US-based SavaJe has developed a pure Java handset OS which it wants to sell to operators directly. *Mobile Communications International*, 100:48.

[Ernst et al., 1993] Ernst, R., Henkel, J., and Benner, T. (1993). Hardware-software cosynthesis for microcontrollers. In *IEEE Design and Test of Computers*, volume 10, pages 64–75.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes Multi-Agents : vers une intelligence collective*. InterEditions.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *3rd International Conference on Multi-Agent Systems (ICMAS 1998), 3-7 July 1998, Paris, France*, pages 128–135.

[FIPA, 2002a] FIPA (2002a). *FIPA ACL Message Representation in Bit-Efficient Specification, Specification number SC00069*. http://www.fipa.org/fipa00069/.

[FIPA, 2002b] FIPA (2002b). *FIPA Agent Message Transport Envelope Representation, Specification number SC00088*. http://www.fipa.org/fipa00088/.

[Foisel et al., 1996] Foisel, R., Chevrier, V., and Haton, J.-P. (1996). De l'organisation d'une société à sa ré-orgranisation. In *Journée Systèmes Multi-Agents*, pages 121–128. IRIT.

[Foukia and Hassas, 2003] Foukia, N. and Hassas, S. (2003). Towards self-organizing computer networks: A complex system perspective. In Serugendo, G. D. M., Karageorgos, A., Rana, O. F., and Zambonelli, F., editors, *Engineering Self-Organising Applications*, pages 77–83.

[Fox, 1981] Fox, M. (1981). An organizational view of distributed systems. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 11, pages 70–80.

[Franklin and Graesser, 1996] Franklin, S. and Graesser, A. C. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *conference of the Third International Workshop on Agent Theories, Architectures and Languages (ATAL'96)*, pages 21–35.

[Gajski et al., 1998] Gajski, D., Vahid, F., Narayan, S., and Gong, J. (1998). System-level exploration with specsyn. In *conference of the 35th Conference on Design Automation*, pages 812–817. ACM Press.

[Gallou and Bouchon-Meunier, 1994] Gallou, F. L. and Bouchon-Meunier, B. (1994). *Introduction à la Systémique*. Hermès.

[Garro, 2003] Garro, A. (2003). Modeling-notation source: Gaia. Technical report, Foundation for Intelligent Physical Agents.

[Gasser, 1992] Gasser, L. (1992). An overview of dai. In *Distributed Artificial Intelligence : Theory and Praxis*, Boston. Kluwer Academic Publishers.

[Gerla and Tsai, 1995] Gerla, M. and Tsai, J. T.-C. (1995). Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265.

[Glunz et al., 1993] Glunz, W., Kruse, T., Rössel, T., and Monjau, D. (1993). Integrating sdl and vhdl for system-level hardware design. In Agnew, D., Claesen, L. J. M., and Camposano, R., editors, *conference of the 11th IFIP WG10.2 International Conference on Computer Hardware Description Languages and their Applications*, volume A-32, pages 187–204. North-Holland.

[Goldberg et al., 2001] Goldberg, D., Li, M., Tao, W., and Tamir, Y. (2001). The design and implementation of a fault-tolerant cluster manager. Technical report, University of California, Los Angeles.

[Graham, 1999] Graham, N. (1999). The nicola mark ii / a new rescue radio for france. In *The CREG Journal*, volume 38, pages 3–6.

[Gray, 2000] Gray, R. (2000). Soldiers, agents and wireless networks: A report on a military application. In *conference of the Fifth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, Manchester, England.

[Grode et al., 1998] Grode, J., Voigt Knudsen, P., and Madsen, J. (1998). Hardware resource allocation for hardware/software partitioning in the lycos system. In *Design, Automation and Test in Europe (DATE '98)*, pages 22–27.

[Groupe MARCIA, 1996] Groupe MARCIA (1996). Auto-organisation := évolution de structure(s). In *PRC-GDR Intelligence artificielle*, pages 139–152. Hermes.

[Guichard, 1996] Guichard, F. (1996). *La réorganisation dynamique dans les systèmes multi-agents*, Thèse de doctorat, Université de Savoie.

[Guillemet et al., 1999] Guillemet, A., Haik, G., Meurisse, T., Briot, J., and Lhuillier, M. (1999). Mise en oeuvre d'une approche componentielle pour la conception d'agents. In Gleizes, M. and Marcenac, P., editors, *Septièmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'99)*, pages 53–65. Hermès Science Publications, Paris, France.

[Gupta and De Micheli, 1993] Gupta, R. and De Micheli, G. (1993). Hardware-software cosynthesis for digital systems. In *IEEE Design and Test of Computers*, volume 10, pages 29–41.

[Haegg, 1997] Haegg, S. (1997). A sentinel approach to fault handling in multi-agent systems. In Zhang, C. and Lukose, D., editors, *conference of the 2nd Australian Workshop on Distibuted Artificial Intelligence*, volume LCNS n°1286, pages 181–195, Cairns, Australia. Springer.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274.

[Helin and Laukkanen, 2002] Helin, H. and Laukkanen, M. (2002). Performance analysis of software agent communication in slow wireless networks. In *conference of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, pages 354–361, Austin.

[Herlea et al., 1999] Herlea, D., Jonker, C., Niek, J., and Wijngaards, J. E. (1999). Specification of bahavioural requirements within compositional multi-agent system design. In *conference of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume LNCS n°1647, pages 8–27. Springer.

[Heylighen, 1999] Heylighen, F. (1999). The science of self-organization and adaptivity. In *Computational and Mathematical Theory of Organizations*, volume 5(3), pages 253–280.

