MINISTERE DE L'EDUCATION NATIONALE DE LA RECHERCHE ET DE LA TECHNOLOGIE

******

Institut National des Sciences Appliquées

Département de Génie Electrique et Informatique

******

PROJET DE FIN D'ETUDES
DEA Systèmes Informatiques

Contrôle-commande pour les nacelles pointées à base de technologies standard

Une contribution : prototypage d'applications embarquées réparties

Centre Spatial de Toulouse

18 avenue Edouard Belin
31 401 TOULOUSE CEDEX 4
FRANCE

Seuma Vidal Jean-Pierre
5GII-TRS 2

Septembre 2002

MINISTERE DE L'EDUCATION NATIONALE DE LA RECHERCHE ET DE LA TECHNOLOGIE

******

Institut National des Sciences Appliquées

Département de Génie Electrique et Informatique

******

PROJET DE FIN D'ETUDES
DEA Systèmes Informatiques

Contrôle-commande pour les nacelles pointées à base de technologies standard

Une contribution : prototypage d'applications embarquées réparties

Centre Spatial de Toulouse

CENTRE NATIONAL D'ETUDES SPATIALES

18 avenue Edouard Belin
31 401 TOULOUSE CEDEX 4
FRANCE

Seuma Vidal Jean-Pierre
5GII-TRS 2

Septembre 2002

REMERCIEMENTS

Je remercie vivement Jean Evrard pour m'avoir accueilli au sein de son équipe et André Laurens pour m'avoir permis d'effectuer ce stage et pour son soutien et ses conseils avisés.

Je tiens à remercier Philippe Laporte pour sa gentillesse et sa disponibilité.

Je remercie André Pélissier et Alain Vecten pour leur sympathie.

Je remercie également Semra Sarpdag pour avoir posé les bases du travail et pour son introduction sur les nacelles pointées.

Je remercie par ailleurs Eric Poupart et Vincent Berjot, compagnons de nombreuses discussions dans la bonne humeur.

Enfin merci à toute l'équipe Nacelles pointées pour l'accueil qu'elle m'a réservé en février et pour m'avoir donné l'occasion de participer au lancement d'un ballon à Aire-sur-l'Adour, le 22 avril 2002.

SOMMAIRE

INTRODUCTION

Le document présent s'inscrit dans le plan de modernisation des architectures informatiques des nacelles pointées initié il y a deux ans. Cet effort repose sur l'idée que les changements de technologie sur du matériel industriel existant et opérationnel peuvent se révéler opportuns. Mais les choix effectués peuvent laisser les personnes impliquées dans le projet dubitatives quant à la capacité du nouveau système à répondre aux mêmes exigences que l'ancien. Il ne sera pas ici question de convaincre quiconque de l'intérêt d'un changement d'architecture mais bien d'essayer de la mettre en œuvre, en mettant en évidence ses points forts et ses éventuelles faiblesses.

J'ai effectué mon stage de DEA d'une durée de six mois (mars-août 2002), au sein de l'équipe des Nacelles Pointées, à la division Ballons du CNES. Ce rapport présente l'état d'avancement à la fin du mois d'août du prototypage d'applications réparties sur plusieurs calculateurs embarqués à bord de nacelles pointées.

On ne peut pas séparer la conception du prototype de l'application à proprement parler des aspects méthodologiques et l'étude de la faisabilité d'une architecture répartie. Ces dernières donnent lieu à un travail qui repose sur des études déjà effectuées. En conséquence, il aurait été inopportun de se lancer dans une tentative de conception détaillée, sans étudier d'abord le modèle de répartition. Nous nous efforcerons ici de présenter et de justifier les choix effectués pour la nouvelle architecture informatique des logiciels bord et sol des nacelles pointées, et de définir et tester l'architecture répartie.

1. DOMAINE D'APPLICATION

1.1. LES NACELLES POINTEES

Une nacelle pointée est un véhicule d'emport d'expériences scientifiques sous ballons stratosphériques. La particularité de ce type de nacelle est la présence d'un système de pointage destiné à orienter et maintenir la charge utile (l'instrument scientifique) dans une direction bien précise.
La nacelle est destinée à être acheminée à une altitude qui peut aller jusqu'à 45 km. Le poids de celle-ci peut atteindre 500 kg qui est la limitation réglementaire en France. C'est par l'intermédiaire d'un ballon qu'elle va être transportée dans la stratosphère. Elle est reliée à celui-ci via un parachute pour la récupérer et une chaîne de vol qui est constituée de sangles ou câbles porteurs, et elle est équipée de divers dispositifs de conduite du vol (réflecteur ou répondeur radar, GPS, boîtier de commande du délestage, séparateurs ballon-parachute-nacelle, équipements de télécommunication associés).
Les domaines scientifiques intéressés par les vols stratosphériques sont nombreux, car le ballon est un véhicule peu coûteux qui permet à une expérience de s'affranchir de l'essentiel de l'atmosphère, donc des inconvénients qui y sont souvent liés comme l'atténuation du signal scientifique et l'émissivité parasite. Néanmoins, les clients des nacelles pointées appartiennent essentiellement à deux domaines :

- les instruments d'observation astronomique ou astrophysique. Le rôle du système de pointage sera de diriger ces instruments dans la direction des objets astronomiques souhaités, et ce avec la précision et la stabilité requises. Il peut y avoir plusieurs objets à observer au cours d'un même vol ;
- les expériences de chimie atmosphérique ont pour but essentiel d'identifier les espèces chimiques qui composent l'atmosphère, et d'en déterminer les concentrations en fonction de l'altitude. Une des méthodes de mesure les plus fréquentes est la spectrométrie par occultation : en visant un astre (typiquement, le soleil qui traverse les différentes couches de l'atmosphère) du zénith jusqu'à son coucher, on verra le spectre du soleil se modifier en fonction de la "quantité" d'atmosphère traversée, donc des espèces rencontrées et de leur concentration. Dans ce cas, le rôle du système de pointage sera d'orienter et maintenir la charge utile dans la direction de l'astre en question.

Quelle que soit la nature de la mission, la nacelle transporte :

- des équipements de mesure et de traitement scientifique (instrument scientifique);
- une source d'énergie électrique : piles ou batteries ;
- des équipements d'acquisition et de surveillance de paramètres de servitude (capteurs de température, gyromètres, magnétomètres, inclinomètres, etc);
- un système de localisation (GPS le plus souvent) ;
- un système de télécommunication avec le sol (équipements de télécommande/télémesure (TM/TC) Cnes) ;

- les équipements constituants le système de pointage (capteurs et actionneurs).

Elle embarque à son bord un ou deux calculateurs :

- dans tous les cas, le module de commande, logiciel (MC) qui réalise les fonctions de pointage primaire en azimut et la gestion des servitudes : contrôle des bus d'alimentation, acquisition des mesures de température, tensions et courants.
- et selon la mission, le module de pointage fin, logiciel (MPF) qui réalise les fonctions de pointage primaire en élévation et un pointage très précis .

Un calculateur est actuellement constitué de cartes PC104 qui sont comparables à de véritables PC industriels tenant sur une carte de 10*10 centimètres et qui comprend toutes les interfaces d'un PC standard. Il intègre une carte réseau, est dépourvu de disque dur, mais c'est une carte Flash PCMCIA de 256 Mo qui le remplace. On peut, en empilant des cartes sur un bus, disposer de cartes d'entrées/sorties analogiques, logique, série, commandes moteurs.

Ces calculateurs exécutent des applications développées en c++ et utilisant le système d'exploitation temps-réel VxWorks.

1.2. LES TRAVAUX DE L'EQUIPE NACELLES POINTEES

L'équipe Nacelles pointées a en charge la définition, la réalisation et la mise en œuvre des nacelles qui nécessitent un pointage solaire ou stellaire pour les besoins scientifiques des utilisateurs.

C'est elle qui va analyser les besoins de la mission, évaluer sa faisabilité et mettre en place les différents dispositifs pour assurer la meilleure intégration de l'expérience à la nacelle. D'une mission à l'autre, les équipements sont récupérés afin de les réutiliser. Mais certains projets scientifiques peuvent nécessiter des développements spécifiques.

L'équipe réalise :

- le système de pointage;
- la chaîne de contrôle-commande qui comprend, entre autre, le sous-système informatique de bord (calculateurs embarqués, bus de communication, périphériques informatiques, logiciels) ;
- l'intégration de l'expérience scientifique;
- le test et la validation de la nacelle ;
- l'opération de la nacelle en vol.

Le 22 avril 2002, j'ai eu la chance d'assister au lancement d'une nacelle embarquant des senseurs stellaires, "caméras" construites sur le principe des CCD (Coupled Charged Device) et pointant un objet céleste. Ce vol était uniquement "technologique" et avait pour but de tester des nouveaux équipements comme la roue à inertie, un système d'équilibrage de la nacelle, et une instrumentation de mesure du fond de ciel. J'ai pu me rendre compte du travail de préparation qui précède un vol et de la fiabilité requise par

les équipements au cours d'un vol, qui somme toute n'a pas duré longtemps. En mettant de côté les durées de montée et de descente et en ne considérant que la phase de plafond pendant laquelle le ballon est maintenu à une certaine altitude, l'équipe n'a pu disposer que de deux heures de plafond pour mettre en œuvre les tests.

1.3. LES IDEES DIRECTRICES ET L'ETAT DE L'ART

Actuellement, André Laurens, ingénieur en informatique de l'équipe, prend en charge l'évolution et la maintenance du logiciel de vol et du sol. Les logiciels de vol et de sol sont tous les deux programmés en C++, mais tournent sous un environnement différent. Le premier dispose du système d'exploitation VxWorks et le second de Windows 98.
Son objectif est de faire évoluer l'architecture du système informatique afin :

- d'obtenir une application plus flexible vis-à-vis des missions (flexibilité des fonctions) et des équipements (diversité des interfaces);
- de réduire les coûts et les délais;
- de disposer d'un environnement de développement (matériel et logiciel) qui permet d'uniformiser les technologies utilisées entre bord et sol, en s'appuyant sur des solutions standards, puissantes, peu coûteuses et pérennes.

La base de la solution repose sur le fait que les cartes PC104 « embarquables » sont de plus en plus puissantes et intégrées, ce qui permet de les faire fonctionner comme des calculateurs sol, donc avec des systèmes d'exploitation et des chaînes de développement standards, dits de "monsieur tout le monde".
Le système d'exploitation retenu par mon responsable de stage est Linux.
En effet, beaucoup d'applications utilisent aujourd'hui un Linux embarqué, entre autre, sur des PC104 et au prix d'une configuration du noyau légèrement différente de ce que l'on trouve sur les calculateurs sol. On obtient alors une cible bord tout à fait standard, avec des fonctionnalités et un confort de développement analogues aux systèmes sol. Le développement de logiciels bord s'effectue alors en natif et non plus en croisé.
Du côté applicatif, deux décisions ont été prises :

- l'utilisation de la méthode de conception HOOD et de l'outil STOOD, afin d'obtenir une architecture saine, fiable et évolutive;
- le développement en Ada95, qui outre ses vertus de modularité et maintenabilité, offre un multi-tâches préemptif, ce qui permet de développer des applications temps-réel comme les logiciels de bord.

Dans ce cadre, le travail effectué par Semra Sarpdag, stagiaire qui m'a précédé l'an dernier a été double. D'une part, elle s'est appliquée à rechercher et à mettre en place une distribution de Linux adaptée aux besoins et, d'autre part, elle a débuté une conception générale HOOD de l'application.
Actuellement, je dispose d'un poste de développement avec une version Linux Mandrake 8.1, et de deux maquettes de calculateurs embarqués à base de cartes PC104 connectés au réseau local.
Une version Slackware de Linux est installée sur les deux maquettes, et l'exécution d'applications de type Ada95 ne présente aucun problème.
La traduction de classes et de fonctions C++ actuellement utilisées dans le logiciel de vol vers des paquetages et des procédures Ada95 a été commencé par André Laurens.

1.4. LES OBJECTIFS DE MON STAGE

Dans la première partie de mon stage, sur les bases du travail de Semra Sarpdag, je devais tester une application minimale Ada95 s'exécutant sur la carte PC104, représentative des tâches qui coexistent lors d'une exécution typique en vol en s'assurant du respect des contraintes temporelles. Il s'agit ici de vérifier en premier lieu, la réalité d'une solution Ada sur un Linux embarqué.

En deuxième approche, il s'agissait d'étudier le modèle de répartition décrit dans l'annexe de la norme du langage Ada95 et son implémentation par GNAT-GLADE pour évaluer son emploi dans notre application.

La dernière partie de mon stage s'attache à intégrer ces deux approches et à continuer la conception entreprise par Semra avec l'outil STOOD pour s'acheminer vers une application répartie sur plusieurs calculateurs bord.

Je tacherai d'abord dans ce rapport de présenter les principes d'architecture informatique pour les nacelles pointées, d'étudier le modèle de répartition GNAT-GLADE, et le travail effectué jusqu'à ce jour.

2. PRINCIPES D'ARCHITECTURE.

2.1. LE CONTEXTE GENERAL

Avant de présenter la nouvelle architecture logicielle à atteindre, définissons d'abord les axes d'évolution des solutions informatiques pour les nacelles pointées. Cette partie reprend certains aspects des documents internes de spécifications du contrôle-commande des nacelles pointées et [Laurens,2000]. On peut d'abord dégager quelques critères sur lesquels reposent les choix effectués.

Comme cela a été présenté dans la partie précédente, il est question de développer des architectures flexibles. Qu'est-ce à dire ?
Les fonctions du logiciel de vol d'un vol à l'autre ne sont pas figées, et à chaque nouvelle mission, des besoins nouveaux sont amenés, et peuvent faire l'objet de fonctions supplémentaires. La solution informatique devra permettre d'ajouter facilement des modules. Cette vision modulaire correspond à ce qu'offre le langage Ada95 par l'intermédiaire de ses paquetages. Il faut noter ici, que l'on aura une approche maximaliste, consistant à garder tous les composants déjà créés, de façon à pouvoir les réutiliser.

Deuxièmement, il faut rappeler le contexte économique lié aux nacelles pointées. En effet, cette équipe ne dispose pas des mêmes moyens alloués aux satellites, et le temps de réaction du développement d'applications sur des ballons est plus court que celui des satellites. De plus, en pratique, on ne peut pas attribuer aux applications embarquées sur des nacelles pointées l'appellation "hautement critique" du point de vue des contraintes temporelles. Malgré tout, c'est un aspect qu'il ne faudra pas négliger dans la conception et auquel il faudra porter une attention particulière.

Dans ces conditions, il est souhaitable de miser sur des solutions fiables car déjà éprouvées, facilement réalisables, et reposant sur un environnement de développement peu coûteux et confortable.

Le choix du système d'exploitation Linux paraît se prêter à notre cas sans risque. Ce système est diffusé mondialement en open source, et il est fourni avec tous les utilitaires périphériques d'un système d'exploitation (drivers, couches réseau, window manager …). A titre de comparaison, les systèmes d'exploitation traditionnels bord (VxWorks, Lynx,…) sont marqués « développement croisé », chers et d'un confort relatif. Bien qu'ils soient bien fournis en bibliothèques utilitaires, ce sont des OS propriétaires et il faut payer pour toute maintenance ou ajout de fonctionnalités. Le monde Linux offre l'avantage de rassembler à travers l'internet des communautés de travail et développement qui sont susceptibles de pouvoir aider grâce aux listes de questions.
On peut signaler que le système d'exploitation Linux n'est pas préemptif à la base, ce qui donne à l'emploi du langage Ada95 une justification supplémentaire car celui-ci offre un multi-tâches (ou tasking) préemptif. De même les listes comme celle d'Ada France permettent d'être orienté vers des solutions assez rapidement.

Ces critères de flexibilité, de coûts et fiabilité que l'on espère être satisfaits par les choix de Linux et Ada dans le cadre d'une conception HOOD seront à évaluer tout le long de la chaîne de développement logiciel.

2.2. UN SYSTEME VU EN COUCHES

Dans ce contexte, je vais essayer de dégager quelques idées simples de la philosophie de l'informatique embarquée dans les nacelles. Tout d'abord, on peut faire des considérations d'architecture d'ordre général.
Chaque entité du monde réel peut être associée à une entité logicielle à la manière de l'architecture de référence définie par l'ISO (International Standard Organization), architecture en couches de modèle OSI (Open System Interconnection) pour les réseaux. La figure 1 ci-dessous présente cette vue en couches.

