Septembre 1998

Filière Informatique et Mathématiques Appliquées - DEA Représentation de la Connaissance et Formalisation du Raisonnement

---

Rapport de stage de DEA

Supervision de processus complexes

Manuel ORIOL

Lieu: LAAS-CNRS, Toulouse
Groupe: Robotique et Intelligence Artificielle
Responsable E.N.S.E.E.I.H.T.: Fabrice Evrard
Responsables de stage: Malik GHALLAB (LAAS) et Félix INGRAND (LAAS)

Remerciements

Avant tout, je remercie M. Laprie, Directeur du L.A.A.S. pour m'avoir permis d'effectuer mon stage au sein de ce laboratoire.

Je tiens à remercier tout particulièrement M. Ghallab, Responsable du groupe Robotique et Intelligence Artificielle du L.A.A.S. et Directeur de Recherche au C.N.R.S., de m'avoir accueilli dans son équipe et M. Ingrand, Chargé de Recherche au C.N.R.S., pour avoir suivi mon travail tout au long de l'année et avoir fait preuve d'une disponibilité de tous les instants.

Ce stage n'aurait certainement pas pu se passer dans d'aussi bonnes conditions sans l'aide amicale que m'ont apportée les thésards et stagiaires du groupe, et l'accueil chaleureux qu'ils m'ont réservé. Qu'ils en soient ici remerciés, en particulier Olivier qui s'est montré d'une patience à toute épreuve et enfin le «groupe de la cantine».

Table des matières

Table des figures

Chapitre 1

Présentation du laboratoire

1.1 Le LAAS-CNRS

Le LAAS (voir figure 1.1) est une Unité Propre de Recherche du Centre National de la Recherche Scientifique (**CNRS**), unité associée à trois établissements universitaires de Toulouse : l'Université Paul Sabatier, l'Institut National des Sciences Appliquées et l'Institut National Polytechnique de Toulouse.

Officiellement créé le 10 juillet 1967 sous le nom de Laboratoire d'Automatique et de ses Applications Spatiales avec un effectif de 140 personnes, le LAAS (**Laboratoire d'Analyse et d'Architecture des Systèmes**) compte aujourd'hui 450 personnes menant des recherches dans trois grandes disciplines des Sciences Pour l'Ingénieur (SPI) :

AUTOMATIQUE
INFORMATIQUE
MICROÉLECTRONIQUE

L'approche SYSTÈMES permet, par la présence au LAAS de chercheurs de cultures complémentaires utilisant des concepts issus des trois disciplines aux traditions différentes, une synergie qui facilite l'émergence de nouveaux champs de recherche interdisciplinaires et de nouvelles applications.

Les activités de recherche sont menées par onze Groupes et deux Laboratoires Communs qui reçoivent l'appui des Services Techniques, Logistiques et Administratifs ; chaque groupe conduit un domaine de recherche entrant dans la stratégie globale du laboratoire.

Le LAAS a, comme tout laboratoire du Département SPI, trois missions principales :

Avancée des Connaissances : Participation à la vie de la communauté scientifique nationale et internationale.

Transfert des Connaissances : Partenariat avec le secteur économique et les collectivités.

Formation : Par la Recherche et à la Recherche.

FIG. 1.1 – *Vue aérienne du LAAS-CNRS*

1.2 Le groupe RIA

Les recherches du groupe Robotique et Intelligence Artificielle (RIA) portent sur la problématique de la machine intelligente. Il s'agit de concevoir et de réaliser des machines en mesure de percevoir et d'agir dans un environnement variable et évolutif, capables de raisonner sur une diversité de tâches et sur les moyens nécessaires pour les accomplir de façon autonome.

Cet objectif nécessite une démarche scientifique qui mène en parallèle et couple étroitement des développements formels et des développements expérimentaux. Ainsi le groupe RIA travaille de façon concomitante:

- à l'approfondissement des principaux thèmes de la machine intelligente (perception, commande, mouvement, manipulation, et décision) au niveau des concepts, des méthodes et des algorithmes nécessaires à la robotique;
- à la réalisation de robots complets et d'expérimentations dans des environnements réels qui permettent effectivement les développements thématiques visés, ainsi que la validation des approches mises en oeuvre et la qualification de l'autonomie et du degré de rationalité des robots conçus par le groupe.

L'activité de RIA se décline thématiquement et relativement aux principaux projets internes du groupe selon le schéma suivant :

Perception :

- Acquisition, segmentation, fusion
- Modélisation de l'environnement
- Interprétation de scènes
- Perception réactive, planification sensorielle

Commande :

- Commande référencée capteurs
- Commande optimale
- Commande dynamique
- Commande coordonnée

Mouvement :

- Planification de trajectoires pour robots non holonomes
- Mouvement en terrain accidenté
- Incertitude et mouvement asservi capteur

Manipulation :

- Modélisation géométrique, cinématique et dynamique
- Téléprogrammation et manipulation autonome
- Préhension

Décision :

- Planification de tâches
- Supervision et contrôle d'exécution
- Coopération, actions coordonnées
- Architecture décisionnelle temps-réel
- Apprentissage

Les liens entre les thèmes de recherche poursuivis par RIA sont multiples : à travers leur nécessaire intégration dans des plates-formes robotiques, mais également à travers leur utilisation et leurs contributions à des bases formelles communes et à une algorithmique commune, relevant de l'automatique, de la géométrie et de la logique.

Par ailleurs, les orientations internes affichés ci-dessus indiquent clairement le choix stratégique de RIA qui privilégie, pour des raisons scientifiques et socio-économiques, la robotique non-manufacturière, dite robotique d'intervention et de service. Ceci a conduit le groupe à l'utilisation de divers robots :

- ADAM et LAMA sont des robots tout terrain qui servent de support aux recherches sur la robotique d'intervention en site non coopératif : exploration planétaire, de la Lune, ou de l'Antarctique.
- HILARE est une famille de robots mobiles, plates-formes essentielles de développement des travaux de RIA sur la robotique autonome en environnements structurés, et en particulier depuis début 1994, sur les problèmes de coopération multi-robots.

Ce choix stratégique s'accompagne de développements sur des bras manipulateurs (robot SCEMI). Un bras embarquable (Robosoft) vient d'être intégré sur la plateforme d'un HILARE (voir figure 1.2).

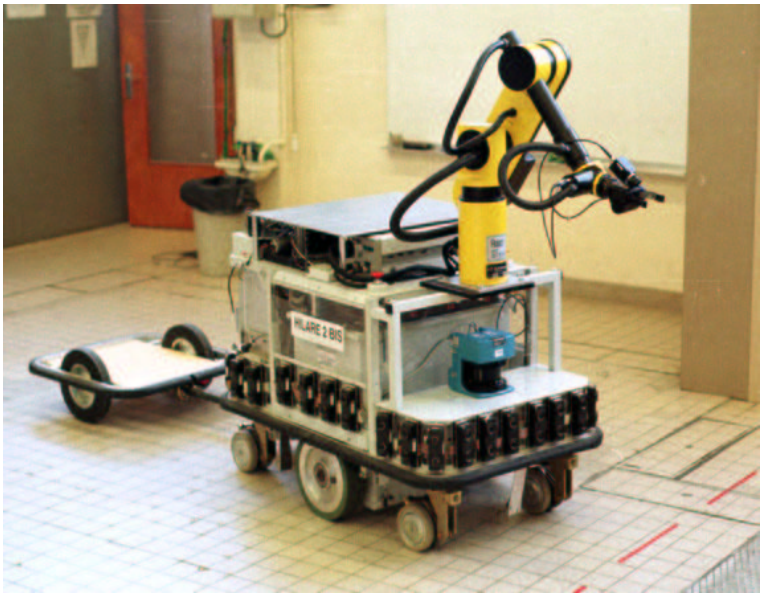

FIG. 1.2 – *Robot Hilare2bis avec bras et remorque*

Parallèlement au développement de grands projets internes, selon leur logique propre, RIA est ou a été impliqué dans plusieurs projets internationaux, parmi lesquels :

IARES : projet EUREKA de robotique d'exploration planétaire;

MARTHA : projet ESPRIT sur la coopération multi-robots pour la manutention lourde dans les ports, aéroports et gares, par des équipes de plusieurs dizaines de robots;

PROMOTION : projet ESPRIT sur la planification de mouvements en robotique;

TIGER : projet ESPRIT sur le raisonnement temporel et la décision temps-réel en supervision de processus.

Le groupe est présent dans de nombreuses autres actions; il anime plusieurs structures internationales, en particulier IARP (International Advanced Robotics Programme), IFRR (International Foundation of Robotics Research), CARRA (Centre Associatif de Recherche en Robotique Avancée).

Au niveau national RIA est actif dans diverses collaborations , par exemple:

- au sein des groupements RISP et GEROMS pour la robotique spatiale;
- au sein des groupements PRC-IA et PRC-CHM pour des problèmes de raisonnement temporel, de planification de tâches, de planification de mouvements et de
- perception réactive;
- au sein du groupement GDR Microsystèmes pour les problèmes de microrobotique;
- au sein du groupement PROMIP, pour la robotique d'intervention et de service.

D'autre part, le groupe entretient de nombreuses collaborations contractuelles avec des organismes publics et des industriels parmi lesquels : le CNES, IFREMER, Matra Marconi Space, Cybernetix, Framatome, Alcatel Espace, Thomson/Midi-Robots, Storagetek, Dassault Aviation, IXI, Laboratoire FABRE.

1.3 Introduction au sujet de stage

De plus en plus, on envisage de contrôler des robots équipés de capteurs de manière à les faire évoluer dans un environnement structuré (ex: fauteuil roulant, véhicule circulant sur autoroute etc.). La réalisation de tels systèmes de supervision pour les processus complexes pose de nombreux problèmes, notamment en ce qui concerne le maintien en temps réel d'une interprétation fine de l'état du système complet. Le système C-PRS est utilisé au LAAS-CNRS comme outil de supervision, il permet de spécifier des procédures de contrôle invoquées sur l'apparition d'événements. Ce mécanisme n'accepte pas la prise en compte de scénarios temporels complexes. IxTeT/Reco est un outil qui permet la supervision et le suivi de tels scénarios. Le travail effectué a eu pour but de combiner ces deux approches, ceci permettant d'utiliser les points forts de chaque outil dans des tâches de supervision.

Chapitre 2

Supervision : état de l'art

2.1 Introduction

Dans le cadre de ce stage, nous nous sommes intéressés à la "supervision de processus complexes"; notre approche a donc naturellement été guidée par les travaux précédemment réalisés dans le domaine. Ceux que nous avons étudiés se divisent en deux catégories. La première concerne la supervision en général et les différentes fonctionnalités pour la mettre en oeuvre. La seconde concerne quelques exemples constituant des systèmes de supervision complexe.

2.2 Approche générale

2.2.1 Définition

La supervision est, pour nous, constituée d'une fonctionnalité de surveillance couplée à une fonctionnalité de conduite exercées sur un système. La fonctionnalité de conduite a pour but de contrôler ce système et de corriger ses erreurs de fonctionnements en temps réel ; la fonctionnalité de surveillance reconnaît les situations et permet éventuellement de les corriger. Ainsi une personne conduisant sa voiture assurera la tâche de supervision de la voiture, lui ordonnant d'aller dans une direction, contrôlant les différents voyants indiquant les erreurs de fonctionnement et réparant ces erreurs quand cela s'avère nécessaire.

En ce qui concerne la surveillance, une classification des modélisations possibles, et actuellement utilisées, a été faite par M. Basseville et M.-O. Cordier [1]; nous reprenons ici cette classification.

2.2.2 Fonctionnalité de surveillance/reconnaissance

Notons que les différents modèles de surveillance que nous allons décrire ici vont souvent plus loin que la simple surveillance même si c'est leur spécialité. On peut distinguer trois types de modèles différents :

les modèles associatifs : Ceux-ci sont basés sur une identification a priori des symptômes ou liens caractérisant une situation qui permet de la reconnaître quand elle se produit. Les représentants de ce type de modèle sont :

- Les systèmes experts qui permettent le diagnostic de systèmes dynamiques.

- La reconnaissance de scénarios qui est basée sur le suivi des événements du système considéré et qui reconnaît des scénarios (ou chroniques) d'événements prédéfinis par des experts. Notons que le système IXTET/Reco (C. Dousson [4]) qui nous intéresse ici fait partie de cette catégorie.