[Hoare, 2004] Hoare, C. A. R. (2004). *Communicationg Sequential Processes*. http://www.usingcsp.com/cspbook.pdf.

[Huang et al., 2001] Huang, H.-P., Liand, C.-C., and Lin, C.-W. (2001). Construction and soccer dynamics analysis for an integrated multi-agent soccer robot system. In *Natl. Sci. Counc. ROC(A)*, volume 25, pages 84–93.

[Hübner et al., 2002] Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *16th Brazilian Symposium on Artificial Intelligence*, pages 118–128.

[Humphrys, 1995a] Humphrys, M. (1995a). Towards self-organising action selection. In Steel, S., editor, *Papers of the 14th Workshop of the UK Planning and Scheduling Special Interest Group (SIGPLAN 14)*.

[Humphrys, 1995b] Humphrys, M. (1995b). W-learning: Competition among selfish q-learners. Technical report, University of Cambridge.

[Idiagram Consulting, 2005] Idiagram Consulting (2005). *http://www.idiagram.com*.

[Iglesias et al., 1998] Iglesias, C., Garrijo, M., and Gonzales, J. (1998). A survey of agent oriented methodologies. In *conference of ATAL 98 - Workshop on Agent Theories, Architectures, and Languages*, volume LNAI 1555, pages 163–176, Paris, France. Springer-Verlag.

[Ishida and Yokoo, 1992] Ishida, T. and Yokoo, M. (1992). Organization self-design of distributed production systems. In *IEEE Transactions on Knowledge and Data Engineering*, volume 4, pages 123–134.

[Ishida et al., 1990] Ishida, T., Yokooand, M., and Gasser, L. (1990). An organizational approach to adaptive production systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 52–58, Boston. AAAI Press.

[Ismail, 1996] Ismail, M. B. (1996). *Synthèse au niveau système et conception de systèmes mixtes logiciels/matériels*, Thèse de doctorat, Institut National Polytechnique de Grenoble.

[ISO/IEC, 1988] ISO/IEC (1988). *LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection, Genève, September*.

[Iwata et al., 1999] Iwata, C., Chiang, C., Pei, G., Gerla, M., and Chen, T. (1999). Scalable routing strategies for ad hoc wireless networks. In *IEEE Journal on Selected Areas in Communications*, pages 1369–1379.

[Jackson and Simpson, 1975] Jackson, K. and Simpson, H. (1975). Mascot - a modular approach to software construction, operation and test. In *RRE Tech. Note*, volume 78.

[Jean, 1997] Jean, M. R. (1997). Emergence et sma. In Quinqueton, J., Thomas, M.-C., and Trousse, B., editors, *Journées Francophones sur l'Intelligence Artificielle Distribuée et Systèmes Multi-Agents (JFIADSMA'97)*, pages 323–342, Colle-Sur-Loup. Hermes.

[Jiang et al., 1999] Jiang, M., Li, J., and Tay, Y. (1999). Cluster based routing protocol. Technical report, IETF - Internet Engineering Task Force.

[Jiang et al., 1998] Jiang, M., Li, J., and Tay, Y. C. (1998). Cluster based routing protocol (cbrp) functional specification. Technical report, National University of Singapore.

[Joa-Ng and Lu, 1999] Joa-Ng, M. and Lu, I. (1999). A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks. In *IEEE Journal on Selected Areas in Communications*, pages 1415–1425.

[Johnson and Maltz, 1996] Johnson, D. and Maltz, D. (1996). Dynamic source routing in ad hoc wireless networks. In Imielinski, T. and Korth, H., editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers.

[Joloboff et al., 2000] Joloboff, V., Aguado, J., Balle, C., Banâtre, M., Berry, G., Duhaut, C., Fabre, J., Flamand, E., Folliot, B., Houeix, P., Nicolas, C., Ouabdesselam, F., Saive, C., and Terrier, F. (2000). Rntl - rapport du groupe de travail b1 : Systèmes embarqués. Technical report, Réseau National de Technologie Logicielle.

[Jones and Sheeran, 1990] Jones, G. and Sheeran, M. (1990). Circuit design in ruby. *Formal Methods for VLSI Design*.

[Kadionik, 2004] Kadionik, P. (2004). *Cours de l'option Systèmes Embarqués*. Ecole Normale Supérieure D'Electronique, Informatique et Radiocommunications de Bordeaux - ENSEIRB.

[Kalavade, 1995] Kalavade, A. (1995). *System-level codesign of mixed hardware-software systems*. PhD thesis, Université de Californie - Berkeley.

[Kalavade and Lee, 1993] Kalavade, A. and Lee, E. A. (1993). A hardware-software codesign methodology for dsp application. In *IEEE design and Test of Computers*, volume 10, pages 16–28.

[Kendall and Malkoun., 1997] Kendall, E. and Malkoun., M. (1997). Design Patterns for the Development of Multi-Agents Systems. In Zhang, C. and Lukose, D., editors, *Multi-Agents Systems : Methodologies and Applications, conference of Second Australian Workshop on DAI*, volume LNAI 1286, pages 17–32, Cairns, Australia. Springer-Verlag.

[Kinny et al., 1996] Kinny, D., Georgeff, M., and Rao, A. (1996). A methodology and modelling technique for systems of bdi agents. In Van de Velde, W. and Perram, J. W., editors, *conference of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume LNCS n°1038, pages 56–71. Springer.