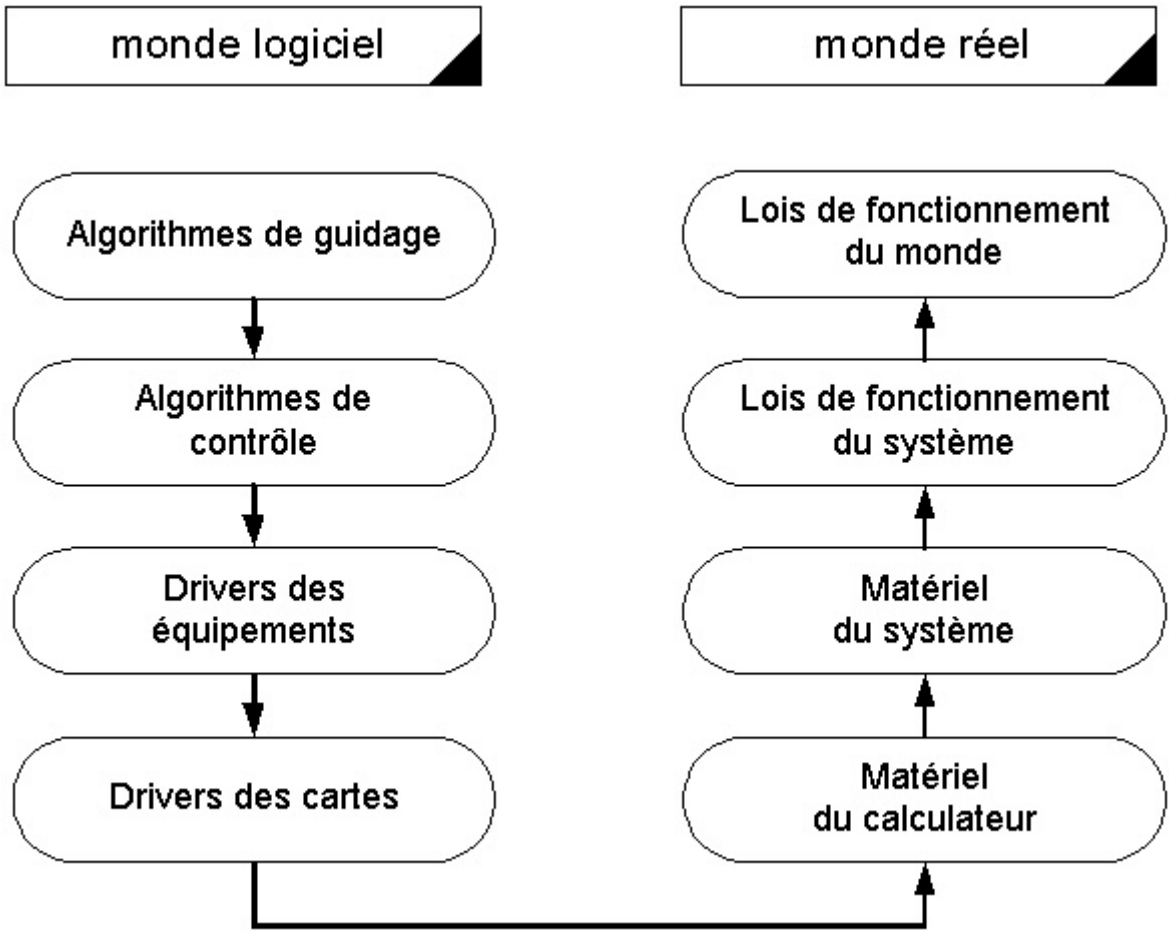

fig1- Système vu en couches

Au matériel lié au calculateur comme les cartes d'entrées/sorties, on peut associer les drivers de ces cartes.

Au dessus de ces cartes se trouve le matériel dit « du système », comme les capteurs (gyromètre, magnétomètre, inclinomètre,….). On leur associe les drivers de ces équipements, qui encapsuleront entre autres leur fonction de transfert.

Au dessus se trouvent les lois de fonctionnement du système (par exemple pointer en azimut) auxquelles on peut associer les composants logiciels qui réalisent les lois de contrôle (asservissements).

Enfin, en haut de cette vue, on trouve les lois de fonctionnement du monde, régissant la position des objets célestes, auxquelles correspondront les composants logiciels du guidage, permettant grâce au calcul astronomique et aux informations de localisation et de temps, de générer les consignes de pointage.

2.3. UNE ARCHITECTURE REPARTIE

L'architecture actuelle fait apparaître une séparation des segments bord et sol, qui communiquent à travers des liaisons de transmission numériques grâce au système « ETNA ». Ces liaisons se présentent comme des voies séries RS232 multiplexées dans un canal radiofréquence. Qu'elles soient bord→sol (télémesure) ou sol→bord (télécommande), elles acheminent les informations, chaque voie pouvant avoir un rythme de bit qui lui est propre.
Le canal physique ne sachant acheminer que des trains d'octets indépendants, il appartient à chaque client du système « ETNA » d'implémenter un protocole pour réaliser un protocole de bout en bout. A titre indicatif, dans le cas précis des nacelles pointées, les TM/TC sont envoyées par paquets.

De plus, dans le système actuel, dont j'ai représenté une vue globale sur la figure2, les deux calculateurs ne communiquent pratiquement pas entre eux et certaines fonctions doivent être obligatoirement dupliquées sur les deux calculateurs.

Des trames TM et TC sont utilisées par les deux calculateurs, et chacun possède des répliques des programmes permettant de traiter les TM/TC, à quelques fonctionnalités près. Les liaisons RS232 entre les 2 calculateurs et le système « ETNA » étant multiplexées pour laisser passer des trames TM ou TC, le port de liaison est limité par la voie dont le rythme de bit est le plus faible : les voies TM sont limitées à 38400 bauds et les voies TC à 9600 bauds. Dans ce cas, la vitesse de transmission sur la liaison Module de Contrôle – système Etna est donc limitée à 9600 bauds.

Il serait intéressant de pouvoir disposer des deux vitesses de transmission différentes pour exploiter le lien à la vitesse de transmission fournie par les voies et de pouvoir centraliser le traitement des TM/TC sur un seul calculateur de façon à ne plus dupliquer les programmes par exemple. Dans cette optique, l'autre calculateur à bord communiquerait avec ce dernier pour traiter des informations de TM/TC.

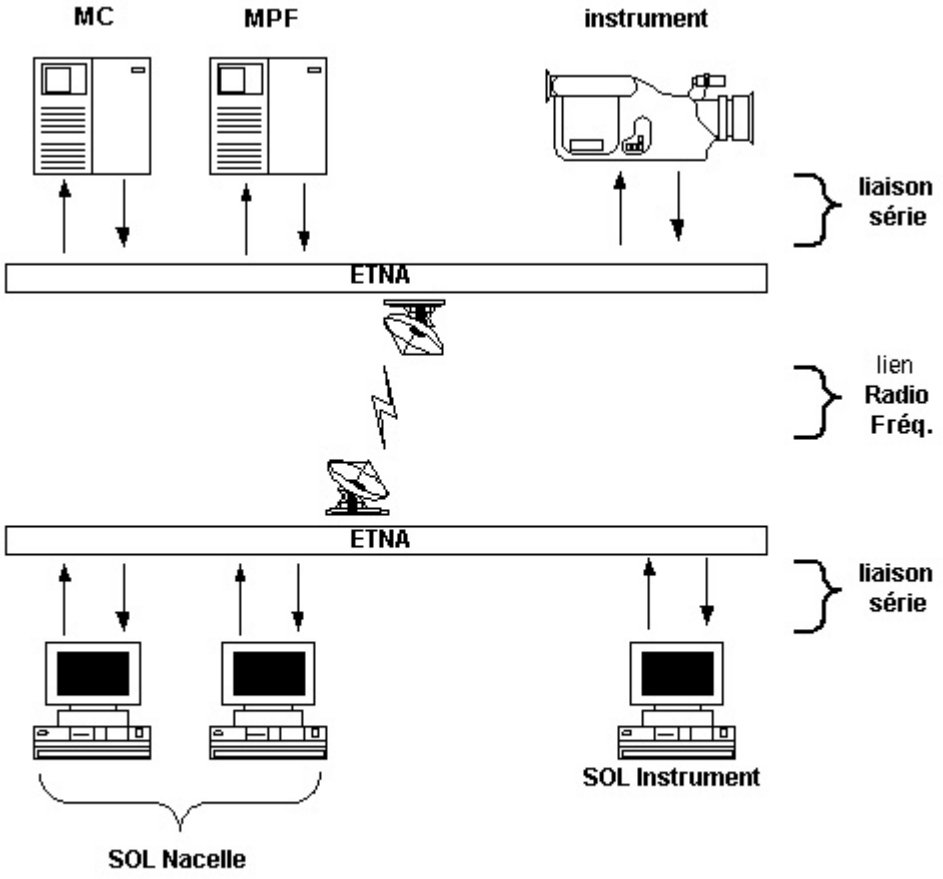

fig2- Architecture bord-sol actuelle

L'utilisation conjointe de machines dotées en standard de contrôleurs réseau et du système d'exploitation Linux disposant de toutes les couches logicielles du réseau associées, ouvre la porte à la mise en place d'architectures bord basées sur la communication entre calculateurs via un réseau local de type Ethernet utilisant le protocole IP. A terme, on pourra donc faire communiquer les calculateurs à bord entre eux par le réseau local (LAN, Local Area Network) et il est aussi envisagé de faire communiquer le segment bord et le segment sol (organisé lui aussi sur un LAN) avec une architecture IP comme indiqué dans la figure3. Le segment sol aurait alors essentiellement pour rôle la présentation des informations recueillies en cours de vol.

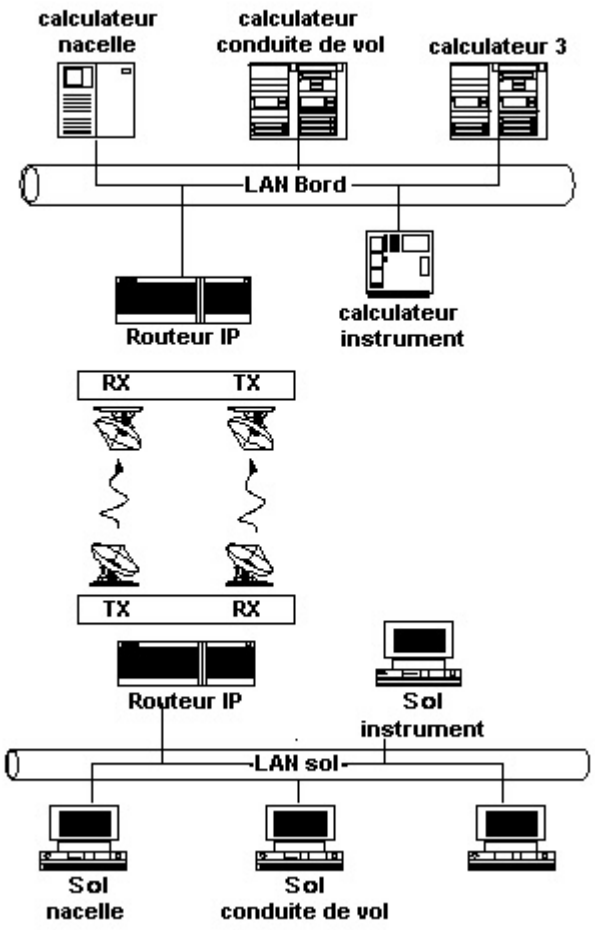

fig3- Architecture IP bord-sol

Dès lors, l'utilisation du gestionnaire de répartition GNAT-GLADE permet de réaliser des logiciels bords répartis avec une grande souplesse de distribution des fonctions. L'idée est de développer une application comme si elle devait s'exécuter sur un seul calculateur, puis de répartir les fonctions sur les nœuds en prenant en compte des considérations purement applicatives et ce, sans modifier le code. Nous présenterons de manière plus détaillée le modèle de répartition de l'annexe des systèmes distribués de la norme du langage Ada95 et son implémentation par GNAT-GLADE dans la partie 3.3 de ce document.

- ***Pourquoi répartir le traitement ?***

En première approche, on peut considérer qu'il est logique d'affecter à un calculateur les paquetages qui lui permettent de dialoguer avec les cartes qui sont connectées directement sur son bus.

En pratique, la configuration matérielle est définie à partir du besoin en interfaces avec les équipements nécessaires à la mission : la capacité de connexion des cartes sur un bus d'un calculateur de type PC104 est en théorie limitée à cinq cartes. En conséquence, la solution à cette limitation serait le simple ajout de calculateurs reliés entre eux à travers le réseau local bord, ce qui permettra de connecter les cartes supplémentaires devant être connectées.

Dans ce cas, il faut imaginer un mécanisme permettant de relier les entités logicielles localisées physiquement sur un seul calculateur et permettant l'accès à ces cartes, avec le calculateur approprié. La répartition de fonctions ou de modules logiciels sur l'architecture répartie ne sera pas faite au hasard.
Parmi les critères de répartition à utiliser, on peut citer :

- Le couplage par les flux de données : il serait maladroit de localiser sur des calculateurs différents des entités qui échangent une grande quantité de données, ou qui ont des échanges de données fréquents ;
- Le couplage par les flux de contrôle : il serait maladroit de localiser sur des machines différentes, des entités qui requièrent l'une de l'autre des services de façon très fréquente ;
- Le couplage par le matériel : il serait maladroit (voire impossible) de localiser le driver d'un équipement ou d'une carte d'entrées/sorties sur un autre calculateur que celui où est connecté l'équipement ou la carte ;
- La charge CPU : la gestion des marges de ressources CPU peut être facilitée par une répartition judicieuse des traitements sur les différents calculateurs.

En résumé, quelles fonctions répartir et sur quels calculateurs, et pour quelle mission ?

Une autre question s'est posée dans la logique de l'architecture nouvelle à atteindre.
Où vaut-il mieux faire le plus gros du travail de calcul, à bord ou au sol ?
Prenons l'exemple du calcul astronomique. Actuellement, le calcul de la consigne en azimut et élévation d'un appareil devant pointer sur un objet céleste met en œuvre un mécanisme coûteux en temps. En effet, le calcul de la position de l'objet céleste est fait au sol, après avoir récupéré les informations de localisation et de temps délivrées par le GPS, la consigne générée au sol est envoyée à bord par une télécommande.
Ce calcul nécessitant un minimum de puissance, les anciens calculateurs embarqués ne permettaient pas de le faire à bord. Désormais, les PC104 dont nous disposons possèdent un co-processeur et font partie de la famille des processeurs 486 et de plus fonctionnent à 133MHz.
Le fait d'embarquer le calcul astronomique permettra de s'affranchir de ces délais de transmission qui sont source d'erreur dans le pointage, de libérer la liaison bord-sol, et de faire d'autres choses à bord qui utilisent le calcul astronomique.

- ***Comment répartir ?***

Deux approches semblent possibles :

La première consisterait à partir d'une architecture matérielle et à venir « coller » l'architecture logicielle associée dessus.

Cette première approche n'est pas réaliste compte tenu du fait que les expérimentations testées avec les nacelles seront potentiellement amenées à utiliser des calculateurs changeant en nombre et en cartes d'entrées/sorties connectées.

La deuxième consisterait plutôt à commencer par concevoir le logiciel sans considérer la distribution. L'avantage de cette technique , c'est qu'en pratique, l'architecture matérielle n'est pas entièrement définie en début de projet et surtout qu'elle peut être amenée à changer rapidement selon les missions.

La distribution est à définir dans l'architecture pendant la conception et non dans la phase de spécifications.

3. TRAVAIL EFFECTUE

3.1. INSTALLATION DE LOGICIELS

Outre la prise de contact avec le sujet et le contexte, il m'a fallu au début de ce stage mettre à jour un certain nombre de logiciels et procéder à des installations de logiciels que j'utilise ou qui seront utiles dans un futur proche. C'est une tâche qui pourrait sembler immédiate, mais la stabilité d'une version de logiciel n'est atteinte qu'au prix de la modification de détails dans les fichiers de configuration, et dans les droits d'exécution de fichiers et répertoires, mise en évidence des bogues du logiciel, détails qui appellent la mise à jour d'autres versions de logiciels…

Parmi les mises à jour, je citerai le version 8.1 de la distribution Mandrake de Linux. Je conseillerais au prochain utilisateur de ce poste de développement de privilégier l'administration de logiciels par paquetages RPM (RedHat Package Manager), qui ont l'avantage de regrouper les fichiers résultants de la compilation et de l'installation complète du produit. Il est facile d'installer/désinstaller avec le gestionnaire de paquetages de la Mandrake, qui permet de savoir quels sont les paquetages utilisés par le paquetage installé, et la connaissance de ces dépendances s'avère fort utile dans la recherche des bonnes versions.

J'ai aussi mis à jour la version de STOOD qui pour le moment se trouve être la 4.2.55 (version pour HOOD4 et Hard Real Time HOOD). Elle a subi quelques « retouches » manuelles sous les conseils avisés de l'assistance technique de STOOD pour régler des bogues liés à la version. Cependant, la rétro-conception de programmes Ada vers des objets HOOD reste encore à tester.

J'ai installé le mode Ada pour Emacs et le compilateur GNAT 3.13, incluant l'utilitaire GNAT-GLADE, implémentation de l'annexe des systèmes distribués (DSA, Distributed Systems Annex) pour GNAT.
Voici sur quoi reposent les outils de création d'interfaces graphiques en Ada :
ActiveTcl 8.3.4.2., Tash 8.3.2 (Tcl Ada Shell) et Rapid 3.0 qui est l'outil de création d'interfaces portables en Ada.