Ces modèles ont comme points faibles la difficulté de constituer la base d'expertise nécessaire (et donc la difficulté de validation des expertises) et une évolution difficile lorsque les techniques contrôlées évoluent. Ils ont néanmoins un certain nombre de points forts comme l'efficacité dont ils font preuve lors de l'exécution.

les modèles prédictifs : Ces modèles décrivent le comportement normal du système suivant son état (par exemple l'ordre de grandeur de certaines mesures) et confrontent ce comportement standard avec le comportement actuel pour détecter les anomalies de celui-ci. On a alors un ensemble d'états stables et un ensemble de transitions entre ces états. Les représentants caractéristiques sont :

- Les modèles à base de raisonnement qualitatif.
- Les modèles à événements discrets (réseaux de Pétri temporisés...). Leur utilité se fait particulièrement sentir lors de la modélisation de synchronisation d'événements et de simulations de systèmes.
- D'autres modèles correspondent aussi à cette catégorie : réseaux bayésiens ou les réseaux probabilistes temporels...

On peut noter qu'on rencontre alors souvent des difficultés au niveau de la détection et du diagnostic au cours de la mise au point.

les modèles explicatifs : Ces modèles décrivent partiellement le système en mettant en avant les liens de dépendances. Leurs principaux représentants sont :

- Les graphes d'influence qui explicitent les liens entre les variables du procédé.
- Les graphes causaux (et temporels) qui explicitent les liens entre les états du système et leurs effets.

Les premiers rencontrent des difficultés lorsqu'il existe des boucles dans les équations numériques qui permettent de définir les relations d'influence entre les grandeurs physiques; les seconds sont presque exclusivement utilisés pour des tâches de diagnostic du fait de leur structure causale (telle que "A cause B" et "A peut causer B").

Tous ces modèles peuvent être utilisés pour assurer la fonctionnalité de surveillance. Voyons maintenant de quelles sont les possibilités qui sont disponibles pour la tâche de conduite.

2.2.3 Fonctionnalité de conduite

Nous allons ici prendre en compte le fait que notre outil de supervision doit être capable d'effectuer des actions complexes. En effet, lors de la correction de comportement, il doit être capable de répondre efficacement, par des méthodes parfois complexes, aux erreurs ou aux situations reconnues par la fonctionnalité de surveillance.

Dans la fonctionnalité de conduite, il faut donc considérer l'aspect procédural des problèmes que nous allons traiter. Il constitue l'aspect actant de la conduite. Dans cette optique, on peut considérer qu'une connaissnace de l'état du monde, des possibilités de parallélisme et l'existence d'objectifs constituent un plus. En effet, nous cherchons à avoir des comportements réflexes dans certaines

situations qui ne nécessitent pas l'intervention de la tâche de surveillance. Un exemple de tels comportements à adopter peut être que lorsqu'un signal est reçu on peut avoir à effectuer toujours une même suite d'actions (quand un capteur indique qu'une voiture arrive devant une barrière automatique, celle-ci doit se lever par exemple). Il devient donc intéressant d'avoir des langages comme RAPS [6] ou PRS [8]. En effet, ces langages intègrent déjà une notion minimale de temps et peuvent réagir de manière réactive à des événements simples. La fonctionnalité de surveillance assure alors uniquement la reconnaissance des situations complexes, chose pour laquelle les outils qui découlent des modèles précédemment cités sont particulièrement adaptés.

2.2.4 Fonctionnalité temps-réel

Le temps réel est devenu une préoccupation importante de l'intelligence artificielle [16] dans le sens où, de plus en plus, on cherche à utiliser des systèmes capables de s'adapter au monde réel et d'interagir avec lui. Les exemples d'application de tels systèmes sont nombreux en supervision. Des travaux sont donc en cours dans ce domaine, ceux-ci débouchent sur des techniques (comme, par exemple, les algorithmes *anytime* [3]), sur des systèmes ayant des temps de réaction suffisamment rapides pour pouvoir réagir en temps réel (comme les systèmes réactifs du type PRS [8, 17] ou d'autres langages gérant formellement ces aspects [10]) ou encore sur des langages appelés "langages temps-réel synchrones" comme Lustre ou Estérel [2].

L'approche synchrone, en fait, considère qu'un automate théorique qui change d'état à une fréquence donnée produit, théoriquement, ses sorties de manière simultanée avec ses entrées. L'approche synchrone [2] ne paraît pas pouvoir convenir dans le cas présent car les avantages qu'elle propose ne permettent pas de compenser ses défauts. En effet, elle garantit que l'on fait les changements d'états de l'automate principal dans un temps borné mais elle n'assure pas un temps borné pour les calculs ou les procédures complexes car le temps de calcul nécessaire à de telles actions est souvent supérieur à la fréquence et difficilement calculable dans le cas qui nous intéresse.

Le fait que nos outils doivent pouvoir réagir en temps réel, c'est-à-dire de manière performante, est le facteur principal qui va décider quelle approche doit être retenue pour la supervision. Comme nous l'avons écrit dans la section précédente, les systèmes de surveillance les plus performants sont ceux basés sur des modèles associatifs (i.e. soit des systèmes experts, soit des systèmes de reconnaissance de scénarios). Au niveau de la conduite il nous paraît plus prometteur de considérer plutôt des langages intégrant déjà une gestion du temps et capables d'assurer des raisonnements procéduraux que des langages plus procéduraux (comme le C) auxquels on devrait adjoindre des fonctionnalités temporelles.

2.3 Quelques exemples de superviseurs

2.3.1 Couples système réactif/système déductif existants

Cette section introduit rapidement trois systèmes qui sont constitués d'un système réactif et d'un système déductif qui coopèrent.Notons toutefois que, bien que l'on puisse penser au premier abord que notre travail s'inscrit dans ce cadre, ce n'est pas le cas. En effet tous ces systèmes ont comme système déductif un planificateur [13]: ils ont plutôt pour but de réaliser de la planification réactive que de la supervision à proprement parler.

Le système CIRCA

Cooperative Intelligent Real-time Control Architecture (CIRCA) [14, 15] a pour principe de planifier à propos de tâches qui doivent se dérouler en temps-réel plutôt que de planifier en temps réel pour répondre à des événements. En fait, le planificateur a une compétence sur les scénarios

s'effectuant en temps-réels qu'il va proposer au sous-système temps-réel. On a alors une séparation complète entre la composante prédictive (planificateur) et la composante temps-réel.

Le système NMRA

New Millennium Remote Agent (NMRA) [18, 19, 21] est un projet de la NASA qui vise à contrôler un engin spatial au moyen d'un système à base d'intelligence artificielle. Dans ce but, les concepteurs du projet ont décidé de distinguer deux parties : une partie d'exécution (Exec) et une partie déductive appelée Mode Identification and Reconfiguration system (MIR). La partie MIR est, en fait, un planificateur qui reçoit les données extérieures au moyen de capteurs. La partie Exec agit sur le monde extérieur. Entre ces deux parties va s'instaurer un dialogue lorsque MIR aura reconnu un problème. MIR envoie alors un message à Exec pour lui signifier qu'il a reconnu un problème. Exec fait une demande de plan. MIR planifie et renvoie alors la première action du plan à exécuter. Tant que la situation ne sera pas rétablie les demandes de plan continueront ainsi que les envois d'actions.

Ce système est très intéressant; néanmoins, il nous paraît important que le système que nous utiliserons puisse réagir aux actions standard sans que le planificateur ait eu à finir sa reconnaissance (certaines actions sont à effectuer immédiatement).

Le système $\mathcal{RS}$

Reasonning System ($\mathcal{RS}$) [11, 12] est un système ressemblant beaucoup à ce que nous proposons. Il repose sur un système réactif appelé RAPS [6] et sur un système déductif se basant sur un planificateur du type STRIPS [13]. Son architecture ressemble à la nôtre (voir figure 2.1), elle semble néanmoins plus lourde du fait qu'elle comprends un planificateur qui n'est pas spécialisé dans la reconnaissance de scénarios mais qui est plutôt généraliste.

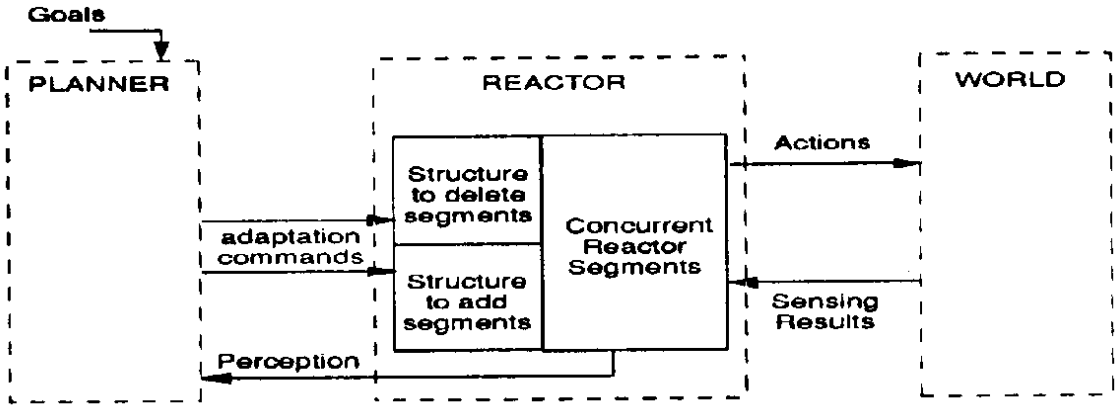

FIG. 2.1 – *Architecture générale du système $\mathcal{RS}$ (source [12]).*

Le formalisme est très intéressant même si on peut regretter sa difficulté apparente d'utilisation.

Lacunes de ces systèmes par rapport à nos attentes

Ces systèmes ont des lacunes dans la gestion du temps et donc dans les aspects temps-réel que nous avons évoqués précédemment : ils constituent plutôt des systèmes de planification réactive

que des systèmes de supervision. Ils n'assurent ainsi, en général, que des tâches de surveillance minimales.

2.3.2 Système utilisant le raisonnement à partir de cas : DEPUR

Le système DEPUR [20] que nous allons rapidement détailler ici a pour but de superviser une centrale de traitement des eaux usées. Cet exemple nous intéresse car c'est à ce type de problème que nous désirons pouvoir appliquer notre approche.

DEPUR est un système dédié. Il se constitue de plusieurs modules (agents) qui ont chacun une compétence sur la marche à suivre dans un cas particulier. Les principaux problèmes proviennent du fait que les systèmes controlés n'ont pas de description générale trés précise ce qui oblige le superviseur à être dynamique et à ne pas pouvoir prévoir avec certitude ce que ses actions vont entraîner. Le superviseur est en fait constitué de différents agents ayant des compétences complémentaires. L'agent central collecte les informations et décide quelles actions vont être confiées à quels agents actants.

Ce système est très intéressant du point de vue de la conception. En effet, il combine un certain nombre de propriétés que nous avons dégagées pour caractériser ce qui nous semblait être un bon superviseur. On peut néanmoins regretter que l'architecture présentée ne semble pas facilement réutilisable pour un autre problème de supervision. De fait, l'outil de supervision a été créé autour du problème de la centrale de traitement et non pas dans l'idée de faire un outil général.

2.4 Conclusion

Notre approche apparaît comme étant une tentative pour créer un système de supervision général adapté à des problèmes industriels complexes. Les systèmes que nous avons détaillés précédemment ne paraissent pas pouvoir répondre à nos besoins. Rappelons que ces derniers sont orientés dans trois directions différentes nous paraissant être les composantes essentielles de la supervision : la surveillance, la conduite et le temps-réel. Nous essayerons par la suite de répondre à ces objectifs fondamentaux dans notre approche. Les choix que nous avons faits se situent pleinement dans les conditions évoquées précédemment. En effet notre problème étant de concevoir un outil de supervision, nous avons choisi d'adopter un modèle de reconnaissance de scénarios (IxTeT/Reco) pour effectuer la surveillance et de le coupler à un système réactif (C-PRS) pour effectuer la conduite. Notre démarche sera détaillée au Chapitre 4.