[Kleinrock and Stevens, 1971] Kleinrock, L. and Stevens, K. (1971). Fisheye: A lenslike computer display transformation. Technical report, UCLA, Computer Science Department.

[Knublauch, 2002] Knublauch, H. (2002). Extreme programming of multi-agent systems. In *Proceeding of Autonomous Agent and Multi-Agent System*, pages 704–711, Italy,. ACM.

[Ko and Vaidya, 1998] Ko, Y. and Vaidya, N. (1998). Location-aided routing (lar) in mobile ad hoc networks. In *conference of ACM/IEEE International Conference on Mobile Computing and Networking - MOBICOM'98*, pages 66–75, Texas, USA. ACM.

[Kolski, 1997] Kolski, C. (1997). *Interfaces Homme-Machine, application aux systèmes industriels complexes (2éme édition revue et corrigée)*. Hermes.

[Koning et al., 1995] Koning, J.-L., Demazeau, Y., Esfandiari, B., and Quinqueton, J. (1995). Quelques perspectives d'utilisation des langages et protocoles d'interaction dans le contexte des télécommunications. In *JFIADSMA'95 - 3émes Journées Francophone d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, Chambéry.

[Kornman, 1993] Kornman, S. (1993). *SADE : un Système Réflexif de Surveillance à Base de Connaissances*, Thèse de doctorat, Université Paris VI.

[Koudil, 2002] Koudil, M. (2002). *Une approche orientée objet pour le codesign*, Thèse d'état, Institut National d'Informatique d'Alger.

[Koudil, 2004] Koudil, M. (2004). *Cours intitulé -Méthode de conception conjointe des systèmes embarqués-*. Institut National de formation en Informatique (Alger).

[Krishna et al., 1995] Krishna, P., Chatterjee, M., Vaidya, N. H., and Pradhan, D. K. (1995). A cluster-based approach for routing in ad-hoc networks. In *Symposium on Mobile and Location-Independent Computing*, pages 1–10. USENIX.

[Kruchten and Kroll, 2003] Kruchten, P. and Kroll, P. (2003). *Guide pratique du RUP*. CampusPress.

[Ku and De Micheli, 1990] Ku, K. and De Micheli, G. (1990). Hardwarec - a language for hardware design version - 2.0. Technical report, Standford University.

[Kumar, 2002] Kumar, S. (2002). *The Codesign of Embedded Systems: A Unified Hardware Software Representation*. Kluwer Academic Publishers.

[Kumar et al., 1993] Kumar, S., Aylor, J., Johnson, B., and Wulf, W. (1993). A framework for hardware/software codesign. In *IEEE Computer*, pages 39–45.

[Lagreze, 2003] Lagreze, A. (2003). *Cours d'architectures embarquées -VHDL-*. Support de cours - Université Pierre Mendès-France - Valence.

[Langley, 2003] Langley, B. (2003). Discovery of infrastructure in multi-agent systems. In *conference of second international joint conference on autonomous agent and multiagent systems*, Melbourne, Australia.

[Larvet, 1994] Larvet, P. (1994). *Analyse des systèmes : de l'approche fonctionnelle à l'approche objet*. InterEditions.

[Laurgeau et al., 2002] Laurgeau, C., Steux, B., and Metman, G. (2002). Outils pour le prototypage des systèmes embarqués temps réel. *Revue Jautomatise*, 21:58–62.

[Le Moigne, 1990] Le Moigne, J.-L. (1990). *La modélisation des systèmes complexes*. Dunod.

[Le Strugeon, 1995] Le Strugeon, E. (1995). *Une méthodologie d'auto-adaptation d'un système multi-agents cognitifs*, Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.

[Le Strugeon et al., 1993] Le Strugeon, E., Mandiau, R., and Libert, G. (1993). Proposition d'organisation dynamique d'un groupe d'agents en fonction de la tâche. In *Actes des 1ére Journées Francophones Intelligence Artificielle Distribuée et Systèmes Multi-Agents - JFIADSMA*, Toulouse, France.

[Lee and Benford, 1995] Lee, O.-K. and Benford, S. (1995). An explorative model for federated trading in distributed computing environments. In *conference of the International Conference on Open Ditributed Processing*.

[Legras, 2002] Legras, F. (2002). Écoute flottante et communications locales pour la formation de groupes. In *Journées Francophones sur les Systèmes Multi-Agents*, pages 31–43. Hermès.

[Lemlouma, 2000] Lemlouma, T. (2000). Le routage dans les réseaux ad-hoc, Mémoire de maîtrise, Université des Sciences et de la Technologie Houari Boumèdiene.

[Lenay, 1996] Lenay, C. (1996). Emergence et sma. In *Actes des Journées Francophones Intelligence Artificielle Distribuée et Systèmes Multi-Agents - JFIADSMA*, pages 265–272, Sète, France. Hermes.

[Lind, 2001] Lind, J. (2001). *Interative Software Engineering for multiagent systems: The MASSIVE Method*, volume 2001 of *LNCS/LNAI*. Springer Verlag.

[Malone, 1987] Malone, T. (1987). Modeling coordination in organizations and markets. In *Management science*, volume 33, pages 1317–1332.

[Mamei and Zambonelli, 2003] Mamei, M. and Zambonelli, F. (2003). Self-organization in multi-agent systems : a middleware approach. In *AAMAS 2003 Workshop on Engineering Self-Organising Systems*, pages 233–248.

[Marca and McGowan, 1987] Marca, D. A. and McGowan, C. L. (1987). *Sadt: Structured Analysis and Design Techniques*. Cambridge University Press.