3.2. TEST MINIMAL ADA + LINUX

La première chose à accomplir durant mon stage était de tester une application simple Ada95 s'exécutant sur les PC104. Mais simple ne veut pas dire qu'elle ne soit pas représentative des tâches s'exécutant sur les calculateurs lors d'un vol sur un calculateur. Comment alors se rapprocher le plus de la réalité ?
Il faudra d'abord attribuer aux tâches des caractéristiques leur conférant un comportement concurrent, et qui permet de vérifier que leurs périodes d'activation sont bien respectées dans ce contexte multi-tâches.

On peut associer à une tâche deux valeurs : un temps de cycle (ou période d'activation) et un temps de retard (temps entre l'activation de la tâche par ses entrées et l'activation

des sorties fournies par la tâche) car en pratique l'exécution d'une tâche n'est pas instantanée.
André Laurens m'a fourni les programmes disponibles dans l'annexe 2, avec lesquels je devais :

- Identifier la durée d'une boucle de calcul ordinaire consommant du temps processeur (calcul d'un sinus) sur le PC104.
- tester la coexistence de plusieurs tâches, de périodes d'activation définies sur la base de ce temps de boucle.

La chaîne de test consiste à compiler les fichiers de test sur le PC de développement (test_taches.adb), puis à transférer par ftp le fichier exécutable sur le PC104. Les résultats écrits au fur et à mesure dans un fichier de log, sont récupérés sur le PC de développement puis remis en forme par le programme défini dans depouille_taches.adb. Un exemple de fichier mis en forme par ce petit programme figure dans l'annexe 3.

3.2.1. Détermination de la durée d'une boucle de calcul

Ce temps est déterminé avec test_boucles.adb. On calcule le temps total de calcul pour N tours de boucle ainsi que la durée moyenne d'une boucle.

L'exécutable est transféré sur le PC104 de façon à pouvoir exécuter le programme de test directement sur la carte cible en s'affranchissant des temps supplémentaires engendrés par une connexion à distance du type telnet.

On observe qu'il faut tourner quelques milliers de boucles pour que le temps moyen d'une boucle converge vers une valeur stable. Les valeurs de stabilité sont approximées avec un nombre de boucles de 200000.

Voici les résultats obtenus :

Pour le PC104 : $100{,}8\ 10^{-6}$ s
Pour le PC de développement : $4{,}5\ 10^{-6}$ s

Ce temps va pouvoir maintenant être intégré dans l'exécution des tâches. Il suffit de le spécifier dans le fichier tache-cyclique.adb en affectant la bonne valeur pour la variable Duree_Du_Tour.

3.2.2. Test avec 6 tâches

Le choix a été fait d'effectuer un premier test avec 6 tâches cycliques.
Chaque tâche est caractérisée dans le programme par un nom, une priorité, une période et se découpe en 4 phases, chacune d'entre elles ayant pour fonction d'occuper le processeur pour une durée exacte. On distingue donc :

- une phase de lecture ;
- une phase de calcul ;
- une phase de télémesure ;
- une phase de commande.

Les phases de lecture, calcul, et commande sont exprimées directement avec un temps en secondes, et la phase de télémesure est exprimée sous forme de la taille du paquet TM émis. Les priorités les plus fortes sont dans la convention Ada95 celles de valeur les plus grandes, et les temps sont exprimés en secondes.
Les valeurs choisies n'ont ici pas vraiment d'importance, et ne sont pas représentatives d'un comportement en vol. Voici rassemblées les caractéristiques de ces tâches dans le tableau suivant :

	Tâche1	Tâche2	Tâche3	Tâche4	Tâche5	Tâche6
Priorité	10	5	2	10	5	2
Période	0.1	1	3	0.1	1	3
Durée phase Lecture	0.03	0.1	0.2	0.3	0.4	0.5
Durée phase Calcul	0.05	0.02	1	0.05	0.2	1
Flot phase Tm	100	200	20	100	200	20
Durée phase Commande	0.1	0.1	**0**	0.3	0.4	0.5

- ***1er test***

A la lecture des fichiers de logs, la tâche 2 démarre après la tâche 1, ce qui n'est pas conforme à l'ordre des appels d'entrée aux tâches 1 et 2.
La solution a été d'inclure l'enregistrement des temps de début et de fin de tâche dans l'accept (instruction Ada95) de l'entrée « Demarrer » dans le corps de la tâche contrôle (dans le fichier tache-cyclique.adb), tâche générique permettant d'instancier toutes les tâches créées.

```
accept demarrer do
  enregistrer
end accept
```

- ***Remarques sur les temps***

*Les périodes sont globalement respectées : on mesure 1.01s au lieu de la seconde spécifiée à l'instanciation, soit une valeur légèrement supérieure de quelques pour cent. Ceci est acceptable.

*Les durées de calcul ne ressemblent en rien aux valeurs attendues, nous supposons que l'optimiseur doit intervenir et fausser ces valeurs. Il faudrait pouvoir faire varier le calcul à chaque tour de boucle pour avoir des valeurs plus réalistes.

*La tâche 3 pour laquelle on a un delay(0), fait apparaître un temps de 3.5 10^{-4} s pour la phase de commande, ce qui doit correspondre peu ou prou au temps pris par le Put_Line de la phase de commande.

*Les durées de lecture et de commande sont assez « inattendues», car le temps passé correspond juste à une instruction de « delay » et il ne devrait pas y avoir de différence entre la durée prévue et la durée effective. Ici, on obtient presque le double de la valeur passée à l'instanciation du paquetage tache-cyclique. On peut penser que le temps de mesure du temps et le temps passé à l'enregistrement sur le log participent à cet écart.

*Enfin, les durées prises par la phase de télémesure fluctuent énormément, et le temps pris par l'instruction put du paquetage text_io pour écrire à l'écran n'est peut-être pas représentatif du temps que prend réellement l'envoi d'octets sur une liaison série.

- ***2ième test avec la correction de l'accept***

On fait la même constatation, les durées de lecture et commande ont des valeurs beaucoup plus grandes que prévu. Par exemple, pour 0.01s attendu, on obtient 0.02s.
La durée des Put_Line estimée à 5.0 10^{-4} s ne suffit pas à expliquer cet écart.

J'ai donc voulu savoir si ce défaut de durée était lié à la plateforme utilisée mais j'ai effectivement constaté les mêmes écarts en exécutant test_taches sur le PC de développement.
J'ai entrepris un autre test en modifiant les durées de delay pour voir si cet écart est significatif quand on augmente la durée des delay.
tache1 : durée lecture 0.03 et durée phase de commande 0.1
tache2 : durée lecture 0.1 et durée phase de commande 0.1
tache3 : durée lecture 0.2 et durée phase de commande 0.2
tache4 : durée lecture 0.3 et durée phase de commande 0.3
tache5 : durée lecture 0.4 et durée phase de commande 0.4
tache6 : durée lecture 0.5 et durée phase de commande 0.5

Sur le PC de développement, on constate que les durées de lecture et commande ont en moyenne augmenté toujours du même temps (9 ou 10 ms).
Sur le PC104, il y a toujours 9 à 10 ms d'écart pour la phase de lecture, et un peu moins pour la phase de commande. C'est cet exemple dont les fichiers de logs ont été dépouillés qui figure en annexe 2.

A partir de maintenant, nous négligerons cet écart, mais nous aurions pu nous poser la question suivante : Jusqu'à quelle limite inférieure de temps de delay (ou plus généralement de temps d'activation d'une tâche) on peut considérer que les écarts de temps par rapport aux valeurs escomptées sont insignifiants ?

3.2.3. Test avec 12 tâches

L'objectif du test est charger le calculateur, et d'observer le fonctionnement avec plusieurs tâches en considérant des durées réelles, représentatives des durées des boucles de contrôle et commande en vol.
Je me suis surtout attaché à vérifier que les périodes d'activation des tâches sont bien respectées.

Faisons le test avec 12 tâches (1 tâche TM ,1 tâche TC, 4 tâches de pilotage, 4 tâches représentant les servitudes, 2 tâches de calculs astronomiques) dont les caractéristiques sont rassemblées dans le tableau suivant.

	TC	TM	Pil_1	Pil_2	Pil_3	Pil_4	Serv_1	Serv_2	Serv_3	Serv_4	Astro_1	Astro_2
Priorité	5	1	10	10	10	10	2	2	2	2	5	5
Période	0.1	0.1	0.1	0.1	0.1	0.1	**30**	**10**	**1**	**1**	**1**	**0.1**
Durée phase Lecture	0.02	**0**	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.03	**0**	**0**
Durée phase Calcul	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.1	0.1
Flot Phase Tm	20	250	100	100	100	100	300	300	300	300	100	100
Durée phase Commande	**0**	0.065	0.01	0.01	0.01	0.01	**0**	**0**	**0**	**0**	**0**	**0**

J'ai laissé tourner ce test pendant cinq minutes.

- ***Commentaires sur les choix des priorités et des périodes***

<u>Taches dans l'ordre décroissant des priorités :</u>
pilotage > calcul astro = TC > servitudes > TM

Le plus important durant le vol est d'assurer le pilotage de la nacelle, de pouvoir la repérer à tout instant dans un repère horizontal local grâce aux mesures de temps et de localisation, et pouvoir lui envoyer des télécommandes. Enfin on considère les servitudes thermiques et électriques comme moins prioritaires, et avec des périodes plus grandes ainsi que les TM. En effet, il n'est pas vital de recevoir toutes les TM.

<u>Choix des périodes :</u>
Les taches de TC, TM et pilotage ont des périodes de 0.1 s. Et l'on choisit des périodes différentes pour les servitudes (exemple servitude thermique de 30 s, servitude électrique de 1 s)
Pour le calcul astronomique, qui n'a ni phase de lecture ni phase de commande/actuation, on prend 2 périodes différentes 0.1 et 1.

- ***Observations***

On constate que les périodes sont respectées aussi bien sur le PC de développement que sur les PC 104. On observe toujours les même écarts par rapport à la valeur passée à l'instanciation du paquetage pour les phases de calcul.

- ***Extensions possibles aux tests***

On aurait pu créer des tâches apériodiques se chargeant de lancer des tâches périodiques par exemple ou en imaginant une interface simple permettant à l'utilisateur de lancer une tâche apériodique pendant l'exécution du test. Mais le comportement essentiel à vérifier à ce stade de la conception du logiciel de vol était le respect des fréquences d'activation des tâches.

Il conviendrait d'utiliser un outil de trace non intrusif comme sur l'expérience Pronaos. Pour cette expérience, avait été utilisé un outil maison CNES très simple et très commode pour tracer l'activité et le comportement temporel du logiciel de vol, ce qui permettait de visualiser des chronogrammes d'activation de tâches, des statistiques de charge CPU, etc,...
Cet outil était basé sur :

- une instrumentation très légère du noyau multitâche et de l'application, qui écrivaient des informations à une adresse mémoire donnée ;
- une sonde connectée au bus du calculateur et qui interceptait les accès à cette adresse, les datait à la microseconde, les enregistrait, et les servait sur demande à un PC de dépouillement.

L'idée est de développer quelque chose de similaire pour nos besoins, et utilisant la périphérie standard des PC104. L'outil serait basé sur l'utilisation du port parallèle puisqu'on n'utilise pas d'imprimante sur les nacelles.

L'instrumentation écrit quelques octets, toujours le même nombre, sur le port parallèle, accompagnés de quelques signaux utiles (un signal de début de message, et si besoin un signal par octet transmis)

- Un boîtier électronique éventuellement numérique avec un calculateur traditionnel inclus pour des besoins de rapidité, date très précisément le signal de début de message, acquiert les octets, et les enregistre en mémoire avec la datation, et ce jusqu'à arrêt de l'acquisition ou saturation de la mémoire.
- Sur demande, le boîtier vide les données sur une liaison série à destination d'un PC qui les dépouillera.

En attendant mieux, cet outil sera prototypé par un PC standard permettant de simuler le fonctionnement de cette sonde. Ce développement a été entrepris par André Laurens.

- ***Conclusions générales du test Ada+Linux***

Globalement les périodes sont respectées, et nous avons mis en évidence des sources d'imprécisions temporelles, dues soit au temps pris par les entrées/sorties (Ada.Text_Io) utilisées comme moyen de trace, soit à l'optimiseur (pour les phases de calcul), ce dernier point pouvant être vérifié simplement.
Cependant on peut dire que ces tests temporels devront être confirmés lorsque les tâches tournant sur les PC104 seront des vraies tâches, avec des vrais calculs comme les calculs astronomiques qui occuperont le CPU de manière réelle.

3.3. ETUDE DU MODELE DE REPARTITION DU LANGAGE ADA95

La deuxième partie de mon stage consistait à évaluer le modèle de répartition du langage Ada95 pour notre application.

Cette partie s'appuie sur le travail réalisé par un des développeurs de l'annexe des systèmes répartis d'Ada95 (DSA) pour le compilateur GNAT [Pautet,2001]. J'énoncerai d'abord rapidement les entités d'Ada95 indispensables à la compréhension de DSA pour qu'un utilisateur de cet outil sache sur quelles notions du langage repose DSA. Je présenterai par la suite les fonctionnalités qu'offre DSA ainsi que l'architecture d'une application répartie Ada95. Enfin j'aborderai les tests effectués sur la base d'une répartition entre les PC104 et mon poste de développement grâce aux exemples fournis avec GNAT-GLADE.

Tout d'abord, essayons de répondre à cette question : Qu'est-ce qui caractérise une application distribuée en Ada95?
Au sens général, une application distribuée est caractérisée par plusieurs processus communicants, chacun s'exécutant potentiellement sur différents ordinateurs, les ordinateurs étant connectés à un réseau. Il s'agit de déterminer comment les processus distribués communiquent au niveau programmation et comment les petites parties de logiciel d'une application distribuée peuvent interagir.
Il y a plusieurs façons de distribuer une application : en utilisant les services réseau d'un système d'exploitation, ou en utilisant un environnement de type middleware, ou enfin en utilisant un langage distribué.
C'est dans ce dernier cadre que le langage de programmation Ada95 a été étendu par les développeurs de GNAT-GLADE avec des caractéristiques de distribution.

Définissons maintenant les entités fondamentales d'Ada95 nécessaires à la compréhension des mécanismes de répartition :

- unités de bibliothèques : un paquetage regroupe un ensemble logique d'entités du langage. Une unité de bibliothèque est un paquetage ou un sous-programme pris comme unité d'abstraction de niveau le plus haut. Notamment, elle n'est pas imbriquée dans une partie déclarative. Les paquetages génériques ainsi que leurs instances, peuvent également constituer des unités de bibliothèque. L'unité de bibliothèque constitue l'élément de granularité de la répartition. En effet, certaines de ces unités de bibliothèque vont doter les entités présentes dans la partie déclarative d'un comportement réparti ;
- types accès et types accès généralisé (rôle important pour la définition de références sur entités réparties) ;
- types limités et privés ;
- types étiquetés (tagged type) : ils interviennent lors de la définition d'objets répartis ;
- types tâches (flots de contrôle s'exécutant indépendamment des autres, sauf en cas de synchronisation explicite) : ils permettent de traiter plusieurs appels distants simultanément ;
- types protégés (destinés à la gestion des accès concurrents) ;

- attributs ‘Read et ‘Write : les types dérivés du type abstrait Root_Stream_Type permettent d’emballer ou de déballer des objets dans un flot de données. Ils permettent par exemple de stocker un objet dans un fichier. Dans le cas spécifique de la répartition, le flot de données correspond à un message.
- pragma : c’est une directive de compilation qui indique certaines propriétés au compilateur sans effet sémantique. Six pragmas, dits de catégorisation, imposent aux unités de bibliothèques des restrictions propres à un contexte réparti. Nous définirons ces pragmas plus tard.

3.3.1. Vers un sous-ensemble d’Ada95 pour la répartition

DSA relève d’une volonté d’intégrer la répartition dans le langage sans lui ajouter de nouvelles entités. Pour ne pas sortir de ce cadre, l’idée est de sélectionner les entités d’Ada95 susceptibles de donner lieu à un comportement réparti. Ensuite, l’usage de ces entités peut être restreint grâce à des pragmas afin de répondre aux contraintes de répartition. Y. Kermarrec, participant au projet de développement de DSA pour GNAT, constate que la norme énumère de manière exhaustive les restrictions imposées aux unités de bibliothèque sans toutefois les justifier, ni indiquer les entités réparties autorisées. L’approche de Laurent Pautet que je vais synthétiser ici, est une approche constructive, consistant à explorer l’ensemble du langage Ada95 pour dégager les entités susceptibles d’être réparties, en éclaircissant les motivations cachées derrière les restrictions imposées aux entités Ada95.

Notons simplement quelques propositions permettant de justifier par la suite la définition des pragmas de catégorisation :

DSA devant rendre aussi faible que possible la différence existant entre la version répartie et la version monolithique d’une même application, la localisation des entités réparties comme les nœuds logiques ou les **unités de bibliothèque**, doit s’effectuer de manière transparente pour l’utilisateur grâce à un service de nommage interne. Cependant, les systèmes temps réel répartis peuvent exiger un placement statique.