Chapitre 3

Présentation des outils utilisés

Les systèmes que nous avons utilisés et modifiés pour concrétiser notre démarche sont des outils qui étaient déjà présents dans le groupe de Robotique et d'Intelligence Artificielle(RIA). Le premier se nomme C-PRS, c'est un système d'exécution de procédures. Le deuxième, LaTeT/Reco, est un système de reconnaissance de scénarios.

3.1 Présentation de C-PRS

3.1.1 Introduction

C-PRS est un langage réactif de raisonnement procédural en domaine dynamique. La version dont nous disposons est distribuée par ACS Technologies[1]. Ce langage est utilisé dans un certain nombre de projets du groupe RIA tels que MARTHA (projet de coopération multi-robots) pour le contrôle des robots mobiles. Il a été défini par Georgeff et Ingrand [8] qui ont développé pour la NASA une application de supervision du RCS (Reactive Control System) de la navette spatiale [5, 7].

3.1.2 Présentation du système

Représentation/Architecture

Un module PRS est constituée des éléments suivants :

1. une base de données contenant les faits représentant l'état du système à superviser.
2. un ensemble de buts ou d'objectifs actuels.
3. une bibliothèque de plans (ou procédures) qui décrivent les actions à effectuer pour atteindre certains buts ou les réactions à adopter pour réagir à certaines situations.
4. une structure d'intentions qui est un ensemble ordonné de tous les plans choisis pour l'exécution.

1. ACS Technologies (INGENIA Group), 5, Place du Village d'Entreprises,BP 556, 31674 LABEGE Cedex, FRANCE

Un interpréteur utilise tous ces composants en choisissant les plans, en les plaçant dans la structure d'intentions et finalement en les exécutant.

PRS intéragit avec son environnement par l'intermédiaire de la base de données. Il peut y ajouter des faits représentant l'évaluation du système. On peut remarquer que les mises à jour de la base de données font appel à des techniques de maintien de la consistance.

En PRS les buts sont des descriptions des tâches désirées et des comportements. Dans la logique PRS, le but pour atteindre une certaine condition C s'écrit *(! C)*; tester une condition s'écrit *(? C)*; attendre jusqu'à ce qu'une condition soit vraie s'écrit *($\wedge$C)*; maintenir une condition s'écrit *(# C)*; conclure une condition s'écrit *(=> C)* et retirer une condition s'écrit *($\rightsquigarrow$ C)*.

Les connaissances sur la façon d'accomplir un but sont stockées dans les procédures (ou plans). Ces derniers consistent en une partie d'appel (*INVOCATION*) qui spécifie dans quelles circonstances la procédure est utile et un corps (*BODY*).

Le corps d'une procédure est représenté soit par un graphe orienté soit par un programme textuel. On peut noter qu'une procédure a un autre champs appelé *CONTEXT* qui est une condition d'exécutabilité de celle-ci suivant l'état du monde. Voir l'exemple 3.1 tiré de l'Annexe B.

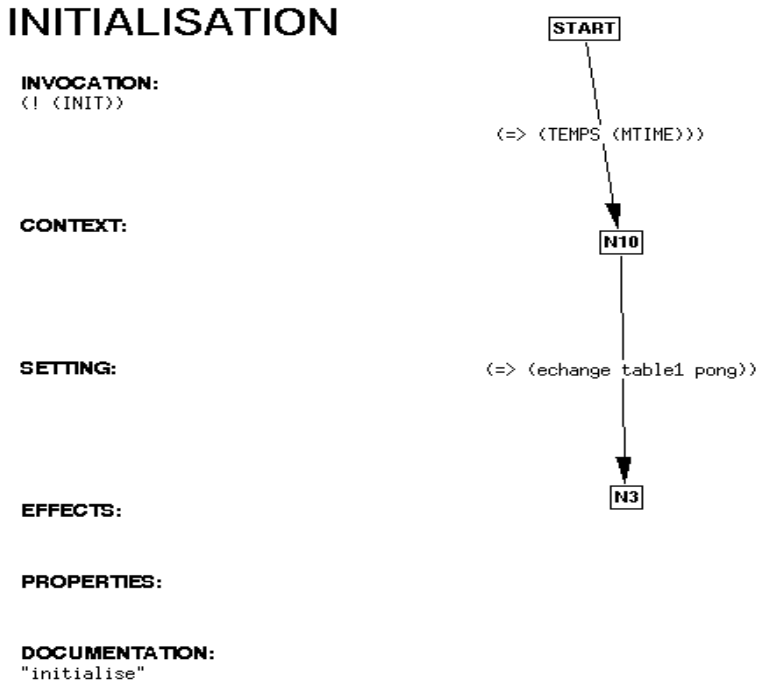

FIG. 3.1 – *Procédure d'initialisation* INIT.

Le corps d'une procédure indique ce qu'il faut faire en cas d'appel. Le point d'entrée est le noeud *START* et l'exécution suit les arcs les uns après les autres. L'exécution réussit lorsqu'on arrive à un noeud fin (noeud sans arc sortant). Lorsque plusieurs arcs partent d'un même noeud l'interpréteur choisit au hasard parmi les buts associés applicables. Lorsqu'un arc porte un but à atteindre, cela signifie que l'on va vérifier si le but n'est pas déjà réalisé ou s'il existe un moyen de le satisfaire par une procédure qui sera appelée. Une possibilité est offerte par le noyau PRS pour réaliser des structures concurrentes. De plus il existe un méta-niveau de procédure qui permet typiquement de choisir entre des procédures qui conduisent à un même but. Autre fonctionnalité, plusieurs agents PRS peuvent être utilisés et communiquer entre eux.

Fonctionnement interne

Le langage est interprété et l'interpréteur suit un cycle particulier que nous allons décrire ici (voir figure 3.2). À un certain moment, un but arrive ou il y a une altération de la base de données(1). Ceci appelle alors une ou plusieurs procédures (2). Un certain nombre de procédures sont applicables : on en choisit une et on la place dans les intentions courantes (3). Finalement PRS sélectionne une intention (4) et exécute une étape de cette intention (5). De ceci résulte l'accomplissement d'une action primitive (6) ou l'établissement d'un nouveau sous-but ou la conclusion d'un nouveau fait (7). Le cycle recommence alors.

On peut noter que les appels de procédures s'empilent et forment des sous-buts qui réagissent comme les buts de niveaux supérieur.

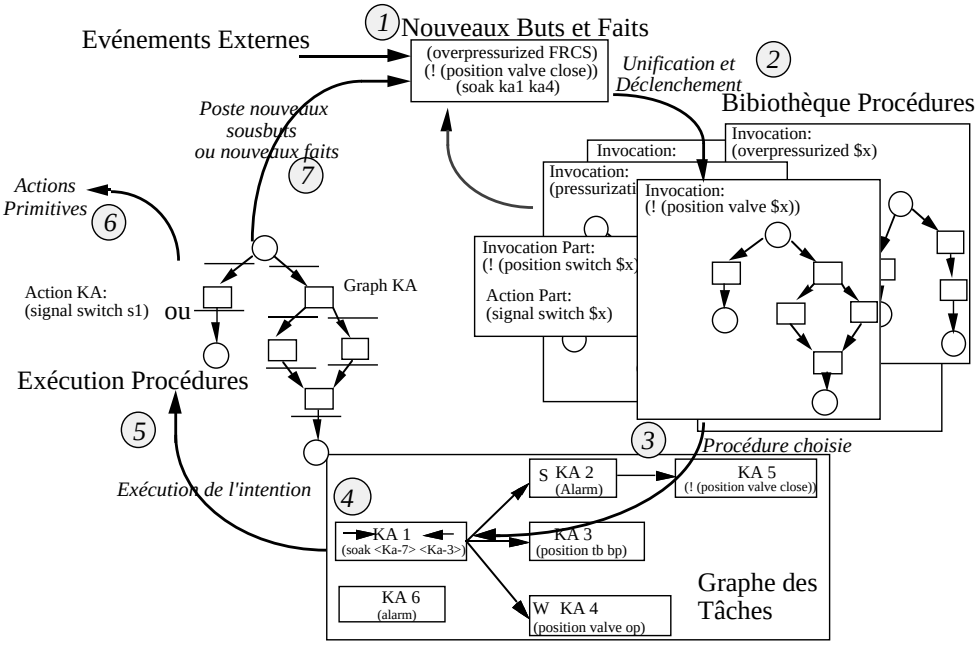

FIG. 3.2 – *Boucle principale de C-PRS.*

3.1.3 Spécificités

Détaillons les spécificités de ce système.

- Il a la particularité de construire et d'agir suivant des plans partiels.
- Il a la possibilité de répondre à un changement d'environnement tout en poursuivant une tâche orientée vers un but.
- Il intègre aussi la capacité de s'occuper de multiples tâches en temps-réel.
- Il possède des mécanismes qui lui permettent de répondre à des demandes en temps-réel.
- Il a ausi présent un méta-niveau de raisonnement qui permet de prendre des décisions sur les procédures à utiliser.

De plus il répond aux impératifs de conduite que nous avons explicités précédemment du fait de la souplesse tant au niveau de la manipulation des variables que de la syntaxe et des comportements temporels implantés.

3.2 Présentation d'IXTET/Reco

3.2.1 Introduction

Introduction

Le système IXTET/Reco (développée par C. Dousson [4]) est un langage qui utilise le formalisme temporel IXTET [9]. Il a été développé dans le groupe de recherche RIA du LAAS-CNRS. Ce langage sert exclusivement à faire de la reconnaissance de scénarios, ou, plus précisément, des "chroniques temporelles".

3.2.2 Présentation du système

Comme nous l'avons écrit au Chapitre 2, IXTET/Reco est un outil dédié à la reconnaissance de situations. Il accepte en entrée un flot d'événements datés et il effectue un traitement des événements qui lui permet de reconnaître les scénarios temporels précédemment décrits. C'est un langage compilé qui offre une interface à l'utilisateur par l'intermédiaire du clavier.

Les chroniques (*chronicle...*) effectuent leurs traitements en parallèle. Elles sont constituées de trois types d'informations utiles à la reconnaissance (voir figure 3.3):

les événements (*event...*) qui sont pris en compte sont en fait des changements de valeurs datés que subissent des variables globales appelées "attributs"(*attribute...*).

les assertions (*hold...*)qui sont des conditions correspondant à la persistance de valeurs d'attributs sur un intervalle de temps

les contraintes temporelles qui vont ordonner les événements et donner des bornes aux assertions.

Lorsqu'un événement se produit chaque chronique concernée par cet événement est mise à jour.

La phase de compilation constitue une étape essentielle de la démarche. En effet, la compilation permet de vérifier la cohérence des chroniques et de les compiler en une structure de données efficace pour le traitement en ligne.

Lors de l'exécution, le flot d'événements datés arrive jusqu'au noyau IXTET/Reco qui va mettre à jour, événement aprés événement, les reconnaissances engagées. Au fur et à mesure qu'il reçoit des événements, le système va développer des hypothèses regroupées sous forme d'un *arbre d'hypotehèses*. Une tentative de reconnaissance de chronique va consister en la construction d'un tel arbre (voir figure 3.4).

Chaque noeud de l'arbre correspond à une portion de chronique reconnue. Lorsqu'un événement arrive, il y a duplication des noeuds où il peut être considéré comme valide pour la reconnaissance en cours. Les noeuds ainsi dupliqués vont ensuite intégrer l'événement en question et être rattachés (comme fils) aux noeuds ne l'ayant pas intégré. Ce système permet d'obtenir une réactivité importante et l'intégration de l'événement se fait en un temps minimal (voir C.Dousson [4]).