[Marcelpoil et al., 1994] Marcelpoil, R., Beaurepaire, E., and Pesty, S. (1994). La sociologie cellulaire: modéliser et simuler une 'société' cellulaire pour étudier le vivant. *Intellectica*, 19.

[Marcenac, 1996] Marcenac, P. (1996). Emergence of behaviors in natural phenomena agent-simulation. In *Complexity International Review*, volume 3.

[Marcenac, 1997a] Marcenac, P. (1997a). Modélisation et Simulation par agents - Application aux systèmes complexes (HDR).

[Marcenac, 1997b] Marcenac, P. (1997b). The multiagent approach : Complex simulations that spew realistic behaviors require independent acting variables. In *IEEE-Potentials*, pages 19–23.

[March and Simon, 1958] March, J. and Simon, H. (1958). Organizations. New-York.

[McNeile, 1986] McNeile, A. T. (1986). Jackson system development (jsd). In *Information Systems Design Methodologies: Improving the Practice*, pages 225–246.

[Meurisse and Briot., 2001] Meurisse, T. and Briot., J. (2001). Une approche à base de composants pour la conception d'agents. *Revue Technique et Science Informatiques (TSI), Numéro Spécial "Réutilisation"*, 20(4):583–602.

[Milanosic et al., 2004] Milanosic, N., Malek, M., Davidson, A., and Milutinovic, V. (2004). Routing and security in mobile ad hoc networks. In Carver, D. L., editor, *IEEE Computer*, volume 37, pages 61–65. IEEE Computer Society.

[Milner, 1985] Milner, R. (1985). Lectures on a calculus for communicating systems. In Brookes, S. D., Roscoe, A. W., and Winskel, G., editors, *Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA, July 9-11, 1984*, volume 197 of *Lecture Notes in Computer Science*, pages 197–220. Springer.

[Milner, 1999] Milner, R. (1999). *Communicating and mobile systems: the pi-calculus*. Mcgraw-Hill.

[Minar et al., 1999] Minar, N., Kramer, K., and Maes, P. (1999). Cooperating mobile agents for dynamic network routing. Springer-Verlag. in *Software Agents for Future Communications Systems*.

[Mintzberg, 1982] Mintzberg, H. (1982). *Structure et Dynamique des Organisations*. Organisation.

[Müller and Van Dyke Parunak, 1998] Müller, J.-P. and Van Dyke Parunak, H. (1998). Vers une méthodologie de conception de systèmes multi-agents de réolution de probléme par émergence. In *JFIAD-SMA -Journée Francophones IAD et SMA*, Pont-à-Mousson, France. Hermes.

[Moreno and Peulot, 1997] Moreno, S. and Peulot, E. (1997). *Le gemma: Modes de marches et d'arrêts, GRAFCET de coordination des tâches, conception des systèmes automatisés de production sûrs*. Casteila.

[Moulin and Brassard, 1995] Moulin, B. and Brassard, M. (1995). A scenario-based design method and an environment for the development of multiagent systems. In Zhang, C. and Lukose, D., editors, *First Australian Workshop on Distributed Artificial Intelligence: Architecture and Modelling*, volume LNCS n°1087, pages 216–232. Springer.

[Murthy and Garcia-Luna-Aceves, 1995] Murthy, S. and Garcia-Luna-Aceves, J. (1995). A routing protocol for packet radio network. In *conference of the IEEE MOBICOM'95*, pages 86–95.

[Mylopoulos et al., 2001] Mylopoulos, J., Manuel, K., and Castro, J. (2001). Uml for agent-oriented software development: The tropos proposal. In Gogolla, M. and Kobryn, C., editors, *conference of the 4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools*, volume LNCS n°2185, pages 422–441. Springer.

[Narayan et al., 1992] Narayan, S., Vahid, F., and Gajski, D. D. (1992). System specification with the speccharts language. *IEEE Design & Test of Computers*, 9(4):6–13.

[Newman, 2004] Newman, D. J. (2004). Embedded Java controllers. *Circuit Cellar*, 166:16–21.

[Ni et al., 1999] Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., and Sheu, J.-P. (1999). The broadcast storm problem in a mobile ad hoc network. In *MOBICOM*, pages 151–162.

[Niang et al., 2004] Niang, P., Grandpierre, T., Akil, M., and Sorel, Y. (2004). Aaa and syndex-ic: A methodology and a software framework for the implementation of real-time applications onto reconfigurable circuits. In Becker, J., Platzner, M., and Vernalde, S., editors, *conference of the 14th International Conference in Field Programmable Logic and Application*, pages 1119–1123.

[Occello, 2003] Occello, M. (2003). Méthodologie et architectures pour la conception de systèmes multi-agents. In *Mémoire d'Habilitation à Diriger les Recherches*, Université Joseph Fourier, Grenoble.

[Occello et al., 2002] Occello, M., Baeijs, C., Demazeau, Y., and Koning, J.-L. (2002). *MASK : An AEIO Toolbox to Develop Multi-Agent Systems, Knowledge Engineering and Agent Technology,*. IOS Series on Frontiers in AI and Applications.

[Occello and Demazeau, 1998] Occello, M. and Demazeau, Y. (1998). Modelling decision making systems using agents for cooperation in a real time constraints. In *3rd IFAC Symposium on Intelligent Autonomous Vehicles*, volume 1, pages 51–56, Madrid, Spain.

[Occello et al., 2001] Occello, M., Koning, J., and Baeijs, C. (2001). Conception de systèmes multi-agents : quelques éléments de réflexion méthodologique. In *Technique et science informatiques*, volume 20, pages 233–263.