Pour participer à la répartition, un type doit disposer des propriétés d’un **type transportable**. Plus précisément, une donnée doit pouvoir être codée par l’émetteur et décodée par le récepteur pour représenter une donnée équivalente à la donnée originelle. En particulier, les paramètres de procédure à distance doivent pouvoir être transportés. Si les primitives de codage et décodage ne sont ni canoniques comme pour les types prédéfinis, ni spécifiques comme pour les références, l’utilisateur doit se charger de leur définition. Un type transportable se déclare dans une unité de bibliothèque catégorisée chargée d’appliquer ces restrictions.

Un sous-programme distant doit pouvoir être localisé sans ambiguïté et l’unité de bibliothèque catégorisée qui le déclare doit donc être unique. L’appel à un tel sous-programme distant se résout de manière statique.

La déréférence d’une variable de **type accès sur sous-programme** provoque l’exécution du sous-programme désigné. Dans le cas d’un type accès distant sur sous-

programme, cette déréférence correspond à l'appel d'un sous-programme distant résolu de manière dynamique.

Un objet réparti doit se protéger contre copies et accès. En conséquence, le type étiqueté qui le définit dans une unité de bibliothèque catégorisée doit être limité privé. Le type accès distant ou référence sur cet objet doit également être défini dans une unité de bibliothèque catégorisée et devient ainsi transportable. Les unités de bibliothèque impliquées dans ce dispositif peuvent être répliquées.

Un pragma doit permettre à un sous-programme sans paramètre de sortie d'une unité de bibliothèque catégorisée de s'exécuter de manière asynchrone. Ce même pragma peut s'appliquer à un type accès distant sur objets et rend automatiquement unidirectionnel tout appel à une méthode d'objet réparti n'ayant aucun paramètre de sortie.

Un autre pragma doit permettre à ce que tout appel à un sous-programme distant transite par le système de communication même si l'appel peut être résolu en local.

Un **objet partagé** doit pouvoir être défini dans une unité de bibliothèque catégorisée, unique et accessible depuis plusieurs nœuds logiques à travers un support partagé éventuellement réparti comme une mémoire, un système de fichiers ou une base de données.

Enfin, des règles de visibilité entre unités de bibliothèque doivent interdire des compositions d'objets de types incompatibles.

3.3.2. Architecture d'une application répartie dans le modèle

Après avoir énuméré les entités du langage Ada95 susceptibles de donner lieu à un comportement réparti, et les restrictions qui devaient les accompagner, nous allons montrer comment ces restrictions se traduisent dans la norme.
Je vais tout d'abord décrire l'architecture d'une application répartie et préciser la notion de nœud logique, puis je décrirai les pragmas de catégorisation qui déterminent les entités réparties dans les différentes unités de bibliothèque et enfin je présenterai succinctement l'interface du système de communication à laquelle une implémentation doit faire appel.

- ***Partitions actives et passives***

En Ada95, une partition correspond à un nœud logique, et se compose d'unités de bibliothèque catégorisées ou non. DSA définit deux types de partition :

- une partition active possède un ou plusieurs flots de contrôle, elle émet et reçoit des requêtes d'autres partitions actives. Elle peut contenir toutes les entités réparties présentées précédemment.
- Une partition passive ne contient que des objets partagés auxquels accèdent des partitions actives. A ce titre, elle ne dispose d'aucun flot de contrôle. Ses objets se trouvent définis dans un espace de stockage partagé.

Chaque partition possède un identificateur unique de partition. L'attribut 'Partition_Id appliqué à la partition donnera la valeur de cet identificateur sous la forme d'un entier.

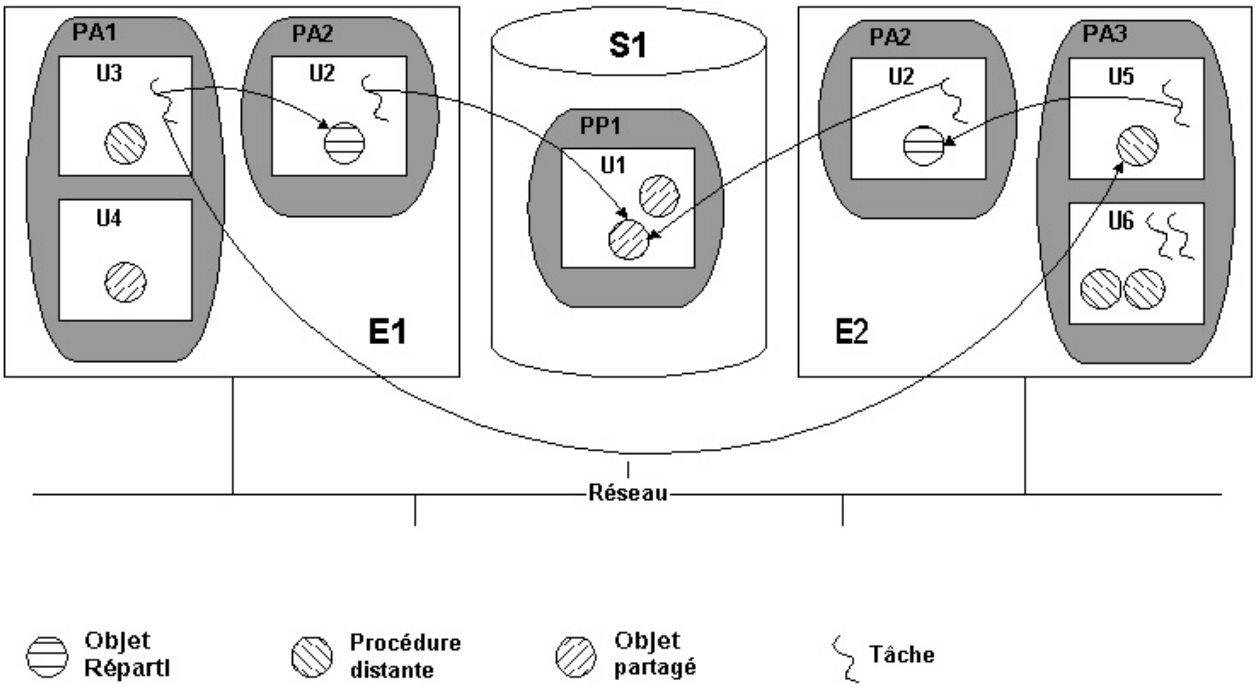

fig4- Exemple de partitionnement

Cette figure illustre l'architecture d'une application répartie selon le modèle DSA. Elle se compose de 2 machines ou supports d'exécution E1 et E2 reliées par le réseau et partageant un disque ou support de stockage commun S1. Seule une partition passive PP peut être configurée sur S1. Cette partition PP1 ne détient donc que des objets partagés qui se trouvent dans l'unité U1. Sur E1 s'exécutent deux partitions actives PA1 et PA2. La première contient une tâche définie par l'unité U3 ce qui fait d'elle une partition active. Cette tâche effectue des appels à des sous-programmes distants de l'unité U5 configurée sur la partition PA3 de la machine E2. La partition active PA1 peut également contenir des objets partagés comme celui défini dans l'unité U4. La partition PA2 ne contient qu'une unité U2 comportant des objets répartis. Aussi peut-elle avoir plusieurs instances, une sur E1 et une sur E2. Au contraire, l'unité U6 qui contient des sous-programmes distants doit être unique comme la partition PA3 qui la contient. Enfin, les instances de PA2 localisées sur E1 et E2 accèdent à un objet partagé au moyen de S1.

- ***Unités de bibliothèque catégorisées***

Seule une unité de bibliothèque catégorisée offre les services répartis proposés en 3.3.1.

L'unité de bibliothèque à laquelle s'applique le pragma **Remote_Types** ne contient, dans la partie publique de sa déclaration, que des types transportables. Elle peut définir des références sur entités distantes mais aussi dériver des types étiquetés susceptibles d'être désignés par ces références. Une telle unité peut être répliquée sur toutes les partitions sur lesquelles elle est référencée.

L'unité de bibliothèque à laquelle s'applique le pragma **Remote_Call_Interface** (RCI) peut contenir dans la partie publique de sa déclaration des types transportables comme des références sur entités distantes mais également des sous-programmes distants. Dès lors, une telle unité ne peut être répliquée. Cette restriction s'explique par la présence éventuelle de sous-programmes distants dont la localisation doit être statique.

L'unité de bibliothèque à laquelle s'applique le pragma **Shared_Passive** ne contient que des objets partagés et aucun flot de contrôle. Les variables définies dans la partie publique de sa déclaration sont accessibles depuis plusieurs partitions à travers un support partagé éventuellement réparti. Une telle unité ne peut être répliquée.

L'unité de bibliothèque à laquelle s'applique le pragma **Pure** est garantie sans effet de bord et ne conserve aucun état interne. Elle contient typiquement des définitions de types simples et les opérations primitives applicables sur ces types. Une telle unité de bibliothèque peut être répliquée sur toutes les partitions sur lesquelles elle est référencée. Ce type de catégorisation n'est pas propre à la répartition et se révèle utile dans d'autres contextes.

Une unité peut avoir visibilité sur les entités d'une autre unité de catégorie différente. Pour prévenir toute incohérence, tout pragma de catégorisation doit respecter la hiérarchie suivante entre les unités de bibliothèque :
Remote_Call_Interface > Remote_Types > Shared_Passive > Pure ,
où C1 > C2 signifie qu'une unité de bibliothèque de catégorie C1 peut avoir une visibilité sur une unité de bibliothèque de catégorie C2.
Toute unité de bibliothèque sans catégorisation est qualifiée de normale et peut être répliquée sur toutes les partitions sur lesquelles elle est référencée. Les unités définissant des entités statiques comme les unités de bibliothèque catégorisées RCI et Shared_Passive ne peuvent être répliquées à la différence des unités de bibliothèque sans catégorisation ou catégorisées Remote_Types et Pure.

En plus des quatre pragmas majeurs, deux pragmas mineurs permettent d'altérer l'exécution des appels de sous-programmes ou de méthodes à distance.
Le pragma **Asynchronous** s'applique à un sous-programme sans paramètres de sortie d'une unité de bibliothèque catégorisée RCI. Tout appel à ce sous-programme est unidirectionnel et toute levée d'exception est alors ignorée. Ce pragma s'applique aussi à un type d'accès distant sur entités réparties. Comme il s'applique aussi bien à des sous-programmes qu'à des références, le destinataire peut être déterminé aussi bien statiquement que dynamiquement. Il offre ainsi la possibilité de mettre en œuvre un mécanisme d'envoi de messages.

Le pragma **All_Calls_Remote** s'applique à une unité de bibliothèque catégorisée RCI de sorte que tout appel à un sous-programme distant de ce paquetage devra transiter

par le système de communication même si l'appel peut être résolu en local. Cette fonctionnalité s'avère utile lors de la mise au point de l'application alors qu'elle n'a pas encore été répartie puisque les latences induites par la communication ne sont pas omises.

- ***Sous-système de communication***

Ada95 définit l'interface à utiliser entre le compilateur et le système de communication de DSA. Le paquetage System.RPC définit types et sous-programmes à utiliser comme points d'entrée du système de communication. Ce dernier à pour rôle :

- d'envoyer les requêtes et recevoir les éventuelles réponses d'appels de sous-programmes ou de méthodes à distance auprès d'un serveur.
- de recevoir les requêtes et envoyer éventuellement les réponses de requêtes auprès du client, débloquant ainsi les tâches responsables des appels à distance.
- Analyser les requêtes du client chez le serveur pour en effectuer le traitement en déléguant éventuellement auprès de tâches Ada95 dédiées à cet effet.

Mais System.RPC ne propose aucun mécanisme de localisation des partitions et des unités de bibliothèque. Il apparaît que DSA s'adresse plus particulièrement aux applications temps réel statiquement réparties. Dans ce cas, chaque partition connaît la localisation prédéterminée des autres partitions ainsi que celle des unités de bibliothèque.

Où se situe ce sous-système de communication dans GNAT-GLADE, la mise en œuvre du modèle de répartition d'Ada95 ?

GNAT-GLADE s'organise autour de trois composants fondamentaux : GNAT, GARLIC, et GNATDIST.
Sans rentrer dans les détails qui pourront être compris par la lecture de [Pautet,2001], le compilateur GNAT génère les souches et squelettes des entités réparties de DSA.
Le système de communication entre partitions est appelé GARLIC pour Generic Ada95 Reusable Library for Inter-partition Communication, offre plusieurs services (gestion de partitions, gestion d'unités de bibliothèque, terminaison, annulation de requêtes, traitement de requêtes concurrentes, représentation de données).
Enfin GNATDIST est l'outil de partitionnement qui produit souches et squelettes grâce à GNAT. De plus, pour chaque partition, il relie les squelettes des unités de bibliothèque qui lui ont été affectées, les souches des autres unités de bibliothèque référencées et le système de communication afin de produire un exécutable.

Arrêtons nous un instant sur GARLIC qui forme l'interface entre la couche de communication Ada et le niveau réseau et précisons ce qui se cache derrière les termes utilisés de souches et de squelettes. GARLIC implémente et encapsule les protocoles utiles pour fournir la sémantique des RPC linux (Remote Procedure Call) . Les RPC sont transparents pour le programmeur, c'est à dire que les paramètres et résultats d'un appel distant sont passés par le réseau, sans son intervention. Comme cela a déjà été précisé, c'est le compilateur GNAT qui crée des souches (stub) du côté client d'un appel et des squelettes (skeleton) du côté serveur de cet appel. Le code de l'application, au

lieu d'appeler directement la procédure appellera ces souches et fera une opération dite de marshalling, sur les paramètres de la procédure et les passera à GARLIC.
En effet, pour envoyer les paramètres d'un appel distant à travers un réseau, ils doivent être "applatis", c'est-à-dire mis à plat dans un simple flot d'octets. Sur la partition réceptrice, ils doivent être reconstruits à partir de cette représentation linéaire. C'est cette opération qui est appelée marshalling et unmarshalling.

Voici la structure de GARLIC, selon [Kienzle,1997].

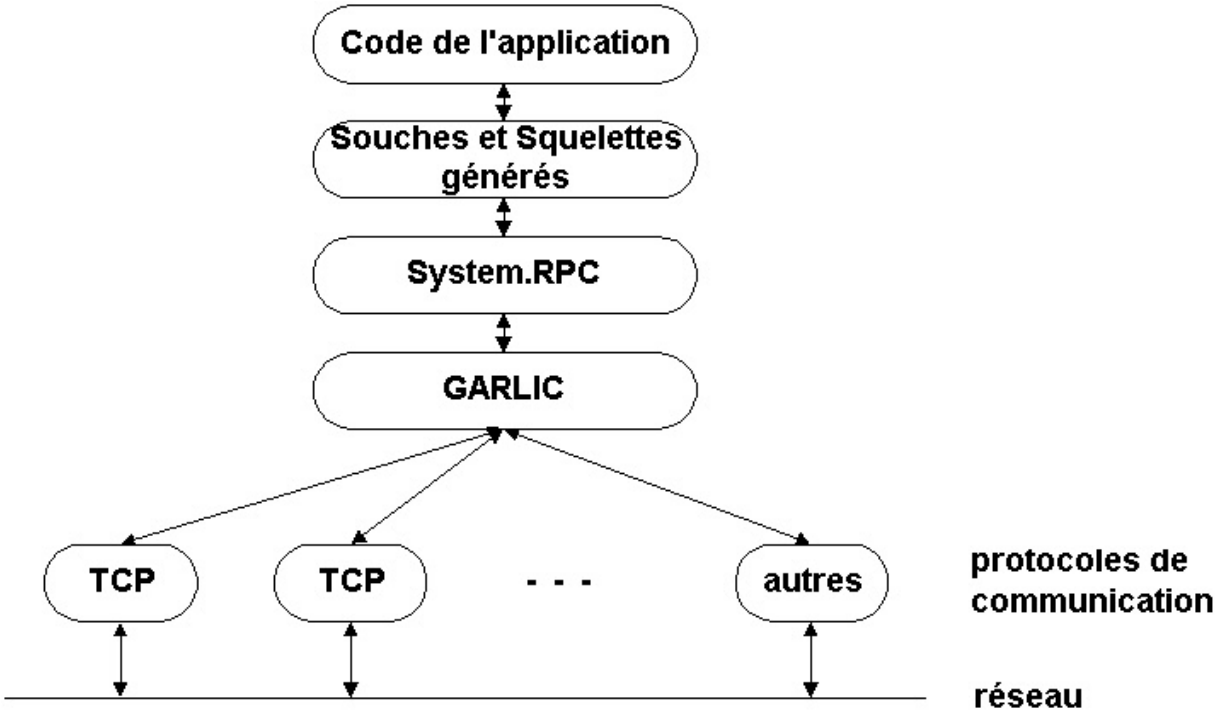

fig5- Structure de Garlic

GARLIC se charge donc de la gestion des appels de sous-programmes ou de méthodes à distance et constitue à cet égard, un ORB (Object Request Broker) pour DSA. En Ada95, pour construire une application monolithique, l'utilisateur doit en spécifier le sous-programme principal. Dans d'autres langages, il se nomme « main » par convention. A l'aide de cet identificateur, GNAT compile toutes les unités de l'application pour produire un exécutable du même nom que l'utilisateur peut invoquer pour exécuter le programme. Dans le souci de toujours minimiser les différences entre le développement d'un programme monolithique et de sa version répartie, GNAT-GLADE attribue un rôle particulier à la partition principale dont le sous-programme principal correspond à celui de l'application monolithique. De plus, GNAT-GLADE peut construire cette partition de sorte qu'elle se charge de lancer automatiquement les autres partitions pour l'utilisateur.