La duplication que nous avons constatée est importante pour pouvoir prendre en compte tous les scénarios possibles. Si elle n'existait pas, on ne pourrait garantir la complétude (c'est-à-dire la

```
constant ETAT_ECHANGE= {ping, pong};
// etat de l'echange
TABLE_POS= {table1};
attribute echange(?ta) {
?value in ETAT_ECHANGE;
?ta in TABLE_POS;
}
chronicle ping ()(t_debut) {
event (echange(table1): (pong,ping),t_debut);
when recognized {
call send_pong();
report "ping";
}
}
```

FIG. 3.3 – *Chronique servant au calcul du temps de parcours*

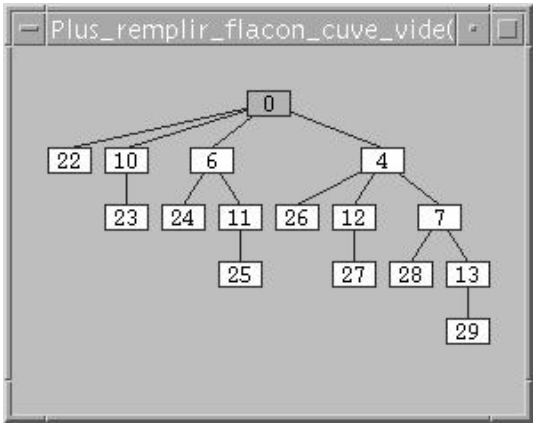

FIG. 3.4 – *Exemple d'arbre de reconnaissance IXTET/Reco.*

reconnaissance de n'importe quelle suite d'événements qui correspond à la chronique construite). En effet, on pourrait avoir des problèmes, par exemple, dans le cas où les contraintes temporelles ne sont pas très explicites.

Comme nous avons pu le voir, ce langage est conçu pour faire de la reconnaissance. La reconnaissance peut être rapportée à l'utilisateur par deux moyens : un rapport émis à l'écran (*report...*) ou par l'utilisation de classes C++ (*call...*) à écrire séparément qui peuvent prendre en argument des variables de la chronique reconnue.

3.2.3 Spécificités

IXTET/Reco est un langage qui est spécialisé dans la reconnaissance de chroniques. À ce titre, il intègre des mécanismes de gestion d'arbres optimisés pour le traitement des données qui lui sont communiquées par l'extérieur. En effet, il a alors une complexité en taille de données importante qui est contrebalancée par une complexité en temps amoindrie [4]. Cette particularité est primordiale pour nous comme nous avons pu le voir au Chapitre 2.

Chapitre 4

Démarche adoptée, travail effectué

4.1 Justification de notre démarche

4.1.1 Introduction

Comme nous l'avons déjà écrit, notre approche consiste en une convergence des démarches de C-PRS et LTT/Reco pour bénéficier des avantages inhérents aux deux points de vue : celui du système réactif et celui du système de reconnaissance temporelle. L'utilité d'une telle approche n'est néanmoins pas évidente : chaque langage peut, à lui seul, réaliser des tâches complexes de supervision. Comme le montre la brève étude suivante, ces deux langages sont complémentaires.

4.1.2 C-PRS comme outil de surveillance

De nombreuses applications utilisent C-PRS comme superviseur, une des plus représentatives d'entre elles, est RCS qui gère des vannes de réacteurs de navette spatiale [7].
Pour pouvoir ajouter la fonctionnalité de surveillance de situation à C-PRS on peut principalement dégager trois méthodes. La première méthode consiste en fait à modéliser les chroniques (ou situations à reconnaître) dans le contexte des KAs à déclencher. La deuxième méthode est de construire des KAs qui vont suivre l'évolution de ce qui correspond aux chroniques. La troisième solution consisterait en l'implantation de mécanismes d'arbres de déclenchement ; cette dernière solution n'est néanmoins pas intéressante dans la mesure où elle reviendrait à re-programmer les mécanismes d'LTT/Reco.
Quelles que soient les solutions, il faudrait néanmoins temporiser la base de données (au moins les faits servant à la reconnaissance).

Modélisation des chroniques dans le contexte des KAs

Le contexte de telles KAs va se présenter de la manière suivante:
`(& (POS1 $V1 $T1) (POS2 $V2 $T2) ... (< 10.00 (- $T1 $T2))...` où on va constater le passage de valeurs du prédicat POS1 à V1 au temps T1 ... on va ensuite tester les contraintes de temps et on va attendre que toutes les conditions soient vérifiées.
Les problèmes que pose cette approche sont importants. En effet, du fait de l'unification, on calcule toutes les instanciations. On obtient alors une complexité exponentielle (voir tableau 4.1). Ce qui nous donne pour des chroniques simples avec cinq unifications possibles (k=5) pour chaque

événement, quatre évènements à vérifier (n=4), trois contraintes de temps et deux propriétés de persistance à vérifier (p=5) on a alors un arbre de recherche avec 3125 feuilles! De plus, ce calcul est à répéter à chaque nouvel événement concernant la conjonction car C-PRS ne garde pas en mémoire les calculs précédents. Ainsi pour qu'une simple chronique soit vérifiée on obtient une complexité très importante (12500 feuilles calculées pour l'exemple pris). Le détail des calculs de complexité est donnés en annexe A.

D'une manière générale, pour :

n nombre d'événements.

k nombre maximum d'unifications possibles pour chaque événement.

p nombre de contraintes de temps et de contraintes de persistance dans le temps, on obtient les complexités suivantes.

Complexité au pire cas pour une évaluation	$\mathcal{O}(k^n p)$
Complexité moyenne pour une chronique reconnue	$\mathcal{O}(nk^n p)$

TAB. 4.1 – *Calcul de complexité dans le cas d'une intégration dans le contexte.*

Cette approche ne paraît donc pas être une bonne solution, la complexité de la vérification de formules pour des chroniques très simples étant déjà importante.

Modélisation des chroniques par des KAs

La réalisation de reconnaissance de chroniques sous forme de KAs résout les problèmes de complexité mais elle en pose de nouveaux. En effet, l'idée principale serait de réaliser une KA qui reconnaît la chronique considérée de par sa structure ordonnée, son parallélisme et ses contraintes temporelles (voir figure 4.1).

Cette approche a un inconvénient fondamental : il ne peut y avoir de duplication comme dans IXTET/Reco à moins de faire à chaque noeud des boucles sur lui-même en parallèle avec la reconnaissance (ce qui nous ramène au problème précédent en ce qui concerne la complexité). On peut aussi mettre en place un filtre sur les évènements qui doivent être considérés... Néanmoins cela apparaît d'une complexité de mise en oeuvre excessive pour des comportements aisément implémentable sous IXTET/Reco.

Conclusion

Ainsi il apparaît que la construction d'un superviseur ayant besoin d'une forte composante au niveau de la reconnaissance temporelle qui ne serait implémenté qu'avec C-PRS serait dommageable à la complexité des algorithmes.

4.1.3 IXTET/Reco comme système de conduite

IXTET/Reco est utilisé comme outil de supervision ayant pour but de faire de la surveillance de processus complexes. Le projet TIGER (voir thèse de C. Dousson [4]) en est un exemple significatif. Ce projet consiste en la gestion de turbines et la reconnaissance des pannes de la séquence de démarrage. Ces processus complexes sont alors surveillés mais l'accent a été mis sur la reconnaissance de situation plus que sur la mise en oeuvre d'actions correctrices. Examinons les principaux problèmes qui gênent cette utilisation.

```
exemple de chronique :

chronicle possibilite_emprunt (?Client)(t_debut) {
timepoint t_avant;
event (en_train_d_emprunter(?Client): (non,oui),t_debut);
hold (nb_cautions(?Client): un, (t_avant,t_debut));
t_avant <= t_debut;
when recognized {
report "section reconnue"}
}
```

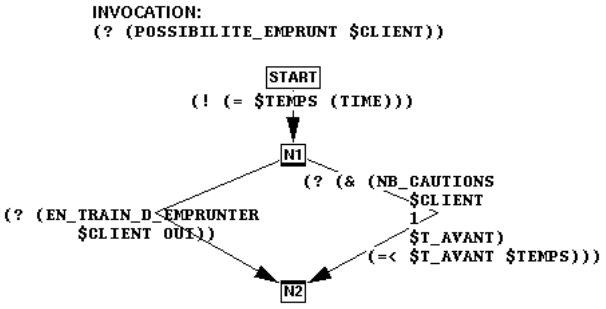

FIG. 4.1 – *Chronique et sa transformée en KA C-PRS*

Problèmes concernant l'implémentation de la fonctionnalité de conduite

Le principal problème que l'on peut dégager est que lorsque l'on veut agir sur le système que l'on surveille, on est toujours confronté au fait que l'on ne peut que difficilement prendre en compte les chroniques qui ont été reconnues précédemment ou faire des choix conditionnels. En effet, la seule manière d'implanter un choix dans une chronique est d'en créer une seconde qui contient l'autre alternative du choix que l'on prend en compte. D'autre part, dans la version sur laquelle nous avons travaillé, on ne peut pas manipuler les variables, on ne peut pas avoir de variables qui aient un domaine qui ne soit pas explicité (comme les entiers par exemple). Tout ceci nous conduit à penser que faire de la conduite avec IXTET/Reco est difficile du fait que les structures de programme n'existent pas. On pourrait, bien sûr, implanter des structures permettant d'avoir une base de données ou la gestion des variables et des tests mais cela nuirait aux caractéristiques qui font d'IXTET/Reco un bon outil de surveillance comme, par exemple, sa rapidité. De plus l'aspect procédural n'est pas réellement présent dans le formalisme IXTET d'où d'importantes lacunes de ce côté-là.

Complexité des algorithmes

Le calcul de complexité des algorithmes est un peu différent de celui que nous avons effectué pour C-PRS : il faut distinguer complexité en temps et complexité en espace. On obtient des complexités importantes en espace et peu importantes en temps. Nous ne produisons pas ces résultats ici car ils ne sont pas vraiment comparable au calcul effectué précédemment. En effet, du fait des algorithmes, autant la complexité en espace est comparable à celle calculée précédemment, autant la complexité en temps ne tient pas compte du fait que l'on doit créer un certain nombre de noeuds à chaque itération du parcours d'arbre. Ainsi elle n'a rien à voir avec la complexité précédente. On peut néanmoins comprendre facilement l'avantage que l'on peut avoir à utiliser IXTET/Reco plutôt que C-PRS pour faire de la reconnaissance : c'est un outil qui fait appel à des

algorithmes qui permettent d'eviter les overheads qu'on avait pu remarquer avec C-PRS grâce à la structure d'arbres de déclenchement évoquée au Chapitre 3.

4.1.4 Conclusion

IxTeT/Reco et C-PRS sont donc naturellement complémentaires : le premier et fait pour remplir la fonctionnalité de surveillance et le second celle de conduite. On peut alors espérer tirer le meilleur parti de chacune des approches à condition de minimiser l'overhead engendré par la combinaison des deux. Les rôles ainsi dévolus aux deux langages correspondront à leurs capacités respectives : la reconnaissance de chroniques temporelles pour IxTeT/Reco et la prise en charge des comportements réactifs pour C-PRS. Voyons maintenant en quoi consiste le travail effectué et en quoi cela correspond à ce que nous en attendions.

4.2 Travail réalisé

4.2.1 Passerelle entre IxTeT/Reco et C-PRS

Nous avons réuni ces deux approches en réalisant une passerelle entre C-PRS et IxTeT/Reco. Nous avons choisi d'utiliser un serveur qui est défini en standard avec C-PRS : le Message-Passer(MP). Ce serveur est habituellement utilisé par les noyaux C-PRS pour pouvoir communiquer entre eux : il consiste en fait en un simple aiguilleur qui reçoit et redistribue les messages. Nous avons donc connecté le noyau IxTeT/Reco sur ce serveur et implémenté un moyen de communiquer en créant un serveur similaire pour IxTeT/Reco (*IxR-server*). Les données provenant du monde extérieur viennent alors jusqu'à IxTeT/Reco par l'intermédiaire de C-PRS (voir figure 4.2). Des interface pour les utilisateurs subsistent néanmoins grâce aux entrées claviers d'IxTeT/Reco (*IxR-Kb_interface*). On a aussi fournit la possibilité pour IxTeT/Reco de connaitre le temps avec un programme qui peut lui envoyer des tops d'horloge (*IxR-temporisation*) voir figure 4.3.