[Olukotun et al., 1994] Olukotun, K., Helaihek, R., Levitt, J., and Ramirez, R. (1994). A software-hardware cosynthesis approach to digital system simulation. In Micro, I., editor, *Technique et science informatiques*, pages 48–58.

[Padgham and Winikoff, 2002] Padgham, L. and Winikoff, M. (2002). Prometheus: A methodology for developing intelligent agents. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, Workshop on Agent oriented software engineering*, pages 174–185.

[Park and Corson, 1997] Park, V. and Corson, M. (1997). A highly adaptative distributed routing algorithm for mobile wireless networks. In *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOMM'97*, pages 1405–1413, Kobe, Japan.

[Perkins and Royer, 1999] Perkins, C. and Royer, E. (1999). Ad-hoc on-demand distance vector routing. In *conference of the 2nd IEEE Workshop on Mobile Computing Systems and Applicqtions*, pages 90–100.

[Petriu et al., 2002] Petriu, E., Patry, G., Whalen, T., Al-Dhaher, A., and Groza, Z. (2002). Intelligent robotic sensor agents for enviroment monitoring. In *conference of the International Symposium on Virtual and Intelligent Measurement Systems*, pages 14–19.

[Picard, 2004] Picard, G. (2004). *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente*, Thèse de doctorat, Université Paul Sabatier, Toulouse III.

[Pinet, 2002] Pinet, C. (2002). *Processus d'ingénierie du logiciel - Méthode et qualité -*. Pearson Education.

[Piquemal-Baluard and Glize, 1996] Piquemal-Baluard, C. and Glize, P. (1996). Des aptitudes non cognivistes d'agents pour l'auto-organisation. In *Journée Systèmes Multi-Agents*, pages 129–138. IRIT.

[Pitrat, 1990] Pitrat, J. (1990). *Métaconnaissance, futur de l'intelligence artificielle*. Hermès.

[PLEIAD, 1992] PLEIAD (1992). Vers une taxinomie de vocabulaire pour les systèmes multi-agents. In *PRC-GDR'92 - Actes de la Journée Système Multi-Agents*.

[Pleisch and Schiper, 2001] Pleisch, S. and Schiper, A. (2001). Approaches to fault-tolerant mobile agent execution. Technical report, IBM Research Report.

[Pountain, 1987] Pountain, D. (1987). *A tutorial introduction to occam programming*. Blackwell Scientific Publications, Oxford.

[Pujolle, 2000] Pujolle, G. (2000). *Les Réseaux, 3éme édition mise à jour*. Eyrolles.

[Quinnell, 2002] Quinnell, R. A. (2002). Embedded java. *TechOnLine, www.techonline.com*.

[Rao and Georgeff, 1995] Rao, A. and Georgeff, M. (1995). Bdi agents : from theory to practice. In *conference of 1st International Conference on Multi-Agent Systems ICMAS*, pages 312–319. AAAI Press.

[Ratsimor et al., 2002] Ratsimor, O., Chakraborty, D., Tolia, S., Kushraj, D., Kunjithapatham, A., Gupta, G., Joshi, A., and Finin, T. (2002). Allia: Alliancebased service discovery for adhoc environments. In *ACM Mobile Commerce Workshop*, Atlanta, USA. ACM.

[Rethman and Wilsey, 1993] Rethman, N. and Wilsey, P. (1993). Rapid: A tool for hardware/software tradeoff analysis. In *conference of IFIP Conference on Hardware Description Languages*, Ottawa, Canada. Elsevier Science.

[Rodriguez, 1994] Rodriguez, M. (1994). *Modélisation d'un agent autonome: Approche constructiviste de l'architecture de contrôle et de la représentation de connaissances*, Thèse de doctorat, Université de Neufchâtel.

[Roques and Vallée, 2003] Roques, P. and Vallée, F. (2003). *UML en action*. Eyrolles.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial Intelligence : a Modern Approach*. Prentice-Hall.

[Sansores and Pavón, 2004] Sansores, C. and Pavón, J. (2004). An agent architecture for wireless computing. In Evans, R., editor, *International Workshop on Practical Applications of Agents and Multiagent Systems (IWPAAMS 2003)*.

[Schira, 2003] Schira, O. (2003). Conception d'une couche physique adaptative pour réseau de terrain en vue de l'instrumentation de réseaux hydrographiques souterrains, Mémoire de DEA Signal Image Parole Télécom , Institut National Polytechnique de Grenoble.

[Schroeder et al., 1996] Schroeder, M., De Almeida Mora, I., and Pereira, L. M. (1996). A deliberative and reactive diagnosis agent logic programming. In *Eigth International Conference on Tools with Artificial Intelligence*, pages 436–437, Toulouse. IEEE Computer Society.

[Sedmak and Evans, 1995] Sedmak, R. M. and Evans, J. (1995). A hierarchical, desgin-for-testability (dft) methodology for the rapid prototyping of application-specific signal processors (rassp). In *conference IEEE International Test Conference*, pages 319–327.

[Shah and Zelikovsky, 2004] Shah, C. and Zelikovsky, A. (2004). Reactive connected dominating set for dynamic source routing, http://alla.cs.gsu.edu/ chintan/dsr/cds_dsr.pdf. Technical report, Georgia State University, Atlanta.

[Shapiro et al., 2001] Shapiro, M., Kermarrec, A.-M., Bowstron, A., and Druschel, P. (2001). The ice-cube approach to the reconciliation of divergent replicas. Technical report, Microsoft Research Cambridge.