3.3.3. Tests effectués avec les exemples fournis par GNAT-GLADE

Mon travail consiste à faire marcher une application répartie entre plusieurs calculateurs (la machine de développement et les maquettes de calculateur embarqué). En

m'appuyant sur les exemples de démonstration fournis par l'outil GNAT-GLADE, ma démarche a été de commencer par faire d'abord marcher une application simple (du type client/serveur) et à lui rajouter des fonctionnalités au fur et à mesure.

- ***Processus de test***

L'annexe traitant des applications réparties ne décrit pas comment une application doit être configurée. Il appartient donc au programmeur de préciser quelles sont les partitions dans son application et sur quelles machines elles doivent être exécutées. L'outil gnatdist lit un fichier de configuration, écrit avec une syntaxe « Ada-like », qui permet de construire les divers exécutables, un pour chaque partition. Cette syntaxe est détaillée dans le manuel d'utilisateur de GNAT-GLADE [Pautet &Tardieu,2001].

Voici la démarche à adopter pour répartir une application sur plusieurs machines :
1. Ecrire une application non-répartie en précisant les pragmas de catégories pour spécifier les paquetages qui peuvent être appelés à distance (Remote_Call_Interface)
2. Ecrire le fichier de configuration qui repartit les paquetages dans les partitions en précisant le programme principal de l'application répartie.
3. Lancer la commande « gnatdist fichier_configuration <options>»
4. Enfin, avant de lancer l'application répartie en invoquant le programme principal, ou en invoquant le script-shell de lancement, positionner les variables d'environnement S_RPC et S_PARINT à true pour pouvoir tracer l'échange des RPC et avoir des informations sur l'état des partitions et des unités.

- ***Tests***

L'objectif principal de ces tests est de mettre en évidence l'utilité de certains mots clés du langage de configuration de gnatdist et d'éventuels disfonctionnements liés au processus de configuration. On cherchera à terme à faire en sorte que la partition principale se charge de lancer automatiquement les autres partitions, en utilisant le mode de lancement Starter-Ada qui n'est autre qu'un exécutable correspondant au script-shell généré dans le mode de lancement Starter-Shell.
En effet, dans le cadre de notre application pour le logiciel de vol, il est intéressant de pouvoir générer des exécutables qui seront placés physiquement sur un calculateur choisi, et sur la base de cette allocation physique, pouvoir lancer l'exécutable de la partition principale qui se chargera de lancer les autres partitions. Une des contraintes dans cette phase de démarrage de l'application est de s'assurer que toutes les partitions ont été lancées et sont prêtes à échanger des RPC.

- 1er test : lancement manuel des partitions

Dans l'ordre, je me propose de tester dans une première phase à l'aide de l'exemple BANK :
- la création de deux partitions sur la même machine et un échange du type client/serveur entre ces partitions, en limitant le test où le serveur et le client se trouvent chacun sur leur partition.

- la distribution de cette même application sur mon PC de développement et sur une seule maquette PC104.
- la distribution de cette même application sur mon PC de développement et sur les deux maquettes PC104.

Le fichier de configuration simcity.cfg de cet exemple correspondant aux cas testés figure en annexe 4.

Cette phase a pu être validée en lançant manuellement chaque exécutable préalablement copié sur chaque partition et en précisant dans la ligne de commande correspondante la localisation de la partition principale avec ce format :

Partition1 serveur sur pc-bana-diderot :
./bank_server --boot_server tcp://pc-bana-diderot.cst.cnes.fr:5557
Partition 2 client sur pc-bana-daurat :
./bank_client --boot_server tcp://pc-bana-diderot.cst.cnes.fr:5557
Partition 3 client sur pc-bana-saintex :
./bank_client --boot_server tcp://pc-bana-diderot.cst.cnes.fr:5557

Il est à noter que dans ce mode de lancement manuel (starter None), il n'est pas nécessaire de faire figurer dans le fichier de configuration le pragma Boot_Server.

- 2ième test : starter Shell et Ada

Le mode de lancement automatique (Starter-Ada) est celui qu'il reste à faire fonctionner dans un contexte réparti. A partir de l'exemple Eratho-spiral, dont le fichier de configuration figure en annexe5, j'ai pu tester que ce mode marche lorsque les partitions sont définies sur un même calculateur.

Ce mode demande, à l'exécution de la partition principale, les noms des hôtes sur lesquels les exécutables doivent se lancer. On peut sauter cette phase en précisant dans le fichier de configuration :

```
for Partition2'Host use "localhost";
for Partition3'Host use "localhost";
```

Dans le cas d'une répartition sur deux calculateurs, ce mode est resté inopérant.
Précisons ici qu'il y a deux façons d'assigner les partitions à des nœuds physiques. Comme cela a été dit plus haut, par défaut, l'allocation est dynamique et c'est la partition principale qui demande le nom de l'hôte à l'exécution. Ici, j'ai testé le cas d'une allocation statique, où le nom de l'hôte est défini dans le fichier de configuration comme suit :

```
for Partition2'Host use "pc-bana-daurat.cst.cnes.fr";
for Partition3'Host use "localhost";
```

On teste ainsi l'allocation physique suivante :

partition 1 et 3 : pc-bana-diderot
partition 2 : pc-bana-daurat

Pour débugger ce mode, je suis remonté au mode Starter-Shell et j'ai constaté que le script généré par l'outil GNAT-GLADE comportait des incohérences.

- Pour une partition donnée, si on précise dans le fichier de configuration, « for Partition3'Host use "localhost"», ceci est traduit en un rsh sur le localhost. Ce n'est pas utile.

- un rsh lance une commande dont l'exécutable est physiquement sur la machine distante. Or la commande (ici l'exécutable de la partition) est considérée comme étant sur la machine depuis laquelle le rsh est lancé.

J'ai passé assez de temps à essayer de faire marcher l'outil GNAT-GLADE uniquement à l'aide des exemples fournis, ce qui m'a amené à faire des modifications sur les machines daurat et diderot dans les fichiers /etc/hosts, /etc/hosts.allow et /etc/hosts.deny, qui sont à reproduire sur la machine saintex.

- ***Idées et perspectives***

La première chose qui devrait être fixée doit être la mise au point de l'outil GNAT-GLADE pour que les scripts générés à partir du fichier de configuration marchent.

Une solution possible de stockage des exécutables pendant la mise au point de l'application serait envisageable en utilisant les services NFS. Elle permettrait de stocker sur un seul calculateur tous les exécutables, et moyennant quelques modifications dans les fichiers gérant la configuration NFS, l'ensemble des calculateurs connectés sur le réseau local pourrait monter le répertoire dans lequel seraient stockés tous les exécutables. Un calculateur distant pourrait ainsi avoir accès à l'exécutable comme s'il se trouvait dans sa propre arborescence de fichiers.

3.4. ETUDE DE L'INTEGRATION DES CONCEPTS DE REPARTITION DANS UNE CONCEPTION HOOD.

Cette dernière partie pose les bases d'un travail sur la correspondance que l'on peut faire entre des partitions au sens GNAT-GLADE et les nœuds virtuels HOOD. Ce travail est né du besoin d'intégrer les concepts de répartition dans une conception HOOD. Nous rappellerons quelques propriétés de l'approche de conception HOOD, nous présenterons le concept des nœuds virtuels et nous essaierons de présenter les différentes approches possibles pour le travail restant à faire.

3.4.1. L'approche méthodologique de conception HOOD pour les systèmes complexes

- ***HOOD***

HOOD 4 comme HOOD 3 offre au concepteur des possibilités de représentation des modèles à différents niveaux d'abstraction et de raffinements tout en gardant la cohérence avec les représentations initiales.

Dans le cadre du développement d'un système complexe, ces propriétés vont être exploitées pour produire des versions successives des modèles HOOD (maquettes, prototypes) jusqu'au système opérationnel avec une très bonne traçabilité et un haut niveau de réutilisation. Elles s'appuient sur la notion d'invariance des modèles initiaux et sur la notion de raffinements d'une description d'ODS.

La démarche de développement est basée sur les principes suivants :

- Elaboration d'un modèle initial /solution logique :

  Ce modèle est une abstraction de solution en termes d'objets : les éléments d'implémentation sont ignorés et le modèle est donc garant d'un indépendance de la solution vis-à-vis des systèmes cibles (langages, machines, systèmes répartis)

- Raffinement des modèles initiaux :

  Les modèles obtenus par raffinement sont plus détaillés, vérifiables et validables. Ils permettent de maîtriser la complexité tout en respectant les contraintes d'invariance énoncées ci-dessus.

- ***Particularités de l'outil STOOD et objectifs de l'étude***

STOOD permet de regrouper les applications en ensembles (systèmes) pouvant avoir une intersection non nulle (applications partagées).

Les applications HOOD sont accessibles à leur plus haut niveau sous forme d'un objet racine d'une arborescence de décomposition ("Root Object ").

Cet objet racine peut être de type " DESIGN " (application principale), "GENERIC" (anciennement appelé " CLASS " : modèle d'application) ou " VIRTUAL NODE " (schéma d'implémentation physique).

En séparant le modèle logique de conception (" design ") du modèle d'implémentation physique (" virtual node "), HOOD permet la conception (le design) de systèmes distribués sans laisser les aspects liés à la répartition bouleverser toute la conception. De façon simplifiée, car nous les définirons en détail plus tard, les nœuds virtuels ont été pensés pour pouvoir passer d'une conception logique HOOD à une implantation physique sur un environnement multi-processeurs.

Or, jusqu'à ce jour, l'outil STOOD n'implémente pas la génération de code pour des nœuds virtuels. Nous avons vu dans la partie précédente comment on pouvait distribuer un logiciel sur plusieurs partitions à l'aide de l'outil GNAT-GLADE. Un des objectifs de cette partie est d'étudier le moyen de considérer les partitions GLADE comme des nœuds virtuels dans une conception. L'idée est de définir de manière formelle les règles de correspondance entre partition et nœud virtuel pour que l'outil STOOD permette de générer de façon automatique les fichiers de configuration GLADE.

3.4.2. Les nœuds virtuels HOOD

- ***Concepts***

D'après le manuel de référence HOOD [HOOD User's Group,1995], cette méthode offre 3 concepts permettant de traiter les systèmes distribués:

- le nœud virtuel (Virtual Node, VN) : c'est un ensemble d'objets HOOD, qui est allouable à un processeur, et qui est graphiquement représenté par un objet avec la lettre V dans sa partie supérieure gauche comme sur la figure6 ci-dessous.

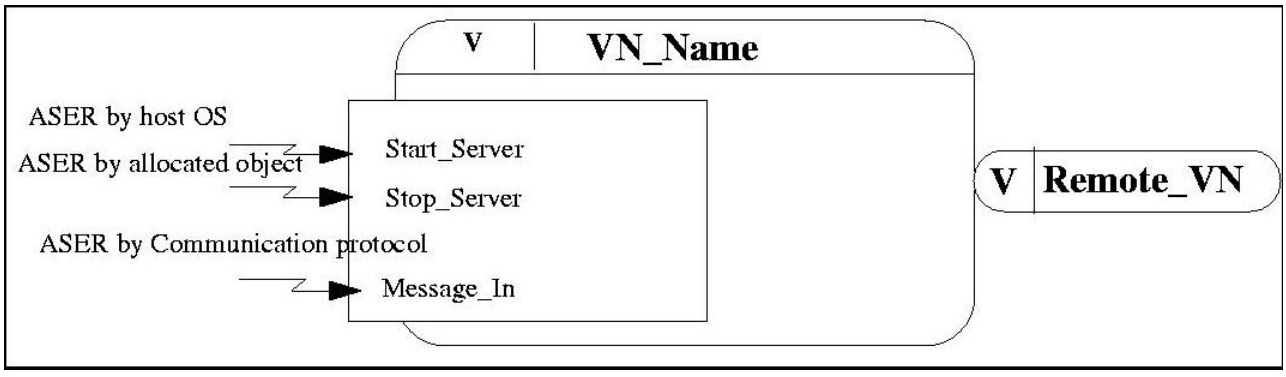

fig6- un nœud virtuel HOOD

- le partitionnement : processus de séparation d'un arbre de conception HOOD en un ensemble de nœuds virtuels.

- le nœud physique (Physical Node, PN) : c'est le modèle de conception HOOD d'un programme exécutable. Il résulte de la mise en correspondance (binding) entre au moins un nœud virtuel, des représentants d'objets situés dans des nœuds physiques distants (de tels représentants s'appellent des objets remplaçants,"surrogates"), et un "run-time environment" (RTE).

- ***Propriétés***

- un VN terminal est défini comme l'encapsulation d'objets HOOD, instances de classes et génériques alloués à ce VN.
- un VN peut avoir une ou plusieurs opérations prédéfinies "Message_In" qu'il fournit à l'extérieur, permettant de spécifier le protocole de communication avec le client VN à l'aide du label ASER.
- un VN terminal inclut un objet VNCS (Virtual Node Control Structure) prédéfini et les objets alloués. La représentation graphique d'un tel VN terminal est optionnelle, dès lors que toutes les représentations graphiques de VN terminaux devraient être similaires et une description textuelle des objets alloués est suffisante.
- un VN peut seulement utiliser d'autres VNs.
- un VN peut seulement être décomposé en d'autres VNs, ce qui définit une hiérarchie de VNs.

Cette caractéristique permet l'abstraction et le raffinement de gros programmes en des entités VNs maîtrisables et bien définies.

- ***Principes d'implémentation***

Il y a trois vues indépendantes dans un système distribué :

- la vue spatiale logique, consistant en un ensemble d'arbres de conception.
- une vue spatiale distribuée, qui définit des unités indivisibles de distribution, mais encore comme des entités logiques.
- une vue spatiale de nœuds physiques, qui définit des nœuds physiques en configurant les unités dites de distribution.

L'espace de distribution est modélisé par une hiérarchie de nœuds virtuels. Ils sont appelés nœuds parce qu'ils pourraient être des unités de distribution, mais ce sont des nœuds virtuels parce qu'ils ne correspondent pas nécessairement aux nœuds physiques. En réalité, plusieurs nœuds virtuels peuvent être implémentés sur un unique nœud physique.

Un nœud virtuel est soit terminal (et correspond à un exécutable s'il est implémenté par du logiciel), ou décomposé en nœuds virtuels enfants.

On peut représenter un projet comme l'organisation d'un ensemble d'espaces et de hiérarchies (voir figure7). Les design HOOD ou hiérarchies d'objets sont « branchées » sur l'espace des génériques, c'est à dire l'espace des modules réutilisables. Si l'application est distribuée, les différents nœuds sont projetés sur l'espace des nœuds virtuels, qui lui-même est projeté sur l'espace des nœuds physiques.