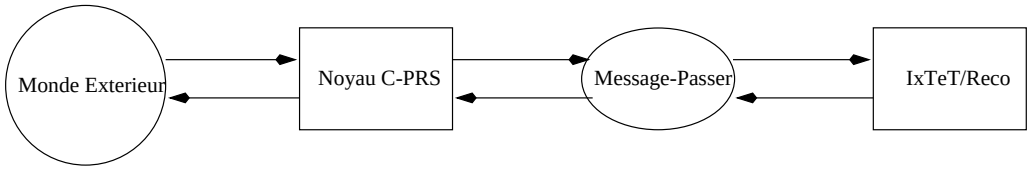

FIG. 4.2 – *Architecture générale de l'application produite*

4.2.2 De C-PRS vers IxTeT/Reco

Notre démarche a été aidée par l'existence de faits particuliers de la base de donnée se dénommant "Functionnal Facts"(ff). Ces faits doivent être déclarés au préalable comme tels et ont la particularité de ne pouvoir être présents dans la base de données à un instant donné qu'avec une seule valeur pour un certain nombre d'arguments.

exemple :
pour un "Functional Fact" `position_dans_la_piece` déclaré à
un seul argument (par exemple un robot Hillare1) présent dans la
base sous la forme
`(position_dans_la_piece Hillare1 3)`
si l'on conclut dans la base le fait
`(position_dans_la_piece  Hillare1 4)`
on n'aura alors plus le fait précédent.
Cette notation correspond à la notion de fonction :
"la fonction `(position_dans_la_piece)`
a la valeur `(4)` pour `(Hillare1)`".

En fait, cette propriété des "Functionnal Facts" correspond parfaitement à la notion d'attributs de domaines qui est utilisée dans IXTET/Reco. Nous avons donc choisi de les utiliser et de transmettre à IXTET/Reco les changements de valeur de ces faits. Cette fonctionnalité est ajoutable à cause du traitement spécial déjà nécessité par ces faits particuliers. Pour permettre à IXTET/Reco de gérer ces faits dans le temps, nous envoyons le temps courant au même instant. Notons que nous n'utilisons pas *IxR-temporisation* pour ordonner temporellement nos évènements pour des raisons que nous expliciterons par la suite.

4.2.3 D'IXTET/Reco vers C-PRS

La réception des données se fait par un procédé similaire à celui déjà présent dans IXTET/Reco pour la prise en compte des entrées au clavier. Nous avons créé une classe d'objets dédiée à la communication vers le Message-Passer et plus généralement à la communication vers l'extérieur. À l'exécution, nous créons un processus qui va gérer la communication par l'intermédiaire de cette classe. la méthode permettant la réception des messages effectue une traduction du message avant de le communiquer au noyau de reconnaissance. Parallèlement à ces deux processus s'exécute un troisième qui effectue une lecture au clavier et une traduction similaire lors d'entrées (voir figure 4.3).

4.2.4 Étude quantitative de la connexion

Un des problèmes que nous avons à prendre en compte est le temps de parcours de tous ces processus lourds. En effet, lors d'un aller-retour de message, celui-ci passe par quatre processus différents (voir figure 4.4). Ceci implique que le système doit commuter six fois entre les différents processus. Pour vérifier que la réactivité de notre système n'est pas trop altérée, nous avons donc fait quelques mesures de temps de parcours lors d'envoi de message en une espèce d'*aller-retour*. La seule chronique IXTET/Reco alors utilisée ne reconnaît que le passage d'un attribut de ping à pong... (les résultats sont présentés dans le tableau 4.2). La chronique et les KAs utilisées sont fournies en annexe B.

On voit apparaître ici la raison qui a fait que nous n'avons pas utilisé *IxR-temporisation* et que nous avons préféré envoyer le temps courant du noyau C-PRS avec les événements significatifs. En effet, nous ne pouvons garantir que deux événements ne seront pas considérés comme simultanés par le noyau Reco du fait de la lenteur de la connexion si on impose un rythme qui ne demande

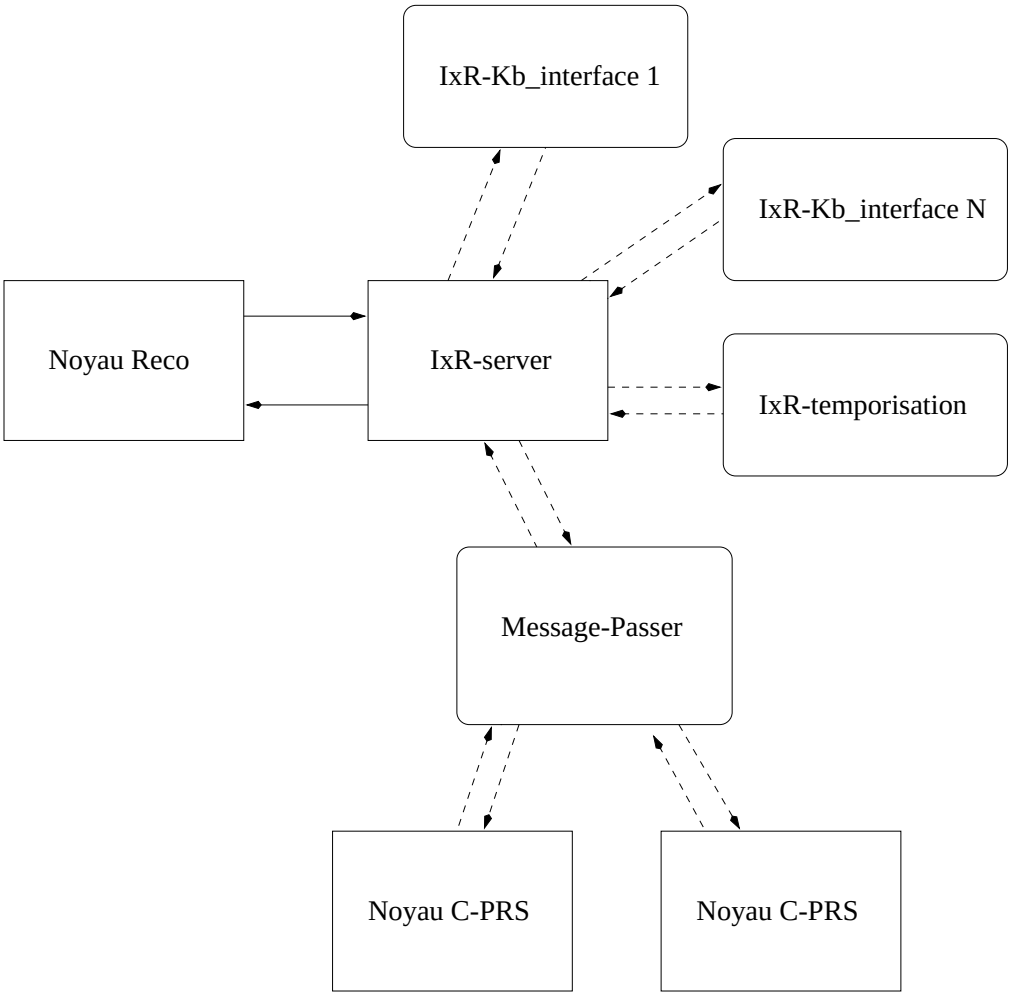

FIG. 4.3 – *Architecture de la partie I_XT_ET/Reco*

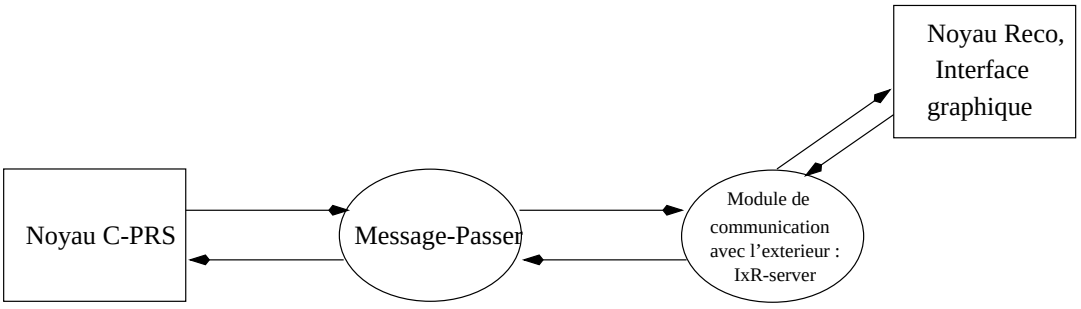

FIG. 4.4 – *Processus mis en jeu lors d'un aller-retour de message*

pas beaucoup de temps au noyau Reco. Si, en revanche, on impose des tops plus rapprochés on peut assister à un remplissage des buffers, le noyau Reco ne traitant plus assez vite les requêtes.

Machines	Temps moyen de parcours
SUN Sparc 5	182
SUN Sparc 20	144
Ultra Sparc 1	130
Ultra Sparc 5	94
Ultra Sparc 10	92

Calculs effectués sur des échantillons de plus d'une centaine d'aller-retours.

TAB. 4.2 – *Temps moyen mis pour faire un aller-retour (en ms).*

Ces résultats sont plutôt encourageants vu le temps de commutation (de l'ordre de quelques dizaines de millisecondes) entre deux processus sur les plateformes considérées. Notre approche est donc applicable sur un cas concret, en tenant compte de ces valeurs au niveau des temps de réaction désirés du système. Nous avons d'ailleurs développé un exemple fictif d'application nous permettant de tester plus avant la réactivité du système ainsi que sa souplesse.

4.3 Exemple d'application : supervision d'une chaîne de remplissage

4.3.1 Introduction

Pour tester notre approche qui ne constitue que le premier pas vers une intégration plus avancée des comportements d'IXIET/Reco et de C-PRS, nous nous sommes intéressés à la création d'une application utilisant notre approche. Le problème abordé n'est pas un problème industriel réel mais il pourrait servir de maquette à la gestion de tels procédés.

4.3.2 Positionnement du problème

Le problème consiste en la gestion d'une chaîne de remplissage de flacons avec des produits présents dans des cuves situées tout au long de la chaîne (voir figure 4.5). Les flacons défilent sous les différentes cuves emportés par un tapis roulant. Au bout du tapis un mécanisme permet d'éjecter les flacons indésirables.

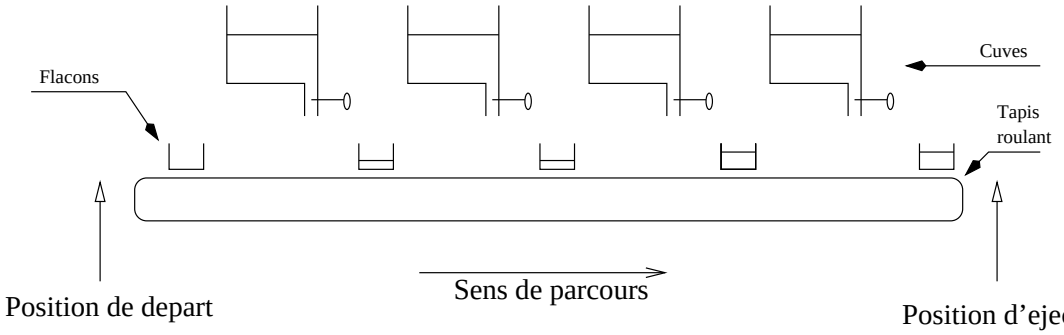

FIG. 4.5 – *Schéma représentant la chaîne de remplissage. Exemple avec 4 cuves.*

À chaque emplacement de cuve sont associés trois capteurs de présence ainsi qu'une vanne et une cuve (voir figure 4.6). Le capteur de type 1 est un capteur qui indique si la vanne est ouverte ou fermée. Le capteur de type 2 indique s'il y a du liquide qui coule ou non. Le capteur de type 3 indique s'il y a un flacon présent ou non sous la cuve. La cuve peut être emplie ou vide et la vanne peut-être ouverte ou fermée. Les capteurs peuvent fonctionner ou non.