[Sichman et al., 1994] Sichman, J., R., C., and Demazeau, Y. (1994). A social reasoning mechanism based on dependence networks. In *conference of 1st National Symposium on Parallel and Distributed Artificial Intelligence (PDAI)*, pages 111–116, Hyderabab, India.

[Simon, 1991] Simon, H. A. (1991). *Sciences des systèmes, science de l'artificiel*. Dunod.

[Skazinski, 2003] Skazinski, J. (2003). Automation revolutionizes embedded systems diagnostics. *Embedded Computing Design*, winter:28–29.

[Smith, 1980] Smith, R. G. (1980). The contract net protocol : High-level communication and control in a distributed problem solver. In *IEEE transactions on computers*, volume C-39 n12, pages 1104–1113.

[So and Durfee, 1993] So, Y. and Durfee, E. (1993). An organizational self-design model for organizational change. In *Working Notes of the AAAI-93 Workshop on Artificial Intelligence and Theories of roups and Organizations*.

[So and Durfee, 1996] So, Y.-P. and Durfee, E. (1996). Designing tree-structured organizations for computational agents. In *Computational and Mathematical Organization Theory*, volume 2(3), pages 219–246.

[Sorel, 1994] Sorel, Y. (1994). Syndex v4.2 user guide. Technical report, INRIA.

[Spivey, 1988] Spivey, J. (1988). Understanding z : A specification language and its formal semantics. In *Cambridge Tracts in Theoretical Computer Science*, volume 3. Cambridge University Press.

[Steels, 1990] Steels, L. (1990). Cooperation between distributed agents through self-organisation. In Demazeau, Y. and Müller, J.-P., editors, *Decentralized AI*. Elsevier Science Publishers B.V. (North Holland).

[Stoy, 1995] Stoy, E. (1995). *A Petri Net Based Unified Representation for Hw/Sw codesign*. PhD thesis, Departement of Computer and Information Science of Linköping University.

[Sun Microsystems, 2000] Sun Microsystems (2000). J2me building blocks for mobile devices - white paper on kvm and the connected, limited device configuration (cldc). Technical report.

[Talpin et al., 2004] Talpin, J. P., Gamatie, A., Berner, D., LeDez, B., and LeGuernic, P. (2004). Hard real-time implementation of embedded software in Java. *Lecture Notes in Computer Science*, 2952:33–47.

[Terry et al., 1995] Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., and Hauser, C. (1995). Managing update conflict in bayou, a weakly connected replicated storage system. In *the Fifteenth ACM Symposium on Operating System Principles*, volume Operating System Review 29(5), pages 172–183, Cooper Mountain, Colorado. ACM Press.

[Theißinger et al., 1994] Theißinger, M., Stravers, P., and Veit, H. (1994). Castle: an interactive environment for hw-sw co-design. In *conference of the Third International Workshop on Hardware/Software Codesign*, pages 203–209. IEEE Computer Society.

[Thomas et al., 1988] Thomas, D. E., Dirkes, E., Walker, R., Rajan, J., Nestor, J., and Blackburn, R. (1988). The system architect's workbench. In *conference of the 25th ACM/IEEE Conference on Design Automation*, pages 337–343, USA.

[Toh, 1997] Toh, C. (1997). Associativity based routing for ad hoc mobile networks. In *Wireless Personnal Communications Journal*.

[Tripakis and Courcoubetis, 1996] Tripakis, S. and Courcoubetis, C. (1996). Extending promela and spin for real time. In *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, conference*, volume 1055 of *Lecture Notes in Computer Science*, pages 329–348. Springer.

[Turner, 1992] Turner, K. (1992). *Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL*. John Wiley and Sons Ltd.

[Tveit, 2001] Tveit, A. (2001). A survey of agent-oriented software engineering. In *NTNU Computer Science Graduate Student Conference*, Dragvoll, Trondheim.

[Vahid and Givargis, 2002] Vahid, F. and Givargis, T. (2002). *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons.

[Valderrama, 1998] Valderrama, C. A. (1998). *Prototype virtuel pour la génération des architectures mixtes logicielles/matérielles*, Thèse de doctorat, Institut National Polytechnique de Grenoble.

[Van Dyke Parunak, 2000] Van Dyke Parunak, H. (2000). A practitioners? review of industrial agent applications. *Autonomous Agents and Multi-Agent Systems*, 3(4):389–407.

[Venture Development Corp., 2003] Venture Development Corp. (2003). *Embedded Systems Bulletin*. Venture Development Corp.

[Vercouter, 2000] Vercouter, L. (2000). *Conception et mise en oeuvre de systèmes multi-agents ouverts et distribués*, Thèse de doctorat, Université de Jean Monnet et Ecole des Mines de Saint-Etienne.

[Vercouter, 2004] Vercouter, L. (2004). Mast : Un modèle de composant pour la conception de sma. In *Journées Multi-Agents et Composants*, Paris.

[von Bertalanffy, 1932] von Bertalanffy, L. (1932). *Theoretische Biologie*. Borntraeger, Berlin.

[Ward, 1985] Ward, P. T. (1985). *Structured Development for Real-Time Systems: Introduction and Tools*. Yourdon.

[Wasserman et al., 1985] Wasserman, A. I., Pircher, P. A., and Shewmake, D. T. (1985). A RAPID/USE tutorial. Technical report, Medical Information Science, University of California, San Francisco. Release 1.3.