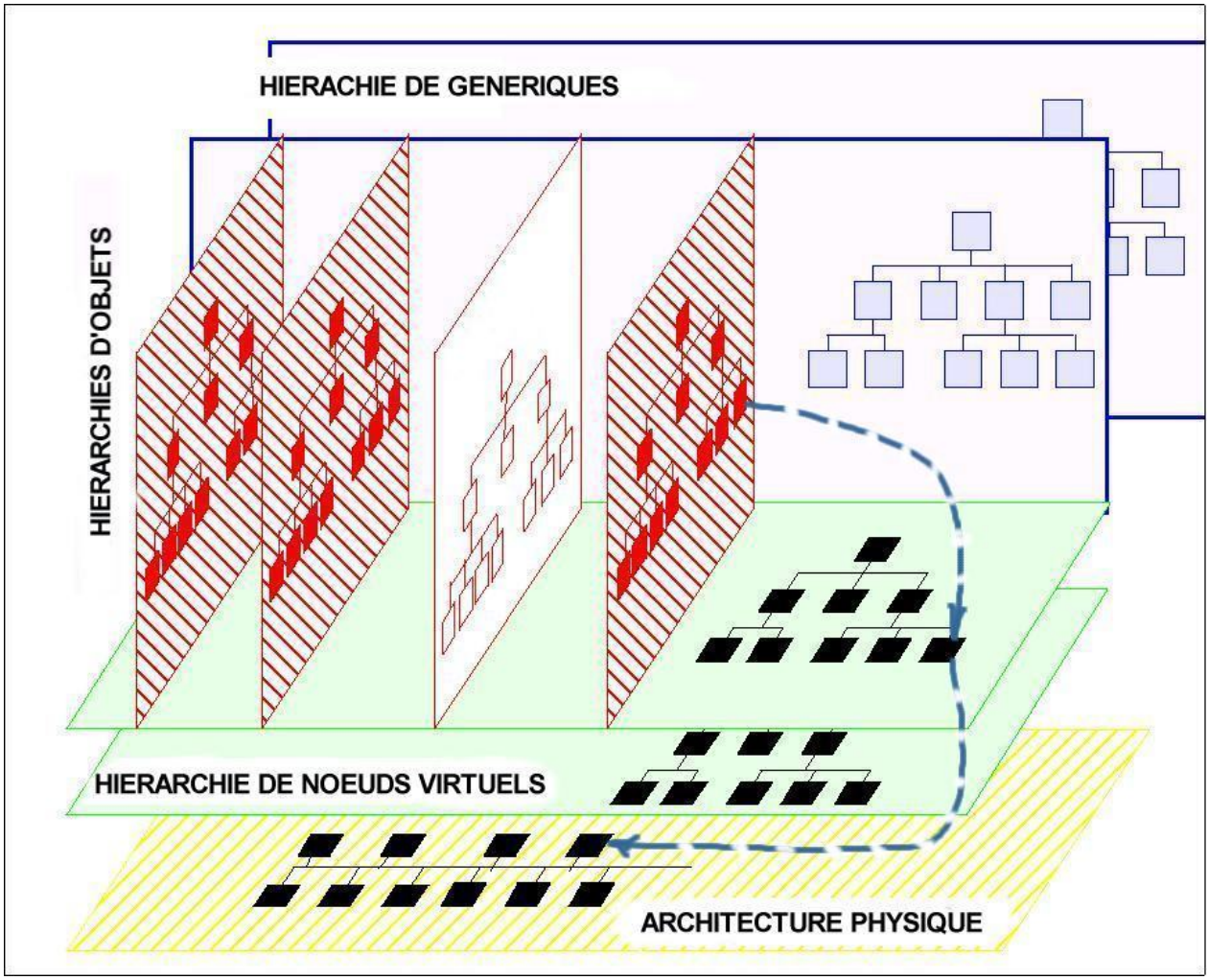

fig7- vue des espaces et hiérarchies HOOD

3.4.3. Les utilisations possibles des nœuds virtuels

On peut utiliser le concept de nœuds virtuels de différentes façons.
Je vais ici présenter succintement la démarche pensée dans [Sneed et Heitz, 07/93] qui tente d'utiliser le concept de nœud virtuel pour structurer les processus d'une application et pour synchroniser ces nœuds virtuels.
On part du constat suivant :
Les opérations des objets HOOD sont exécutées par des processus qui leur sont :

- soit complètement internes pour certains objets (exemple d'un objet parent qui contient forcément un ou plusieurs processus)
- soit complètement externes pour d'autres objets (exemple d'un objet terminal passif qui définit du code complètement séquentiel)
- soit ni externes, ni internes mais selon la chronologie d'exécution et les contraintes d'exécution (certains processus exécutent du logiciel dans un objet puis passent à un autre)

La définition d'un nœud logique comme regroupement d'objets HOOD permettrait donc de réaliser un regroupement des processus logiques associés. La notion de processus

logique est typiquement une tâche Ada mais on peut également penser à un processus de type Unix. On distingue alors les **processus légers**, qui correspondent plutôt à des implémentations de mécanismes de synchronisation, des **processus lourds**, qui correspondent plus à des entités logicielles, machines virtuelles complètes se partageant le processeur.

Pour établir l'implémentation des processus HOOD qui exécutent les opérations des objets et implémentent leur OBCS, nous disposons donc de deux niveaux de granularité :

- le processus léger correspondant à un processus logique HOOD, manipulé de manière transparente par la définition d'opérations et de contraintes d'exécution sur ces opérations,
- le processus lourd correspondant à la notion de nœud virtuel, regroupement d'objets et de processus légers en une seule entité.

Dans ce cadre là, il est conseillé de définir d'abord un processus unique regroupant toutes les tâches dans un programme Ada comme solution logique et maquettable. Puis, dans un deuxième temps, on considère un processus Unix comme un nœud virtuel sur lequel on alloue des objets HOOD de la solution logique. La définition du système en termes de processus Unix/nœuds virtuels va dépendre de deux types de contraintes :

- regroupement logique associé à des fonctionnalités et objets fortement couplés,
- regroupement logistique (un processus Unix par équipe de développement, autorisant un développement en parallèle, avec des bibliothèques de code indépendantes)

Dans le cadre d'une application temps réel, chaque nœud virtuel représente un processus lourd, et supporte l'implémentation d'une fonctionnalité du futur système avec les contraintes de performances associées. Les contraintes de temps sont exprimées par des contraintes sur les services offerts. Enfin, les communications inter-nœuds virtuels peuvent être exprimées d'abord de manière logique, c'est-à-dire par les flèches USE de HOOD, avec indications de flots de données.

Cette approche n'est plus d'actualité car l'outil de répartition GNAT-GLADE gère déjà de manière « cachée » les processus lancés par un exécutable correspondant à une partition. Les communications inter-partitions sont faites au moyen des RPC, et ce mécanisme est totalement transparent pour l'utilisateur de GNAT-GLADE.

Une meilleure approche, consisterait à étudier directement la correspondance entre les nœuds virtuels et les partitions. Autrement dit, on implémentera un nœud virtuel par un exécutable généré par l'outil GLADE, après génération du fichier de configuration GLADE par l'outil STOOD.

L'approche de développement adoptée se fera alors en 3 phases :

- phase de définition logique : définition d'une solution logique, non répartie, indépendante des contraintes non fonctionnelles.

- phase d'allocation : définition d'un modèle d'architecture en termes de nœuds virtuels. Pendant cette phase, le concepteur alloue les objets HOOD de la solution logique sur des nœuds virtuels.

- phase de configuration : regroupement de plusieurs nœuds virtuels en vue de leur affectation sur des processeurs. Cette phase définit des contraintes de configuration associées aux nœuds virtuels. Elles sont liées à la configuration des réseaux et aux possibilités de communication inter-processeurs. La prise en compte de ces contraintes conduit à la définition de la configuration logicielle des nœuds virtuels. Si les interfaces des logiciels supportant la communication inter-processeurs sont définies, cette configuration est semi-automatisable. Pendant cette phase, le concepteur regroupe les nœuds virtuels eux-mêmes, les alloue et les implante sur des nœuds physiques (les processeurs) du système cible.

L'outil STOOD permet de faciliter la phase d'allocation, et vérifie que chaque objet de la conception a été alloué à au moins un nœud virtuel.

Prenons l'exemple de la figure 8 qui représente la vue opération du module Logiciel de Vol de la conception commencée par Semra.

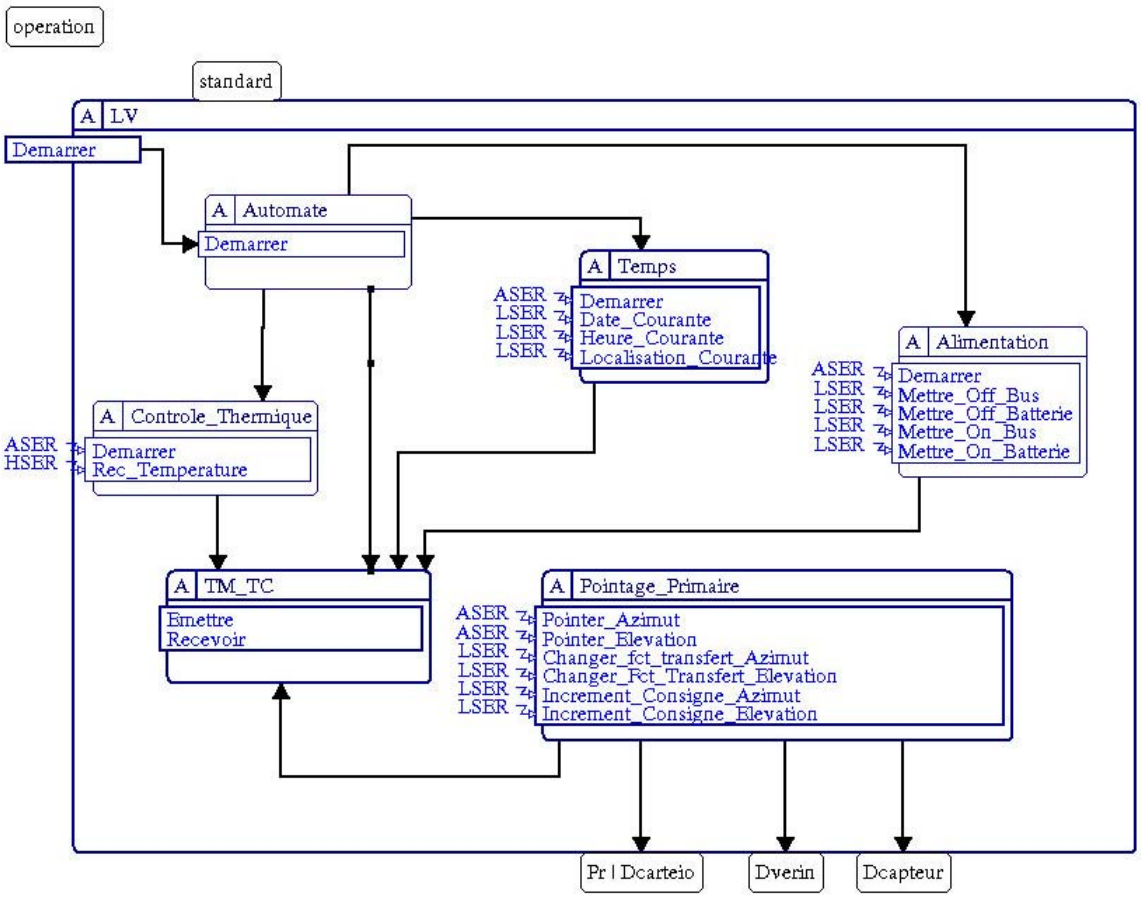

fig8- vue opération du LV

A la suite, j'ai représenté une allocation possible des objets HOOD à deux nœuds virtuels que j'ai directement assimilés aux calculateurs présents dans l'architecture matérielle actuelle. L'allocation des objets aux calculateurs MC et MPF est représentée respectivement sur les figures 9 et 10. Cette allocation n'est qu'une première approche, mais elle préfigure la méthodologie de répartition envisagée pour les développements futurs :

- les objets logiciels représentant les équipements sont affectés au calculateur auquel l'équipement est physiquement connecté : Magneto, GyroZ, Pivot et Couplemetre sur MC, PP_Verin, Inclinometre sur MPF. En effet, le « driver » de l'équipement ne peut pas être réparti ailleurs que sur le calculateur qui possède la carte d'interface.

- les objets de traitement qui ont un fort couplage avec les objets équipements sont localisés sur le même nœud virtuel : Boucle_Azimut sur MC, Boucle_Elevation sur MPF.

- pour la même raison, les objets Controle_thermique et Alimentation sont affectés au MC (les capteurs de température et le sous-système alimentation électrique sont connectés au MC).

- la base de temps n'a pas intérêt à être localisée sur un seul calculateur et accédée via la réseau pour des questions de précision. De ce fait, les objets relatifs au service Temps sont répliqués sur les deux calculateurs, le terminal GPS étant lui aussi physiquement connecté aux deux calculateurs.

- en considérant que le calculateur MC est le calculateur « maître », on lui associe les objets qui lui sont propres au niveau fonctionnel, c'est à dire les objets Automate, Contrôle Pointage, ainsi que et le contrôleur TC (la voie TC est connectée au MC). L'objet Trame TM est localisé sur le MPF puisque la voie TM lui est connectée.

- les objets Dcapteur, Dcarteio, Types et Standard sont répliqués sur les deux calculateurs comme peuvent l'être tous les objets non catégorisés ou relevant d'un pragma Pure.

Il est à noter que l'outil STOOD ne permet de répartir que des objets terminaux, ce qui est cohérent avec la philosophie HOOD : il n'y a de code que pour les objets terminaux, les objets non terminaux n'étant que des « coquilles vides » illustrant le processus mental de décomposition hiérarchique.

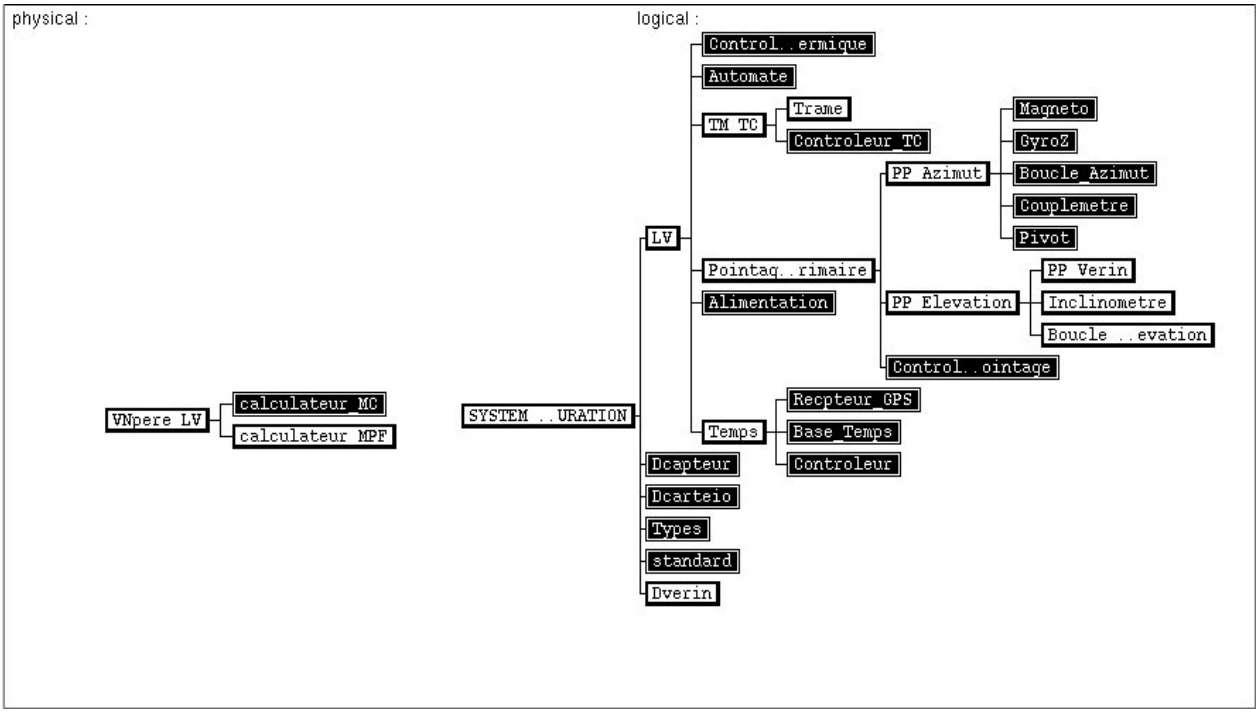

fig9- allocation des objets HOOD au calculateur MC

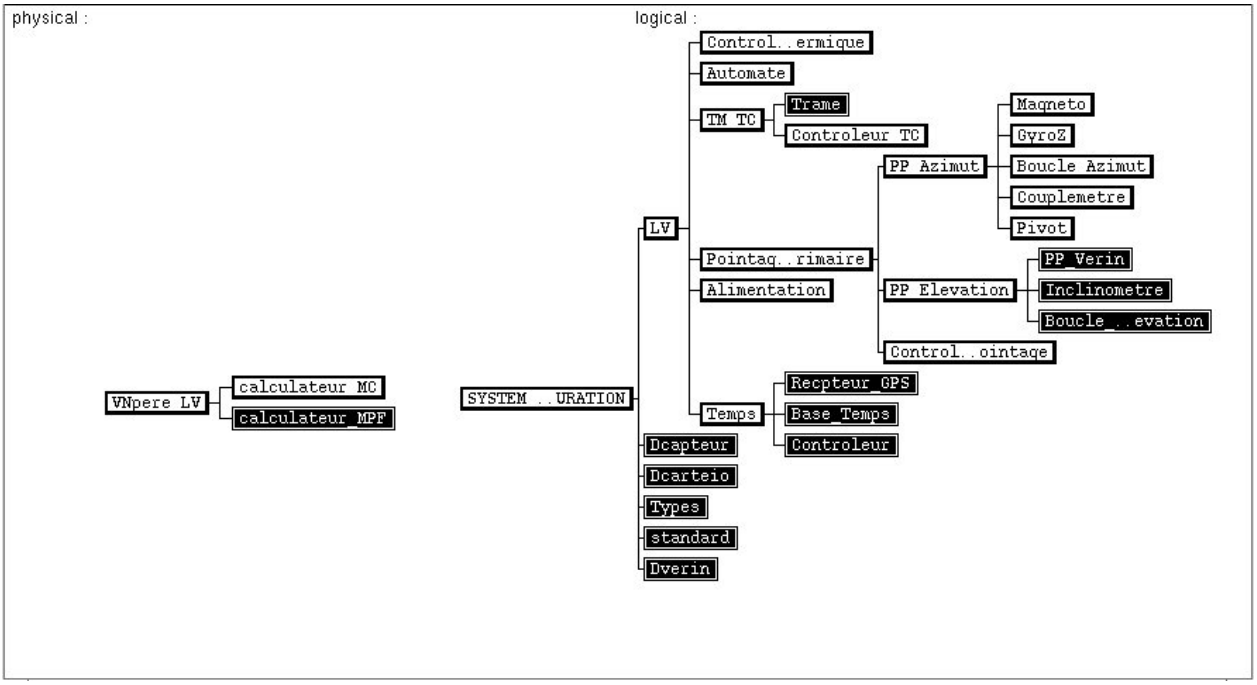

fig10- allocation des objets HOOD au calculateur MPF

3.5. BILAN ET TRAVAIL RESTANT A EFFECTUER

3.5.1. Prototypage avec STOOD

Le prototypage de l'application n'a pas pu être avancé avec l'outil STOOD, et il reste à concevoir une application minimale, à définir une hiérarchie de nœuds virtuels, et la répartition matérielle et le fichier de configuration GNAT-GLADE correspondants. Il faudra générer le code correspondant, le tester sur un calculateur puis le répartir sur deux calculateurs.