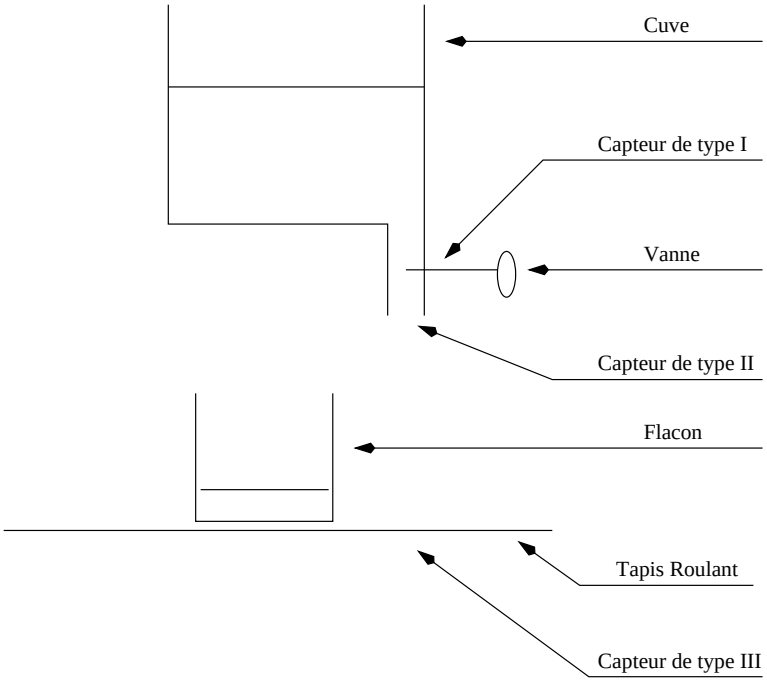

FIG. 4.6 – *Schéma représentant une cuve et tout son appareillage*

Notre but est de gérer la chaîne grâce à notre intégration entre IXTET/Reco et C-PRS. Ici, C-PRS va servir de système réactif : il va gérer la chaîne en fonctionnement normal et proposer des méthodes pour rétablir ce fonctionnement et corriger les erreurs lorsqu'IXTET/Reco les aura reconnues au moyen de chroniques appropriées. D'un point de vue plus pratique C-PRS ordonnera d'ouvrir et de fermer une cuve quand un flacon passera sous celle-ci et ordonnera d'éjecter un flacon si nécessaire.

4.3.3 Identification des problèmes et construction de l'application

Problèmes possibles et chroniques résultantes

Nous avons décidé de distinguer deux problèmes possibles distincts. Le premier est qu'un capteur de type 3 peut tomber en panne. Dans ce cas-là, nous indiquerons que le problème existe et nous utiliserons une stratégie de remplacement pour ouvrir la vanne et la refermer, au risque d'avoir du liquide en dehors des flacons. Le deuxième problème que nous choisissons de considérer est qu'une cuve peut-être vide à un moment quelconque. Dans ce cas, nous lancerons une méthode permettant de la changer (ou de la remplir).

Nous aboutissons donc à trois chroniques distinctes (voir Annexe C):

- `chronicle capteur_3_marche_pas` : détecte au moyen d'un temps limite au-delà duquel un capteur ne répond plus.
- `chronicle Pb_cuve` : détecte qu'une cuve est vide au moyen d'un temps limite entre le moment où devrait s'écouler du liquide après ouverture de la vanne et l'instant présent.
- `chronicle Plus_remplir_flacon_cuve_vide` : détecte qu'un flacon vient de passer sous une cuve qui est vide ou en train d'être changée.

Construction de l'application

Après avoir implanté ces chroniques, nous avons créé les KAs C-PRS qui vont nous permettre de gérer les cuves et l'éjection. Celles-ci sont divisées en trois catégories :

- les KAs de gestion d'événements exceptionnels qui vont contenir les méthodes qui vont permettre de gérer un événement exceptionnel (exemple : changer_cuve).
- les KAs chargées de gérer les évènements standard de manière automatique (exemple : gerer_cuve).
- les KAs d'initialisation qui sont déclenchées par le simulateur et qui vont déclencher les KAs de gestion d'événements au début de la simulation.

4.3.4 Réalisation technique

Nous avons réalisé cette application de telle sorte que l'on puisse gérer un certain nombre de cuves (inférieur à sept dans notre version du fait de la déclaration des variables voir figure C.1, Chapitre 2) à spécifier à l'initialisation. De plus, nous avons écrit un simulateur en C-PRS qui gère les cuves et les flacons indépendamment de la marche théorique du superviseur (voir figure 4.7). Ce simulateur a été écrit en C-PRS pour des raisons de simplicité au niveau de la communication et de la programmation en C-PRS de ce type d'outil, on pourrait parfaitement le réaliser dans un autre langage plus performant.

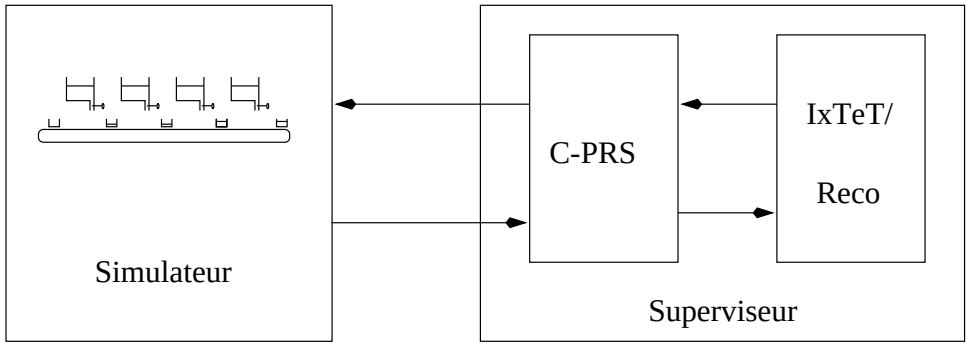

FIG. 4.7 – *Structure générale de l'application de supervision de chaîne de remplissage*

Les principaux problèmes rencontrés étaient des problèmes de mise au point des temps nécessaires au bon déroulement des opérations. De plus, pour peu que l'on fasse s'exécuter le simulateur en parallèle sur une machine distante (le noyau C-PRS gérant le simulateur étant assez gourmand en temps-machine), il n'y a aucun aucun problème de ralentissement (machine de référence : Sparc 20). On peut noter que du fait de la structure de communication utilisée, on ne peut envoyer à IXTET/Reco des événements que dans un ordre chronologique.

Critique de l'application réalisée

La qualité du simulateur pourrait être améliorée en utilisant un langage de programmation moins gourmand en ressources. En effet, du fait que nous n'utilisons pas toutes les fonctionnalités de C-PRS (et notamment la base de donnée) conduit à une perte de temps lors de la gestion du simulateur. Ceci nous conduit à faire exécuter ce dernier sur une machine distante pour ne pas utiliser du temps machine inutilement.

L'approche en elle-même semble être lourde au niveau des messages échangés entre les différentes composantes de notre application : on retrouve les mêmes problèmes que précédemment avec un maillon de plus dans la chaîne de communication. Cette application met donc en évidence des problèmes que nous avions déjà identifiés comme la lenteur de la connexion et la lourdeur du simulateur; elle a néanmoins un intérêt non négligeable : elle a permet de connecter IXTET/Reco de manière efficace à n'importe quel type d'outil du fait de la création d'*IxR-server*. On a ainsi, en partie, comblé une lacune d'IXTET/Reco : sa difficulté à communiquer avec d'autres programmes.

Chapitre 5

Prospective

Malgré la connexion que nous avons établie, qui permet de faire des tests encourageants pour notre approche, nous n'avons fait que réaliser une passerelle entre deux systèmes. Ce travail, n'est qu'un premier pas dans une démarche qui vise à faire de la supervision comme nous l'avons détaillée au chapitre 2. Nous proposons ici des possibilités pour réaliser une intégration plus importante qui permettrait de corriger les problèmes que l'on a pu observer au Chapitre 4.

5.1 Introduction

Nous avons pu constater que notre connexion souffrait des temps de parcours observés. Une première approche est de supprimer tous les processus que l'on a mis en place pour les remplacer par des processus légers (*threads*); on éviterait ainsi la perte de temps nécessitée par les commutations entre les processus. Une autre possibilité est de temporiser la base de données de C-PRS et d'implanter les algorithmes de gestion d'arbres présents dans IXIA/Reco. La troisième approche que nous présentons ensuite correspond à une combinaison de ces deux aspects un peu modifiés et nous paraît constituer l'outil final le plus complet.

5.2 Utilisation des processus légers (*threads*)

5.2.1 Approche proposée

Le moyen le plus aisément utilisable pour accélérer notre système est d'utiliser des processus légers [1] pour implémenter le parallélisme qui a été détaillé précédemment on aurait ainsi un gain certain en temps puisqu'on ne changerait plus de processus (temps de commutation réduit) et que l'on partage un certain nombre de données entre chaque processus léger. On évite la duplication des données internes que nous observons actuellement. Cette approche est néanmoins faible par rapport à ce que nous proposerons par la suite car l'utilisateur se doit encore de programmer selon les deux formalismes et de connaître le fonctionnement de la connexion pour pouvoir rédiger ses programmes. De plus, garder les processus lourds permet de paralléliser sur des machines distantes les différents processus.

1. Nous entendons par processus lourds les processus UNIX au sens classique du terme et par processus légers les *threads* défini selon la norme POSIX.

5.2.2 Difficultés de mise en oeuvre

On a déjà des structures complexes qui permettent de communiquer entre les divers processus (comme des *sockets*). Pour bien faire les choses il faudrait créer des structures de données partagées qui permettraient d'éviter d'utiliser ces ressources gourmandes en temps. Une première approche serait de mettre en place cette approche uniquement sur les trois processus qui constituent IXTET/Reco (voir Chapitre 4).

Un autre facteur entre en ligne de compte : on peut actuellement lancer la partie IXTET/Reco, le Message-Passer et le noyau C-PRS sur des machines distantes, parallélisant ainsi les processus. Le fait de transformer tous ces processus lourds en threads rendrait impossible cette fonctionnalité.

Un dernier facteur non négligeable estn à considérer : la taille des outils à modifier ne facilite pas les mises à jour et les changements d'architecture comme ceux que nous proposons.

5.3 Temporisation de la base de données de C-PRS

5.3.1 Approche proposée

Il serait dommage de temporiser tous les éléments présents dans la base de données de C-PRS. En effet, certains n'ont d'intérêt que parce qu'ils sont valables dans l'instant présent ou à tout jamais (pour reprendre l'exemple sur les nombres premiers développé au Chapitre 3, quel intérêt y aurait-il à savoir que l'on a conclu que 101 est premier a 16h00 le 20 janvier 1998). Nous proposons donc de définir un nouveau type de faits que l'on pourrait appeler *Temporal Functional Fact(tff)* pour lesquels on garderait tous les changements de valeur avec leur date courante. On pourrait alors utiliser les algorithmes développés dans IXTET/Reco pour effectuer les reconnaissances en postant un but du type *(!(chronicle (hold Aa Va Ta1 Ta2)(event Ab Vb1 Vb2 Tb)....(Ta1 < Tb)))* où *Aa* et *Ab* sont des tff, *Va Vb1* et *Vb2* sont des valeurs et *Ta1 Ta2* et *Tb* sont des dates. Le *when recognized* étant alors modélisé par la suite de la KA. On perd alors le phénomène de duplication présent dans IXTET/Reco. Cet inconvénient peut être corrigé en indiquant au noyau C-PRS qu'une chronique est toujours présente en arrière-plan (par exemple).

5.3.2 Difficultés de mise en oeuvre

Cette approche est insatisfaisante dans le sens où elle apparaît très coûteuse en temps de développement et pas forcément très performante. En effet, le fait que l'algorithme de mise à jour d'arbres des chroniques soient interprété ne semble pas très prometteur au niveau des performances en temps-réel. On risque donc de réaliser un travail énorme sans obtenir de résultat probant.

5.4 Vers une intégration d'IXTET/Reco dans C-PRS

5.4.1 Présentation générale de l'outil proposé

Une autre solution est de créer un type de KA particulier (qui s'appellerait KA Chronicle) qui serait, en fait l'ensemble des chroniques exécutables, appelables. Ces chroniques seraient alors activables par un but *(!(KAC-ACTIVATE ...))* et désactivables par un autre but *(!(KAC-UNACTIVATE ...))* et leurs reconnaissances récupérables au moyen de *(RECO ... $TIME)*. Ces KA Chronicles auraient alors des arguments et pourraient être programmées soit de manière graphique soit de manière textuelle (comme les autres KAs).

5.4.2 Présentation logicielle de l'outil proposé

Du point de vue logiciel, les KA Chronicles seraient séparément compilées au chargement des fichiers de KAs (procédures PRS) pour conserver leurs temps de réponse performants. Le noyau gérant ces KAs serait alors un noyau IxTeT/Reco appelé par le noyau C-PRS (multithread) un peu modifié pour pouvoir activer et désactiver les chroniques (devenues KA Chronicles)(voir figure 5.1). Les faits utilisés pour de telles KAs seraient des faits déclarés comme *tff* mais on ne garderait pas les dates de changement de valeurs (inutiles du fait de l'activation et la désactivation des KAs). Ces *tff* seraient alors des *Functional Facts* qui auraient un traitement particulier dans le cas où des KA Chronicles seraient activées.