[Watkins, 1989] Watkins, C. (1989). *Learning from delayed reward*. PhD thesis, University of Cambridge.

[Weiser, 1993] Weiser, M. (1993). Some computer science issues in ubiquitous computing. In *ACM*, volume 7, pages 75–84.

[Weiss, 1999] Weiss, G. (1999). Multiagent systems and distributed artificial intelligence. In Weiss, G., editor, *Multiagent systems : A modern approach to Distributed Artificial Intelligence*. MIT Press.

[Wolf, 1994] Wolf, W. (1994). Hardware/software codesign of embedded systems. In CHERCHER, A., editor, *A CHERCHER*, volume 82, pages 967–989. IEEE.

[Woo et al., 1994] Woo, N., Dunlop, A., and Wolf, W. (1994). Codesign from cospecification. In *IEEE Computer*, volume 27, pages 42–47.

[Wooldridge, 1999] Wooldridge, M. (1999). Intelligent agents. In Weiss, G., editor, *Multiagent systems : A modern approach to Distributed Artificial Intelligence*. MIT Press.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. (1995). Intelligent agents : Theory and practice. In *Knowledge Enginneering Review*.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. In *Autonomous Agents and Multi-Agent Systems*, volume 3, pages 285–312. Kluwer Academic Publishers.

[Wytrebwicz and Budkowski, 1995] Wytrebwicz, S. and Budkowski, S. (1995). Communication protocols implemented in hardware : Vhdl generation from estelle. In *Current Issue in Electronic Modelling*, volume 3, pages 77–98.

[Xu and Xin, 2005] Xu, K. and Xin, L. (2005). Novel 32bit embedded reduced-instruction-set-computer with Java extension. *Mini-Micro Systems*, 26(1):90–95.

[Yeh, 1990] Yeh, R. T. (1990). An alternate paradigm for software evolution. In Ng, P. A. and Yeh, R. T., editors, *Modern Software Engineering: Foundations and Perspectives*. Van Nostrand Reinhold.

[Yiyu et al., 2005] Yiyu, T., Man, L. K., Pak, L. M., Shing, Y. W., and Anthony, S. F. (2005). 10.4-4: A Java processor for mobile devices. *IEEE International Conference on Consumer Electronics*, 23RD:437–438.

[Zhang et al., 2002] Zhang, W., Deng, Z., Wang, G., Wittenburg, L., and Xing, Z. (2002). Distributed problem solving in sensor networks. In *AAMAS'02*, pages 15–19.

[Zimmermann, 1999] Zimmermann, J. (1999). Design and implementation of darx. Technical report, Laboratoire d'Informatique de Paris 6.

[Zurawski, 2005] Zurawski, R., editor (2005). *The Industrial Information Technology Handbook*. CRC Press.

Liste des abréviations

Pour des raisons de lisibilité, la signification d'une abréviation ou d'un acronyme est rappelée à sa première apparition dans le texte d'un chapitre.

A

AAII	Australian Artificial Intelligence Institute (methodology)
ABR	Associativity Based Routing
ACK	Acknowledge
AEIO	Agent Environnement Interaction Organisation
AODV	Adhoc On-demand Distance Vector
ASIC	Architecture for Social and Individual Control
ASIC	Application Specific Integrated Circuit
ASIP	Application Specifif Instruction set Processor
ASTRO	Agent Spécialisé Temps-Réél Organisé
AUML	Agent Unified Modelling Langage
ASAM	Adaptive Service Access Management

B

BDD	Base de Données
BDI	Beliefs Desires Intentions

C

CAN	Controler Area Network
CASE	Computer Aided Software Engineering
CBRP	Cluster Based Routing Protocol
CCS	Calculus of Communicating Systems
CISC	Complex Instruction Set Computer
CGSR	Clusterhead Gateway Switch Routing
co-design	hardware/software Concurrent Design
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
CSP	Communicating Sequential Processes

D

DESIRE	DEsign and Specification of Interacting REasoning framework
DIAMOND	Decentralized Iterative Approach for Multiagent Open Networks Design
DFC	Diagramme de Flot de Contrôle
DFD	Data Flow Diagrams / Diagramme de Flot de Données
DREAM	Distance Routing Effect Algorithm for Mobility
DRP	Dynamic Routing Protocol
DSDV	Destination Sequence Distance Vector
DSP	Digital Signal Processor
DSR	Dynamic Source Routing

E

e-ASTRO	embedded ASTRO
EAUML	Extended Agent Unified Modelling Langage
EEPLD	Electrically Erasable Programmable Logic Device
ENVSYS	Environnement System
EPLD	Erasable Programmable Logic Device
EVM	Embedded Virtual Machine

F

FIPA	Foundation for Intelligent Physical Agent
FITT	Fond Incitatif de Transfert de Technologie
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FSR	Fisheye State Routing

G

GEMMA	Guide d'Etude des Modes de Marche et d'Arrêt.
GSR	Global State Routing

H

HDL	Hardware Description Langage
HiperLAN	High-Performance Radio Local Area Network
HSR	Hierarchical State Routing

I

IA	Intelligence Artificielle
IAD	Intelligence Artificielle Distribuée
IBM	International Business Machine
IDSF	International Dance Sport Federation
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IHM	Interface Homme Machine
ISO	International Standart Organization (A vérifier)

J

JADE	Java Agent DEvelopment Framework
JSD	Jackson System Development

K

KVM	K Virtual Machine

L

LABVIE	Laboratory Virtual Instrument Engineering WorkBench
LAN	Local Area Network
LANMAR	Landmark Ad Hoc Routing Protocol
LAR	Location-Aided Routing
LOTOS	Logical Temporal Ordering Specification