Tout le travail sera alors découpé de la sorte :
- élaborer une stratégie de répartition des unités de bibliothèque sur les calculateurs permettant d'effectuer les décisions de conception ci-dessous,
- l'appliquer (cf. représentation Stood ci-dessous),
- définir formellement les schémas de traduction permettant de faire réaliser automatiquement la traduction en Ada par le générateur de code de Stood.

Décision de conception	***Représentation dans Stood***	***Traduction en Ada***
Catégorisation des unités de bibliothèque	Propriétés des objets logiques	Pragmas de catégorisation (RCI, RT, SP, Pure…).
Affectation des unités de bibliothèque aux partitions	« allocation editor »	Directives d'association U.B – partition dans le fichier de configuration de GNATDIST
Caractéristiques des partitions	Propriétés des VN	Pragmas dans le fichier de configuration de GNATDIST

Les pragmas de localisation des partitions tel que Boot_server, Partition'Host, et de stockage des exécutables (Partition'Storage_Dir) sont à associer au VN correspondant. Les pragmas généraux Version, Starter sont des pragmas que l'on peut associer au VN père de l'application.

Dans un autre d'idée, il faudra aussi éprouver la mise en œuvre de la fonction de rétro-conception de l'outil STOOD, afin d'incorporer à une conception HOOD déjà entamée, des paquetages développés par ailleurs (génériques ou non) et réutilisés pour les applications à développer.

3.5.2. Bilan personnel

Je me suis rendu compte de la profusion des documents que l'on peut trouver sur l'internet traitant d'un sujet particulier. A noter que les versions françaises sont assez rares mais appréciées quand on les trouve. Il est d'ailleurs souvent difficile de résister à

l'envie de se documenter sur un sujet connexe au sujet qui était l'objet de notre rechercher initiale, mais c'est une façon de s'ouvrir à de nouveaux concepts. C'est ainsi que j'ai approfondi mes connaissances sur le système d'exploitation linux.

J'avoue ne pas avoir l'habitude de poser de questions sur les listes de discussion du type Ada-France lorsque j'ai été bloqué par une erreur de compilation ou d'exécution, préférant me documenter grâce aux manuels utilisateurs et aux exemples de démonstration fournis avec les logiciels. Cependant, je pense que c'est la toute la puissance du choix du tandem Linux-Ada, dont la communauté d'adeptes semble motivée par son emploi de manière plus étendue dans l'informatique, à en juger par quelques réponses lues sur des listes de questions.
L'assistance technique du CNES et de TNI pour l'outil STOOD m'ont agréablement aidé à régler mes problèmes d'installation, provenant souvent, d'erreurs liées aux distributions ou à de mauvaises configurations de la machine.

CONCLUSION

Mon travail a surtout été un travail de mise à plat des concepts et des outils qui sous-tendent la nouvelle architecture informatique pour les nacelles pointées. Nous avons validé en première approche la faisabilité d'une application Ada-Linux s'exécutant sur une maquette de calculateur embarqué. La répartition des programmes sur plusieurs calculateurs est rendue possible avec l'outil testé mais reste à expérimenter avec des cas plus représentatifs du vrai logiciel de vol. Bien que j'aie passé une partie non négligeable du temps à faire marcher les installations de logiciels, cette phase fait partie de la prise en main de tout projet informatique, et est formatrice.

Mon stage de DEA m'a permis de m'ouvrir à d'autres dimensions des travaux que l'informatique propose. La dimension « recherche » dans le domaine de l'informatique m'est apparue importante aussi bien pour la mise au point de nouveaux outils que pour l'intérêt personnel qu'elle suscite en moi. C'est en me documentant sur l'annexe des systèmes distribués de la norme du langage Ada95 que j'ai été amené à réfléchir en même temps à la stratégie de répartition et à la méthode de conception. J'espère que cette mise à plat permettra aux futurs concepteurs et développeurs des logiciels informatiques des nacelles pointées de disposer d'un document synthétisant les approches et solutions possibles liées à ce développement.

BIBLIOGRAPHIE

DOCUMENTS INTERNES

- Semra Sarpdag, Linux embarqué pour nacelles pointées, rapport de stage IUP ISI, août 2001.
- André Laurens, Plan informatique nacelles, ref AL-2000-001, 24 août 2000.
- I. Sneed et M. Heitz (Cisi Ingénierie), document Qualité Ada et HOOD, Règles et recommandations HOOD, 07/93

RAPPORTS ET MANUELS DE REFERENCE

- Laurent Pautet, Intergiciels schizophrènes : une solution à l'interopérabilité entre modèles de répartition, Mémoire pour l'obtention de l'Habilitation à Diriger des recherches de l'Université Pierre et Marie Curie-Paris VI, 17.12.2001.
- Jörg Kienzle, Network Applications in Ada 95, Rapport de stage, Ecole Polytechnique Fédérale de Lausanne, 1997.
- Laurent Pautet, Samuel Tardieu, GLADE User's Guide version 3-13p, GNAT Library for Ada Distributed Environment, 07.12.2001
- HOOD User's Group, HOOD Reference Manual, rev4, ref HRM4, 10.12.1995
- HOOD User's Group, HOOD User Manual, release 1.0, ref HUM-1.0 , 27.07.1994
- JP Rosen , HOOD : an Industrial Approach for Software Design, HOOD User's Group 1997
- J.-M.Bergé, L.-O Donzelle, V. Olive, J. Rouillard, ADA avec le sourire, Presses Polytechniques romandes et CNET-ENST,1989