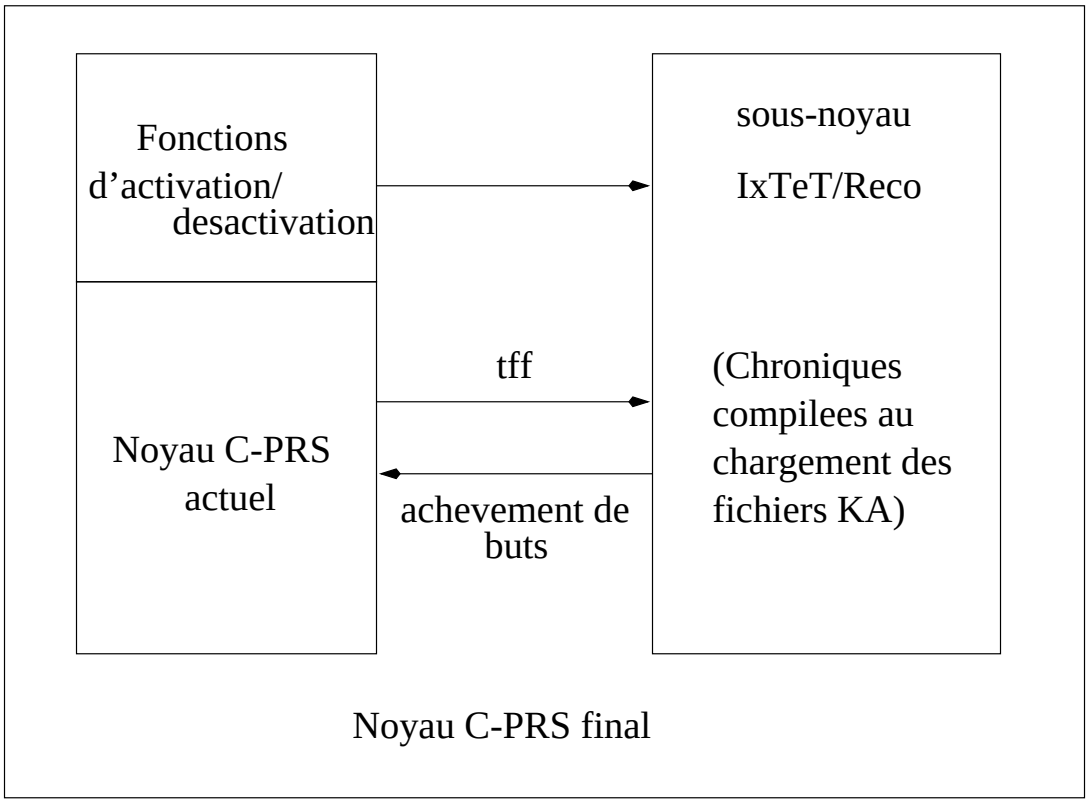

FIG. 5.1 – *Structure générale de l'outil final proposé.*

Il semble important de garder les *Functional Facts*(ff) sous leur forme première pour ne pas surcharger les communications inutilement dans le cas où on aurait besoin de *ff* uniquement pour la partie réactive. De cette manière, on n'envoie aux chroniques activées que les faits susceptibles de leur être utiles.

5.4.3 Conclusion

Le travail réalisé lors de ce stage apparaît donc comme un premier pas vers une intégration entre les fonctionnalités de C-PRS et celles d'IxTeT/Reco qui donnerait un superviseur capable de réaliser de la supervision en temps-réel avec les propriétés que nous avons détaillées au Chapitre

2. Ce que nous avons déjà implémenté peut être une solution acceptable pour des systèmes qui ne génèrent pas trop de messages. On peut regretter que trois formalismes continuent de coexister dans cette approche. Ceci signifie qu'un nouvel utilisateur a de la difficulté à prendre en main le système puisqu'il doit acquérir la connaissance des formalismes qui lui sont inconnus. Le travail réalisé est néanmoins prometteur et nous permet de mieux comprendre nos besoins. Il nous encourage à progresser sur cette voie de manière à corriger ses lacunes.

Annexe A

Détail du calcul de complexité

On se propose de calculer la complexité des tests évoqués au Chapitre 4. On a précédemment posé :

n nombre d'événements.

k nombre maximum d'unifications possibles pour chaque événement.

p nombre de contraintes de temps et de contraintes de persistance dans le temps, on obtient les complexités suivantes.

Schématiquement, pour effectuer une évaluation de la formule, au pire cas, il faut calculer toutes les feuilles de l'arbre des unifications possibles pour les n événements qui ont chacun k unifications possibles. On a donc déjà k^n feuilles. Pour chacune de ces feuilles on doit, au pire cas, tester encore n contraintes de temps d'ou une complexité en $\mathcal{O}(k^n p)$.

Pour reconnaître une chronique, il faut reconnaître les n événements qui la composent (au moins); d'où n évaluations de la formule. On a alors une complexité en $\mathcal{O}(nk^n p)$.

Annexe B

KAs et chroniques de l'aller-retour

Nous fournissons ici les chroniques et les KAs ayant permis de calculer les temps de parcours exhibés au chapitre 4.

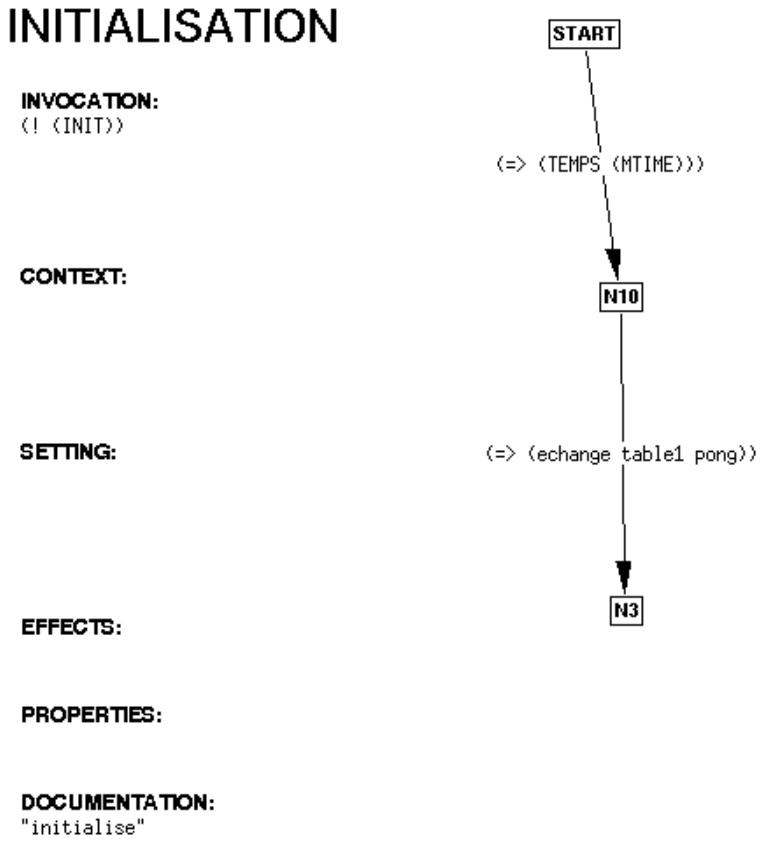

FIG. B.1 – *Procédure d'initialisation* INIT.

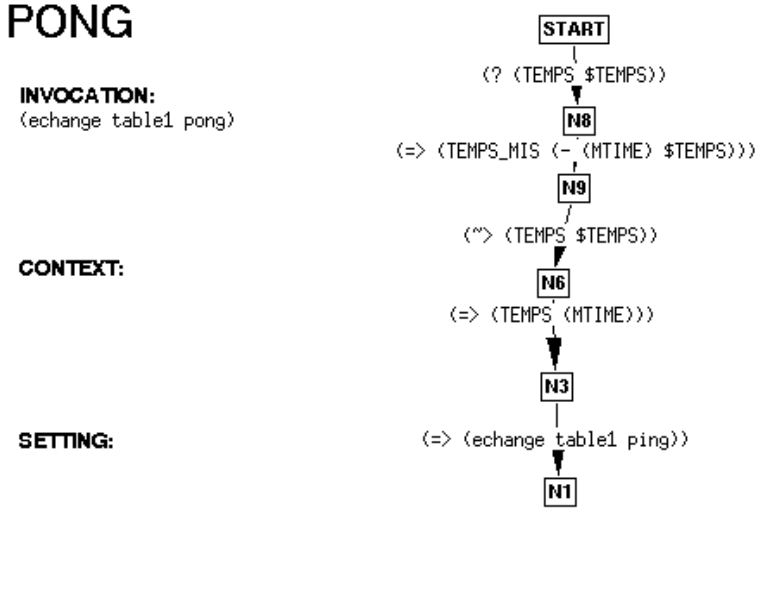

FIG. B.2 – *Procédure servant à renvoyer le message* PONG.

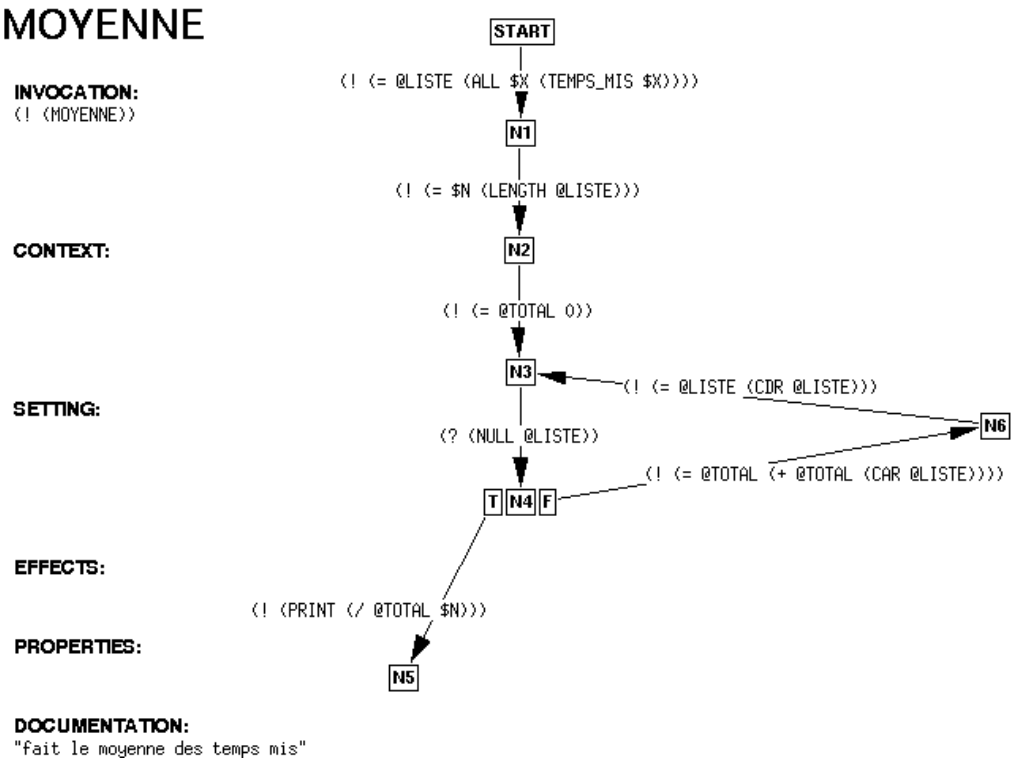

FIG. B.3 – *Procédure de calcul de la* MOYENNE *des temps mis.*

```
constant ETAT_ECHANGE= {ping, pong};
// etat de l'echange
TABLE_POS= {table1};
attribute echange(?ta) {
?value in ETAT_ECHANGE;
?ta in TABLE_POS;
}
chronicle ping ()(t_debut) {
event (echange(table1): (pong,ping),t_debut);
when recognized {
call send_pong();
report "ping";
}
}
```

FIG. B.4 – *Chronique servant au calcul du temps de parcours*

```
extern void write_to_PRS(char *);
void send_pong (const RecoInstance& S)
{ char *strong;
strong="(echange table1 pong)";
write_to_PRS(strong);
}
```

FIG. B.5 – *Fonction C++ utilisée par la chronique*

Annexe C

Chroniques de l'application

Nous allons développer un exemple d'ensemble de chroniques qui est utilisé dans l'application que nous avons construite au Chapitre 4 pour tester notre travail.