M

MAMOSACO	Méthode Adaptable de Modélisation de Systèmes Administratifs COmplexes
MAN	Metropolitan Area Network
MANET	Mobile Ad-hoc Network
MAS	MultiAgent System
MASC	MultiAgent Systems Codesign
MASCOT	Modular Approach to Software Construction Operation and Test
MASE	MultiAgent Systems Engineering methodology
MASSIVE	MultiAgent SystemS Iterative View Engineering
MCD	Modèle Conceptuel des Données
MCT	Modèle Conceptuel des Traitements
MESSAGE	Methodology for Engineering Systems of Sotware AGEnts
MLD	Modèle Logique de Données
MOT	Modèle oPérationnel des Traitements
MOISE	Model of Oganization for multI-agent SystEms
MPD	Modèle Physique des Données
MPT	Modèle Physique des Traitements
MWAC	Multi-Wireless-Agent Communication

N

NECSI	New England Complex Systems Institute

O

OLSR	Optimized Link State Routing
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First

P

PAL	Programmable Array Logic
PAN	Personnal Area Network
PASSI	Process for Agent Societies Specification and Implementation
PCN	Personnal Communication Network
PDA	Personnal Digital Assistant
PFA	Path Finding Algorithm
PLD	Programmable Logic Device

R

RAD	Rapid Application Development
RAM	Random Access Memory
RDMAR	Relative Distance Micro-Discovery Adhoc Routing
RDP	Réseaux de Pétri
RIP	Routing Internet Protocol
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RNTL	Réseau National de Technologie Logicielle
RREP	Route REPly
RREQ	Route REQuest
RUP	Rational Unified Process

S

SADT	System Analysis and Design Technic
SART	Structured Analysis for Real-Time systems
SCAO	Système Critique Auto-Organisé
SDL	Specification and Description Langage
SMA	Système Multi-Agents
SMAO	Système multi-agents Ouvert
SMAOA	Système multi-agents Auto-Organisé
SoC	System on Chip
SRP	Static Routing Protocol
SSR	Signal Stability-based Routing
SST	Signal Stability Table

T

TORA	Temporary Ordering Routing Algorithm

U

UML	Unified Modelling Langage
URREP	Unsolicited Route REPly

V

VHDL	Very High Description Langage

W

WAN	Wide Area Network
WRP	Wireless Routing Protocol
WSN	Wireless Sensor Network

X

XP	Extreme Programming

Z

ZHLS	Zone-Based Hierarchical Link State

Index

E

DIAMOND : Une Approche Pour La Conception De Systèmes Multi-Agents Embarqués

Résumé: Cette thèse propose une méthode pour l'analyse de problèmes relevant des systèmes complexes physiques ouverts avec des systèmes multi-agents physiques. Cette méthode que nous appelons DIAMOND (Decentralized Iterative Approach for Multiagent Open Networks Design) agence quatre phases en un cycle de vie en spirale. Elle propose d'utiliser, pour le recueil des besoins, des notations d'UML mais elle structure le fonctionnement global du système via une étude de modes de marche et d'arrêt. Elle utilise le raffinement notamment entre le niveau local et le niveau global du système et assemble les comportements individuels et les comportements sociaux tout en identifiant les influences de l'un sur l'autre. Elle guide le concepteur durant la phase de conception générique en utilisant les composants comme unité opératoire. En fin de cycle, le partitionnement logiciel/matériel du système intervient et permet la génération du code ou des descriptions matérielles.

Il n'était pas suffisant de proposer une méthode : considérer les composants des systèmes complexes physiques comme des noeuds coopérants d'un réseau sans fil est une démarche attrayante qui peut être vue comme la traduction physique extrême de la décentralisation. De fait, des besoins spécifiques en architectures doivent être traités. Pour cela, nous proposons le modèle MWAC (Multi-Wireless-Agent Communication) qui repose sur l'auto-organisation des entités du système.

Ces deux contributions sont exploitées au sein de l'application EnvSys qui a pour objectif l'instrumentation d'un réseau hydrographique.

Mot-clés: Méthode multi-agents, approche co-design, systèmes multi-agents ouverts, auto-organisation, intégrité fonctionnelle, réseaux maillés de capteurs, réseaux ad-hoc, capteurs intelligents.

DIAMOND : An Embedded Multiagent Systems Design Approach

Abstract: This thesis propose multiagent method to modeled physic complex systems. The DIAMOND (Decentralized Iterative Approach for Multiagent Open Networks Design) method is built to design physical multiagent system. Four main stages, distributed on a spiral cycle, may be distinguished within our physical multiagent design approach. The definition of needs defines what the user needs and characterizes the global functionalities. The second stage is a multiagent-oriented analysis which consists in decomposing a problem into a multiagent solution. The third stage of our method starts with a generic design which aims to build the multiagent system, once one knows what agents have to do without distinguishing hardware/software parts. Finaly, the implementation stage consists in partitioning the system in a hardware part and a software part to produce the code and the hardware synthesis.

Considering complex embedded control systems as networks of decentralized cooperative nodes is an attractive way to design physical intelligent applications. We propose the MWAC (Multi-Wireless-Agent Communication) model to manage communication in such complex systems.

These contribution are used in an underground wireless sensor networks application : the EnvSys project.

Keywords: Multiagent oriented method, codesign approach, open multiagent system, self-organization, fault tolerance, wireless sensor network, adhoc network, intelligent sensor.