C.1 Généralités, déclaration de variables

Nous nous intéressons à la détection de dysfonctionnements qui pourraient survenir lors de remplissages de flacons qui se trouvent sous une cuve remplie de liquide. Nous identifions deux problèmes distincts : le fait qu'un capteur (de type 3 en l'occurence) ne marche pas et la possibilité qu'une cuve soit vide[1].
Intéressons-nous maintenant à la déclaration des variables (attributs) nécessaire à la reconnaissance (voir figure C.1).

On peut remarquer que toutes les variables ont des domaines discrets à déclarer en premier lieu et que chaque attribut a comme domaine ces variables. Ceci est, en partie dû au fait que l'on compile les chroniques au préalable et cela participe au processus d'optimisation qui est utilisé par cette compilation.

C.2 Détail des chroniques

Nous avons pu dégager trois chroniques différentes :

capteur_3_marche_pas : cette chronique va détecter qu'un capteur de type 3 ne marche pas.

Pb_cuve : détecte qu'une cuve est vide.

Plus_remplir_flacon_cuve_vide : détecte qu'une cuve est vide et propage le fait qu'un flacon n'est pas rempli.

Détaillons la chronique *capteur_3_marche_pas* (voir figure C.2).

On peut voir dans cette chronique qu'elle se décompose en plusieurs parties : une partie en-tête, une partie déclaration des variables utiles à la chronique (les variables sont repérées par un *?*), une partie contenant des assertions et des événements ainsi que des contraintes temporelles et une

1. Pour plus de détails sur le fonctionnement des cuves voir Chapitre 4, Section 3, sous-section 2.

```
constant CAPTEURS = {C1,C2,C3,C4,C5,C6,C7};
constant RESULTAT_CAPTEUR_1 = {FERMEE, OUVERTE};
constant RESULTAT_CAPTEUR_2 = {COULE,COULE_PAS};
constant RESULTAT_CAPTEUR_3 = {ABSENCE, PRESENCE};
constant ETAT_CAPTEUR = {MARCHE,MARCHE_PAS};

//les differents capteurs

attribute ETAT_CAPTEUR_1 (?Capteur) {
?Capteur in CAPTEURS;
?value in ETAT_CAPTEUR;
}

attribute ETAT_CAPTEUR_2 (?Capteur) {
?Capteur in CAPTEURS;
?value in ETAT_CAPTEUR;
}

attribute ETAT_CAPTEUR_3 (?Capteur) {
?Capteur in CAPTEURS;
?value in ETAT_CAPTEUR;
}

attribute CAPTEUR_1 (?Capteur) {
?Capteur in CAPTEURS;
?value in RESULTAT_CAPTEUR_1;
}

attribute CAPTEUR_2 (?Capteur) {
?Capteur in CAPTEURS;
?value in RESULTAT_CAPTEUR_2;
}

attribute CAPTEUR_3 (?Capteur) {
?Capteur in CAPTEURS;
?value in RESULTAT_CAPTEUR_3;
}
```

FIG. C.1 – *Déclaration des variables utilisées dans les chroniques de l'exemple*

dernière partie indiquant les actions à effectuer quand la chronique est reconnue. En l'occurence, on peut détailler le comportement de la chronique. Elle concerne un capteur (variable *?Capteur*) et elle est déclenchée à *t_debut* si les conditions sont vérifiées. La chronique en elle-même est constituée d'un seul événement (le changement d'état du capteur considéré), de deux assertions (le fait que le capteur n'a pas changé d'état depuis et le fait que l'on considère le capteur en marche) et d'une contrainte temporelle (*t_avant* et *t_debut* sont séparés par plus de six secondes). On peut résumer cette chronique par la phrase suivante : *"Si un capteur de type 3 qui est censé fonctionner n'a pas changé d'état en six secondes, alors..."*. Ce que l'on rajoute après le *"alors"* est contenu dans *when recognized*{ ... }. En l'occurrence, on appelle deux fonctions en C++ (*call...*) et on rapporte sur

```
chronicle capteur_3_marche_pas (?Capteur)(t_debut) {
variable?Etat,?Etat_avant;
?Capteur in CAPTEURS;
?Etat in RESULTAT_CAPTEUR_3;
?Etat_avant in RESULTAT_CAPTEUR_3;
timepoint t_avant;
event (CAPTEUR_3 (?Capteur): (?Etat_avant,?Etat),t_avant);
hold (CAPTEUR_3 (?Capteur):?Etat, (t_avant,t_debut));
hold (ETAT_CAPTEUR_3 (?Capteur): MARCHE, (t_avant,t_debut));
(t_debut-t_avant) > 6.00;
// attention,ici c'est 6.00!!!!!! (delai)
when recognized {
call send_prs_capteur_3_marche_pas(?Capteur);
call send_prs_pas_remplir(?Capteur);
report "detection de fin de remplissage de flacon";
report "capteur_3 marche pas";
}
}
```

FIG. C.2 – *Chronique* capteur_3_marche_pas

l'interface utilisateur ce que l'on a reconnu (*report ...*).

On a un autre exemple de minuterie avec la chronique *Pb_cuve* (voir figure C.3).

```
chronicle Pb_cuve (?Capteur)(t_debut) {
?Capteur in CAPTEURS;
timepoint t_avant;
hold (ETAT_CAPTEUR_2 (?Capteur): MARCHE, (t_avant,t_debut));
hold (ETAT_CAPTEUR_1 (?Capteur): MARCHE, (t_avant,t_debut));
hold (CAPTEUR_2 (?Capteur): COULE_PAS, (t_avant,t_debut));
event (CAPTEUR_1 (?Capteur): (FERMEE,OUVERTE),t_avant);
(t_debut-t_avant) > 1.50;
// attention,ici c'est 1.50!!!!!! (delai)
when recognized {
report "cuve_vide detectee";
call send_prs_cuve_vide(?Capteur);
call send_prs_flacon(?Capteur);
}
}
```

FIG. C.3 – *Chronique* Pb_cuve

Cette chronique peut être traduite en langage naturel par : *" si les capteurs de type 1 et 2 marchent et que le capteur de type 1 a indiqué que l'on a ouvert une vanne mais que rien ne coule pendant une seconde et demie, alors on rapporte qu'une cuve vide a été détectée et on appelle deux fonctions en C++"*. On peut remarquer que les événements et les assertions n'ont pas, syntaxiquement, une place fixe les uns par rapport aux autres.

Examinons finalement la chronique *Plus_remplir_flacon_cuve_vide* (voir figure C.4) qui fonctionne un peu différemment des autres et qui va nous permettre de décrire le phénomène de

duplication présent dans IxTeT/Reco.

```
chronicle Plus_remplir_flacon_cuve_vide (?Capteur)(t_debut){
variable ?Etat;
?Capteur in CAPTEURS;
?Etat in RESULTAT_CAPTEUR_3;
timepoint t_avant;
hold (ETAT_CAPTEUR_2 (?Capteur): MARCHE, (t_avant,t_debut));
hold (ETAT_CAPTEUR_1 (?Capteur): MARCHE, (t_avant,t_debut));
hold (ETAT_CAPTEUR_3 (?Capteur): MARCHE, (t_avant,t_debut));
hold (CAPTEUR_2 (?Capteur): COULE_PAS, (t_avant,t_debut));
event (CAPTEUR_1 (?Capteur): (FERMEE,OUVERTE),t_avant);
event (CAPTEUR_3 (?Capteur): (ABSENCE,PRESENCE),t_debut);
(t_debut-t_avant) > 1.00;
when recognized {
report "detection de fin de remplissage de flacon : cuve vide";
call send_prs_pas_remplir(?Capteur);
}
}
```

FIG. C.4 – *Chronique* Plus_remplir_flacon_cuve_vide

Le début de la chronique *Plus_remplir_flacon_cuve_vide* est identique à celui de la chronique *Pb_cuve* la seule différence réside dans le fait que l'on considère un autre événement en plus : le fait qu'un capteur de type 3 passe de la valeur *ABSENCE* à la valeur *PRESENCE*. Cet événement va conduire au fait qu'aprés avoir reconnu qu'une cuve a un problème et que ce dernier n'a pas été réparé, à chaque fois que le capteur de type 3 correspondant passera de la valeur *ABSENCE* à la valeur *PRESENCE*, on reconnaitra la chronique. Ce comportement observé correspond à la duplication : à chaque arrivée d'un nouvel événement qui concerne une chronique on crée un noeud qui contient le nouvel évènement pris en compte et on laisse actif l'ancien noeud.

Bibliographie

[1] M. Basseville and M.-O. Cordier. Surveillance et diagnostic de systemes dynamiques : approches complementaires du traitement de signal et de l'intelligence artificielle. Technical Report 1004, Institut de Recherche en Informatique et Systemes Aleatoires, 1996.

[2] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *IEEE Computer* , 79(9), Septembre 1991.

[3] M. Boddy and T. Dean. Solving time-dependent planning problems. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'89)*, 1989.

[4] C. Dousson. *Suivi d'Évolutions et Reconnaissance de Chroniques*. PhD thesis, Université Paul Sabatier de Toulouse préparée au Laboratoire d'Analyse et d'Architecture des Systemes (LAAS-CNRS), Septembre 1994.

[5] F.F.Ingrand, M.P. Georgeff, and A. S. Rao. An architecture for real-time reasonning and system control. In *IEEE Expert* , Dec 1992.

[6] R.J. Firby. An investigation into reactive planning in complex domains. In *AAAI-6*, 1987.

[7] M. P. Georgeff and F. F. Ingrand. Research on procedural reasoning systems. Final report, phase 1, AI center, SRI International, Menlo Park, Cal, 1988.

[8] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'89)*, 1989.

[9] M. Ghallab and A. Mounir-Alaoui. Managing efficiently temporal relations through indexed spanning trees. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'89)*, 1989.

[10] G. De Giacomo, Y. Lesperance, and H. J. Levesque. Reasonning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, 1997.

[11] D. M. Lyons and A. J. Hendriks. A practical approach to integrating reaction and deliberation. In *AIPS Artificial Intelligence Planning Systems (AIPS'92)*, 1992.

[12] D. M. Lyons and A. J. Hendriks. Exploiting patterns of interaction to achieve reactive behavior. *Artificial Intelligence (AI)*, (73), 1995.

[13] D. McDermott and J. Hendler. Planning : What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence (AI)*, (76), 1995.

[14] C. B. McVey, E. M. Atkins, E. H. Durfee, and K. G. Shin. Developpement of iterative real-time scheduler to planner feedback. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, 1997.

[15] D. J. Musliner, E. H. Durfee, and K. G. Shin. A cooperative intelligent real-time control architeecture. *IEEE Transactions on Systems,Man and Cybernetics*, 23(6), 1993.

[16] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul. The challenges of real-time ai. *IEEE Computer* , 28(1), Janvier 1995.

[17] K. L. Myers. A procedural knowledge approach to task-level control. In *AIPS Artificial Intelligence Planning Systems (AIPS'96)*, 1996.

[18] B. Pell, E. Gamble, E. Gat, R. Keesing, J. Kurien, B. Millar, P. P. Nayak, C. Plaunt, and B. Williams. A hybrid procedural/deductive executive for autonomousspacecraft. In *Second International Conference on AUTONOMOUS AGENTS (Agents '98)*, 1998.

[19] B. Pell, E. Gat, R. Keessing, N. Muscettola, and B. Smith. Robust periodic planning and execution for autonomous spacecraft. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, 1997.

[20] P. Serra, M. Sànchez, J. Lafuente, U. Cortés, and M. Poch. Depur : a knowledge based tool for wastewater treatment plants. In Pergamon Press, editor, *Engineering Applications of Artificial Intelligence*, 1994.

[21] B. C. Williams and P. P. Nayak. A reactive planner for a model-based executive. In *IJCAI Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, 1997.