ANNEXES

ANNEXE1 : ABRÉVIATIONS

ASER Asynchronous execution request

CCD Charged Coupled Device

DSA Distributed Systems Annex

GARLIC Generic Ada95 Reusable Library for Inter-partition Communication

ISO International Standard Organization

LAN Local Area Network

MC Module de Commande

MPF Module de Pointage Fin

NFS Network File System

OBCS OBject Control Structure

ODS Object Description Skeleton

OSI Open System Interconnection

PN Physical Node

RCI Remote Call Interface

RPC Remote Procedure Call, appel de procédure distante

RPM RedHat Package Manager

RTE Run Time Environment

TC Télécommande

TM Télémesure

VN Virtual Node

VNCS Virtual Node Control Structure

ANNEXE2 : PROGRAMMES DE TEST ADA+LINUX

```
/*************************************************/
/*                test_boucle.adb                */
/*************************************************/

with Ada.Numerics.Generic_Elementary_Functions;
with Ada.Calendar;
use Ada.Calendar;
with Ada.Text_Io;
use Ada.Text_Io;

procedure Test_Boucle is

   package Math is new Ada.Numerics.Generic_Elementary_Functions( Float);

   Nombre_De_Boucles : constant              := 100;
   Debut,
   Fin,
   Duree             :          Day_Duration;
   Duree_Totale      :          Duration     := 0.0;
   X,
   Y                 :          Float        := 1.0;
begin

   for I in 1 .. Nombre_De_Boucles loop
      Debut := Seconds (Clock);

      Y := Math.Sin (X / 2.0);
      X := Y;

      Fin := Seconds (Clock);

      Duree := Fin - Debut;
      Duree_Totale := Duree_Totale + Duree;

      Put_Line ("duree boucle " & Positive'Image (I) & " : " & Duration'
         Image (Fin - Debut) & " s");
   end loop;

   New_Line;
   Put_Line ("duree totale  : " & Duration'Image (Duree_Totale) & " s");
   Put_Line ("duree moyenne : " & Duration'Image (Duree_Totale /
         Nombre_De_Boucles) &
      " s");

end Test_Boucle;
```

```
/*****************************************************/
/*                tache.ads                          */
/*****************************************************/

with System;

package Tache is

   type Mode_Tache is
         (Aperiodique,
          Periodique);
   subtype Taille_Paquet_Tm is Natural range 0 .. 2048;

   type Caracteristiques_Tache
         (Mode : Mode_Tache := Periodique) is
      record
         Nom      : String (1 .. 10);
         Priorite : System.Priority  := System.Default_Priority;
case Mode is when Aperiodique => null;
   when Periodique =>
   Periode              : Duration         := 1.0;
   Duree_Phase_Lecture  : Duration         := 0.0;
   Duree_Phase_Calcul   : Duration         := 0.0;
   Flot_Phase_Tm        : Taille_Paquet_Tm := 0;
   Duree_Phase_Commande : Duration         := 0.0;
end case;
end record;

type Evenement is
      (Creation,
       Demarrage,
       Arret,
       Debut_Phase_Lecture,
       Fin_Phase_Lecture,
       Debut_Phase_Calcul,
       Fin_Phase_Calcul,
       Debut_Phase_Tm,
       Fin_Phase_Tm,
       Debut_Phase_Commande,
       Fin_Phase_Commande);

Erreur : exception;

end Tache;

/*****************************************************/
/*                tache-cyclique.ads                 */
/*****************************************************/
generic
   Descripteur : Caracteristiques_Tache;
   with procedure envoi (tag : in string; taille : in taille_paquet_tm);
package Tache.Cyclique is
   task Controle is
      entry Demarrer;
      entry Arreter;
   end Controle;
end Tache.Cyclique;

/*****************************************************/
/*                tache-cyclique.adb                 */
/*****************************************************/

with Ada.Text_Io;
```

```
use Ada.Text_Io;
with Ada.Dynamic_Priorities;
with Ada.Strings.Maps;
use Ada.Strings.Maps;
with Ada.Strings.Fixed;
use Ada.Strings.Fixed;
use Ada.Strings;
with Ada.Calendar;
with Ada.Numerics.Generic_Elementary_Functions;

package body Tache.Cyclique is

   Espace_En_Underscore :          Character_Mapping := To_Mapping (From
     => " ", To => "_");
   Nom_Tache            : constant String            := Translate (Source
     =>    Trim    (Descripteur.Nom,    Side    =>    Both),    Mapping    =>
Espace_En_Underscore);


   Log : Ada.Text_Io.File_Type;

   function Construit_Nom_Log (
         Tache : in     String )
     return String is
      use Ada.Calendar;
      A : Year_Number;
      M : Month_Number;
      J : Day_Number;
      S : Day_Duration;
   begin
      Split (Clock, A, M, J, S);
      return Translate (
         Source     =>  Tache  &  '_'  &  Day_Number'Image  (J)  &  '_'  &
Month_Number'Image (M) & '_' & Year_Number'Image (A) & '_' & Natural'Image
(Natural (S)) & ".log",
         Mapping => Espace_En_Underscore);
   end Construit_Nom_Log;

   Nom_Fichier_Log : constant String := Construit_Nom_Log (Nom_Tache);

   --Duree_Du_Tour : constant Duration := 4.5e-6;
   Duree_Du_Tour : constant Duration := 1.0e-4;

   package Math is new Ada.Numerics.Generic_Elementary_Functions (Float);

   procedure Enregistrer (
         Action : in     Evenement ) is
      use Ada.Calendar;
      S : Day_Duration;
   begin
      S := Seconds (Clock);
      Put_Line (
         File => Log,
         Item  =>  Nom_Tache  &  '  '  &  Day_Duration'Image  (S)  &  '  '  &
Evenement'Image (Action));
   end Enregistrer;

   procedure Phase_Lecture (
         Duree : in     Duration ) is
   begin
      Enregistrer (Debut_Phase_Lecture);
      delay Duree;
      Enregistrer (Fin_Phase_Lecture);
   end Phase_Lecture;
```

```
   procedure Phase_Calcul (
         Duree : in     Duration ) is
      Nombre_De_Tours : constant Natural := Natural (Float (Duree) / Float
        (Duree_Du_Tour));

      X,
      Y : Float := 1.0;
   begin
      Enregistrer (Debut_Phase_Calcul);
      for I in 1 .. Nombre_De_Tours loop
         Y := Math.Sin (X / 2.0);
         X := Y;
      end loop;
      Enregistrer (Fin_Phase_Calcul);
   end Phase_Calcul;

   procedure Phase_Tm (
         Taille : in     Taille_Paquet_Tm ) is
   begin
      Enregistrer (Debut_Phase_Tm);
      Envoi (Nom_Tache, Taille);
      Enregistrer (Fin_Phase_Tm);
   end Phase_Tm;

   procedure Phase_Commande (
         Duree : in     Duration ) is
   begin
      Enregistrer (Debut_Phase_Commande);
      delay Duree;
      Enregistrer (Fin_Phase_Commande);
   end Phase_Commande;

   --------------
   -- Controle --
   --------------
   task body Controle is
   begin
      Ada.Dynamic_Priorities.Set_Priority (Descripteur.Priorite);
      loop
         select
            accept Demarrer do
               Enregistrer (Demarrage);
            end Demarrer;

            Activite : loop

               select
                  accept Arreter do
                     Enregistrer (Arret);
                  end Arreter;
                  exit Activite;
               else

                  Action_Recurrente : declare
                     use Ada.Calendar;
                     Prochaine_Echeance   :   constant   Time   :=   Clock   +
Descripteur.Periode;


                  begin
                     Phase_Lecture (Descripteur.Duree_Phase_Lecture);
                     Phase_Calcul  (Descripteur.Duree_Phase_Calcul);
                     Phase_Tm (Descripteur.Flot_Phase_Tm);
                     Phase_Commande (Descripteur.Duree_Phase_Commande);
```

```
                  delay until Prochaine_Echeance;
               end Action_Recurrente;

            end select;

         end loop Activite;

      or
         terminate;
      end select;
   end loop;
end Controle;

begin

   if Descripteur.Mode /= Periodique then
      raise Erreur;
   end if;

   Create (
      File => Log,
      Name => Nom_Fichier_Log);

   Enregistrer (Creation);

end Tache.Cyclique;


/*****************************************************/
/*                test_taches.adb                    */
/*                (test_6taches)                     */
/*****************************************************/
with Tache;
with Tache.Cyclique;
with Ada.Text_Io;
with Unchecked_Deallocation;

procedure Test_Taches is

   task Tm is
      pragma Priority (1);
      entry Envoi (
            Tag    : in     String;
            Taille : in     Tache.Taille_Paquet_Tm );
   end Tm;

   task body Tm is
      type Access_String is access String;
      procedure Free is
      new Unchecked_Deallocation (
         Object => String,
         Name   => Access_String);
      Tag_Courant     : Access_String;
      Taille_Courante : Tache.Taille_Paquet_Tm;
      use Ada.Text_Io;
   begin
      loop
         select
            accept Envoi (
                  Tag    : in     String;
                  Taille : in     Tache.Taille_Paquet_Tm ) do
               Tag_Courant := new String'(Tag);
               Taille_Courante := Taille;
            end Envoi;
```

```
            Put (Tag_Courant.all & " : ");
            for I in 1..Taille_Courante loop
               Put ("x");
            end loop;
            New_Line;
            Free (Tag_Courant);
         or
            terminate;
         end select;
      end loop;
   end Tm;

   package tache1 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache1    ", Priorite => 10, Periode => 0.1,
         Duree_Phase_Lecture => 0.03, Duree_Phase_Calcul => 0.05,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.1),
      Envoi => Tm.Envoi);

   package tache2 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache2    ", Priorite => 5, Periode => 1.0,
         Duree_Phase_Lecture => 0.1, Duree_Phase_Calcul => 0.2,
         Flot_Phase_Tm => 200, Duree_Phase_Commande => 0.1),
      Envoi => Tm.Envoi);

   package tache3 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache3    ", Priorite => 2, Periode => 3.0,
         Duree_Phase_Lecture => 0.2, Duree_Phase_Calcul => 1.0,
         Flot_Phase_Tm => 20, Duree_Phase_Commande => 0.2),
                                          Envoi => Tm.Envoi);

  package tache4 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache4    ", Priorite => 10, Periode => 0.1,
         Duree_Phase_Lecture => 0.3, Duree_Phase_Calcul => 0.05,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.3),
      Envoi => Tm.Envoi);

   package tache5 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache5    ", Priorite => 5, Periode => 1.0,
         Duree_Phase_Lecture => 0.4, Duree_Phase_Calcul => 0.2,
         Flot_Phase_Tm => 200, Duree_Phase_Commande => 0.4),
      Envoi => Tm.Envoi);

   package tache6 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache6    ", Priorite => 2, Periode => 3.0,
         Duree_Phase_Lecture => 0.5, Duree_Phase_Calcul => 1.0,
         Flot_Phase_Tm => 20, Duree_Phase_Commande => 0.5),
      Envoi => Tm.Envoi);
begin

   tache2.Controle.Demarrer;

   tache1.Controle.Demarrer;
   delay 10.0;
   tache1.Controle.Arreter;

   tache3.Controle.Demarrer;

   delay 2.0;
   tache1.Controle.Demarrer;
```

```
   delay 10.0;
   tache1.Controle.Arreter;

   tache2.Controle.Arreter;
   tache3.Controle.Arreter;

   --on rajoute des taches
   tache4.Controle.Demarrer;

   tache5.Controle.Demarrer;
   delay 10.0;
   tache5.Controle.Arreter;

   tache6.Controle.Demarrer;

   delay 2.0;
   tache5.Controle.Demarrer;
   delay 10.0;
   tache5.Controle.Arreter;

   tache4.Controle.Arreter;
   tache6.Controle.Arreter;

end Test_Taches;


/*****************************************************/
/*                test_12taches.adb                  */
/*****************************************************/
with Tache;
with Tache.Cyclique;
with Ada.Text_Io;
with Unchecked_Deallocation;

procedure Test_12Taches is

   task Tm is
      pragma Priority (1);
      entry Envoi (
            Tag    : in     String;
            Taille : in     Tache.Taille_Paquet_Tm );
   end Tm;

   task body Tm is
      type Access_String is access String;
      procedure Free is
      new Unchecked_Deallocation (
         Object => String,
         Name   => Access_String);
      Tag_Courant     : Access_String;
      Taille_Courante : Tache.Taille_Paquet_Tm;
      use Ada.Text_Io;
   begin
      loop
         select
            accept Envoi (
                  Tag    : in     String;
                  Taille : in     Tache.Taille_Paquet_Tm ) do
               Tag_Courant := new String'(Tag);
               Taille_Courante := Taille;
            end Envoi;
            Put (Tag_Courant.all & " : ");
            for I in 1..Taille_Courante loop
               Put ("x");
```

```
            end loop;
            New_Line;
            Free (Tag_Courant);
         or
            terminate;
         end select;
      end loop;
   end Tm;


package Tache_TC is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tacheTC   ", Priorite => 5, Periode => 0.1,
         Duree_Phase_Lecture => 0.02, Duree_Phase_Calcul => 0.01,
         Flot_Phase_Tm => 20, Duree_Phase_Commande => 0.0),
      Envoi => Tm.Envoi);

package Tache_TM is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tacheTM   ", Priorite => 1, Periode => 0.1,
         Duree_Phase_Lecture => 0.0, Duree_Phase_Calcul => 0.01,
         Flot_Phase_Tm => 250, Duree_Phase_Commande => 0.065),
      Envoi => Tm.Envoi);

package Tache_pilotage1 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_pil1", Priorite => 10, Periode => 0.1,
         Duree_Phase_Lecture => 0.02, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.01),
                                          Envoi => Tm.Envoi);
package Tache_pilotage2 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_pil2", Priorite => 10, Periode => 0.1,
         Duree_Phase_Lecture => 0.02, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.01),
                                             Envoi => Tm.Envoi);
package Tache_pilotage3 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_pil3", Priorite => 10, Periode => 0.1,
         Duree_Phase_Lecture => 0.02, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.01),
                                          Envoi => Tm.Envoi);

package Tache_pilotage4 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_pil4", Priorite => 10, Periode => 0.1,
         Duree_Phase_Lecture => 0.02, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.01),
                                          Envoi => Tm.Envoi);

package Tache_servitude1 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_ser1", Priorite => 2, Periode => 30.0 ,
         Duree_Phase_Lecture => 0.03, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 300, Duree_Phase_Commande => 0.0),
      Envoi => Tm.Envoi);

package Tache_servitude2 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_ser2", Priorite => 2, Periode => 10.0 ,
         Duree_Phase_Lecture => 0.03, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 300, Duree_Phase_Commande => 0.0),
      Envoi => Tm.Envoi);

package Tache_servitude3 is new Tache.Cyclique (
```

```
      Descripteur => (Mode => Periodique,
         Nom => "tache_ser3", Priorite => 2, Periode => 1.0 ,
         Duree_Phase_Lecture => 0.03, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 300, Duree_Phase_Commande => 0.0),
      Envoi => Tm.Envoi);

 package Tache_servitude4 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_ser4", Priorite => 2, Periode => 1.0 ,
         Duree_Phase_Lecture => 0.03, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 300, Duree_Phase_Commande => 0.0),
      Envoi => Tm.Envoi);


 package Tache_astro1 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_ast1", Priorite => 5, Periode => 1.0,
         Duree_Phase_Lecture => 0.0, Duree_Phase_Calcul => 0.1,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.0),
      Envoi => Tm.Envoi);

 package Tache_astro2 is new Tache.Cyclique (
      Descripteur => (Mode => Periodique,
         Nom => "tache_ast2", Priorite => 5, Periode => 0.1,
         Duree_Phase_Lecture => 0.0, Duree_Phase_Calcul => 0.02,
         Flot_Phase_Tm => 100, Duree_Phase_Commande => 0.0),
                                              Envoi => Tm.Envoi);


begin
   Tache_TC.Controle.Demarrer;
   Tache_TM.Controle.Demarrer;

   Tache_pilotage1.Controle.Demarrer;
   Tache_pilotage2.Controle.Demarrer;
   Tache_pilotage3.Controle.Demarrer;
   Tache_pilotage4.Controle.Demarrer;

   Tache_servitude1.Controle.Demarrer;
   Tache_servitude2.Controle.Demarrer;
   Tache_servitude3.Controle.Demarrer;
   Tache_servitude4.Controle.Demarrer;

   Tache_astro1.Controle.Demarrer;
   Tache_astro2.Controle.Demarrer;

   delay 300.0;

   Tache_TC.Controle.Arreter;
   Tache_TM.Controle.Arreter;

   Tache_pilotage1.Controle.Arreter;
   Tache_pilotage2.Controle.Arreter;
   Tache_pilotage3.Controle.Arreter;
   Tache_pilotage4.Controle.Arreter;

   Tache_servitude1.Controle.Arreter;
   Tache_servitude2.Controle.Arreter;
   Tache_servitude3.Controle.Arreter;
   Tache_servitude4.Controle.Arreter;

   Tache_astro1.Controle.Arreter;
   Tache_astro2.Controle.Arreter;
```

```
end Test_12Taches;

/*************************************************/
/*                depouille_taches.adb           */
/*************************************************/
with Ada.Command_Line;
with Ada.Text_Io;
with Mots.Extraction;
with Tache;

procedure Depouille_Taches is

   type Decompte_Temps is
      record
         Debut              : Duration := 0.0;
         Fin                : Duration := 0.0;
         Duree              : Duration := 0.0;
         Duree_Totale       : Duration := 0.0;
         Nombre_Occurrences : Natural  := 0;
      end record;

   Nom_Fichier_Log   :  constant  String                                  :=
Ada.Command_Line.Argument
     (1);
   Log             :          Ada.Text_Io.File_Type;
   Ligne           :          String (1 .. 200);
   Dernier_Lu      :          Natural;
   Activite,
   Lecture,
   Calcul,
   Tm,
   Commande        :          Decompte_Temps;
   Premiere_Fois   :          Boolean              := True;
   Periode         :          Duration             := 0.0;
   Cumul_Periode   :          Duration             := 0.0;

begin
   Ada.Text_Io.Open (
      File => Log,
      Name => Nom_Fichier_Log,
      Mode => Ada.Text_Io.In_File);

   while not Ada.Text_Io.End_Of_File (Log) loop

      Ada.Text_Io.Get_Line (
         File => Log,
         Item => Ligne,
         Last => Dernier_Lu);

      declare
         package Le_Mot is new Mots.Extraction (
            De          => Ligne (1 .. Dernier_Lu),
            Separes_Par => " ");
         Nom_Tache : constant String   := Le_Mot.Courant;
         Temps     : constant Duration := Duration'Value (Le_Mot.Suivant);


         Action    :    constant    Tache.Evenement    :=    Tache.Evenement'Value
(Le_Mot.Suivant);


         use Tache;
      begin
         case Action is
```

```
            when Creation =>
               null;
            when Demarrage =>
               Activite.Debut := Temps;
            when Arret =>
               Activite.Fin := Temps;
               Activite.Duree := Activite.Fin - Activite.Debut;
               Activite.Duree_Totale := Activite.Duree_Totale + Activite.
                  Duree;
               Activite.Nombre_Occurrences := Activite.Nombre_Occurrences +
                  1;
            when Debut_Phase_Lecture =>
               if Premiere_Fois then
                  Ada.Text_Io.Put_Line ("Tache : " & Nom_Tache);
                  Premiere_Fois := False;
               else
                  Periode := Temps - Lecture.Debut;
                  Cumul_Periode := Cumul_Periode + Periode;
               end if;
               Lecture.Debut := Temps;
            when Fin_Phase_Lecture =>
               Lecture.Fin := Temps;
               Lecture.Duree := Lecture.Fin - Lecture.Debut;
               Lecture.Duree_Totale := Lecture.Duree_Totale + Lecture.
                  Duree;
               Lecture.Nombre_Occurrences := Lecture.Nombre_Occurrences +
                  1;
            when Debut_Phase_Calcul =>
               Calcul.Debut := Temps;
            when Fin_Phase_Calcul =>
               Calcul.Fin := Temps;
               Calcul.Duree := Calcul.Fin - Calcul.Debut;
               Calcul.Duree_Totale := Calcul.Duree_Totale + Calcul.
                  Duree;
               Calcul.Nombre_Occurrences := Calcul.Nombre_Occurrences +
                  1;
            when Debut_Phase_Tm =>
               Tm.Debut := Temps;
            when Fin_Phase_Tm =>
               Tm.Fin := Temps;
               Tm.Duree := Tm.Fin - Tm.Debut;
               Tm.Duree_Totale := Tm.Duree_Totale + Tm.Duree;
               Tm.Nombre_Occurrences := Tm.Nombre_Occurrences + 1;
            when Debut_Phase_Commande =>
               Commande.Debut := Temps;
            when Fin_Phase_Commande =>
               Commande.Fin := Temps;
               Commande.Duree := Commande.Fin - Commande.Debut;
               Commande.Duree_Totale := Commande.Duree_Totale + Commande.
                  Duree;
               Commande.Nombre_Occurrences := Commande.Nombre_Occurrences +
                  1;
         end case;
      end;
   end loop;

      Ada.Text_Io.New_Line;
   Ada.Text_Io.Put_Line ("Activite : " & Natural'Image (Activite.
         Nombre_Occurrences) & " fois");
   if Activite.Nombre_Occurrences > 0 then
      Ada.Text_Io.Put ("   duree totale  : " & Duration'Image (
            Activite.Duree_Totale) & " s");
      Ada.Text_Io.Put_Line (" - duree moyenne : " & Duration'Image (
            Activite.Duree_Totale / Activite.Nombre_Occurrences) & " s");
   end if;
```

```
      Ada.Text_Io.New_Line;
   Ada.Text_Io.Put_Line ("Lecture : " & Natural'Image (Lecture.
         Nombre_Occurrences) & " fois");
   if Lecture.Nombre_Occurrences > 0 then
      Ada.Text_Io.Put ("   duree totale  : " & Duration'Image (
            Lecture.Duree_Totale) & " s");
      Ada.Text_Io.Put_Line (" - duree moyenne : " & Duration'Image (
            Lecture.Duree_Totale / Lecture.Nombre_Occurrences) & " s");
   end if;
   if Lecture.Nombre_Occurrences > 1 then
      Ada.Text_Io.Put_Line ("   periode moyenne : " & Duration'Image (
            Cumul_Periode / (Lecture.Nombre_Occurrences - 1)) & " s");
   end if;

      Ada.Text_Io.New_Line;
   Ada.Text_Io.Put_Line ("Calcul : " & Natural'Image (Calcul.
         Nombre_Occurrences) & " fois");
   if Calcul.Nombre_Occurrences > 0 then
      Ada.Text_Io.Put ("   duree totale  : " & Duration'Image (
            Calcul.Duree_Totale) & " s");
      Ada.Text_Io.Put_Line (" - duree moyenne : " & Duration'Image (
            Calcul.Duree_Totale / Calcul.Nombre_Occurrences) & " s");
   end if;

      Ada.Text_Io.New_Line;
   Ada.Text_Io.Put_Line ("Tm : " & Natural'Image (Tm.
         Nombre_Occurrences) & " fois");
   if Tm.Nombre_Occurrences > 0 then
      Ada.Text_Io.Put ("   duree totale  : " & Duration'Image (
            Tm.Duree_Totale) & " s");
      Ada.Text_Io.Put_Line (" - duree moyenne : " & Duration'Image (
            Tm.Duree_Totale / Tm.Nombre_Occurrences) & " s");
   end if;

      Ada.Text_Io.New_Line;
      Ada.Text_Io.Put_Line ("Commande : " & Natural'Image (Commande.
         Nombre_Occurrences) & " fois");
   if Commande.Nombre_Occurrences > 0 then
      Ada.Text_Io.Put ("   duree totale  : " & Duration'Image (
            Commande.Duree_Totale) & " s");
      Ada.Text_Io.Put_Line (" - duree moyenne : " & Duration'Image (
            Commande.Duree_Totale / Commande.Nombre_Occurrences) & " s");
   end if;

exception
   when Ada.Text_Io.Name_Error =>
      Ada.Text_Io.Put_Line (
         "***** nom fichier log incorrect ou inconnu : " & Nom_Fichier_Log);
end Depouille_Taches;
```

ANNEXE3 : DONNEES DEPOUILLEES DU TEST_TACHES AVEC 6 TACHES SUR LE PC104

Tâche : tache1

Activité : 2 fois
  durée totale : 20.156229000 s - durée moyenne : 10.078114500 s
Lecture : 134 fois
  durée totale : 5.286845000 s - durée moyenne : 0.039454067 s
  période moyenne : 0.165527300 s
Calcul : 134 fois
  durée totale : 0.146764000 s - durée moyenne : 0.001095253 s
Tm : 134 fois
  durée totale : 0.199946000 s - durée moyenne : 0.001492134 s
Commande : 134 fois
  durée totale : 14.310149000 s - durée moyenne : 0.106792156 s
****

Tâche : tache2

Activité : 1 fois
  durée totale : 22.236714000 s - durée moyenne : 22.236714000 s
Lecture : 22 fois
  durée totale : 2.412877000 s - durée moyenne : 0.109676227 s
  période moyenne : 1.010641333 s
Calcul : 22 fois
  durée totale : 0.076616000 s - durée moyenne : 0.003482545 s
Tm : 22 fois
  durée totale : 0.059527000 s - durée moyenne : 0.002705772 s
Commande : 22 fois
  durée totale : 2.291384000 s - durée moyenne : 0.104153818 s
****

Tâche : tache3

Activité : 1 fois
  durée totale : 15.058723000 s - durée moyenne : 15.058723000 s
Lecture : 5 fois
  durée totale : 1.052242000 s - durée moyenne : 0.210448400 s
  période moyenne : 3.011944500 s
Calcul : 5 fois
  durée totale : 0.087116000 s - durée moyenne : 0.017423200 s
Tm : 5 fois
  durée totale : 0.054038000 s - durée moyenne : 0.010807600 s
Commande : 5 fois
  durée totale : 0.001790000 s - durée moyenne : 0.000358000 s
****

Tâche : tache4

Activité : 1 fois
  durée totale : 22.329454000 s - durée moyenne : 22.329454000 s
Lecture : 36 fois
  durée totale : 11.133319000 s - durée moyenne : 0.309258861 s
  période moyenne : 0.620253971 s
Calcul : 36 fois
  durée totale : 0.039924000 s - durée moyenne : 0.001109000 s
Tm : 36 fois
  durée totale : 0.037429000 s - durée moyenne : 0.001039694 s
Commande : 36 fois
  durée totale : 11.061963000 s - durée moyenne : 0.307276750 s
****

Tâche : tache5

Activité : 2 fois
  durée totale : 20.217380000 s - durée moyenne : 10.108690000 s
Lecture : 20 fois
  durée totale : 8.195045000 s - durée moyenne : 0.409752250 s
  période moyenne : 1.116687263 s
Calcul : 20 fois
  durée totale : 0.068369000 s - durée moyenne : 0.003418450 s
Tm : 20 fois
  durée totale : 0.017439000 s - durée moyenne : 0.000871950 s
Commande : 20 fois
  durée totale : 8.092656000 s - durée moyenne : 0.404632800 s
****
Tâche : tache6

Activité : 1 fois
  durée totale : 15.059128000 s - durée moyenne : 15.059128000 s
Lecture : 5 fois
  durée totale : 2.548320000 s - durée moyenne : 0.509664000 s
  période moyenne : 3.012125750 s
Calcul : 5 fois
  durée totale : 0.081245000 s - durée moyenne : 0.016249000 s
Tm : 5 fois
  durée totale : 0.005569000 s - durée moyenne : 0.001113800 s
Commande : 5 fois
  durée totale : 2.507592000 s - durée moyenne : 0.501518400 s

ANNEXE4 : FICHIER DE CONFIGURATION (SIMGEST.CFG) DE L'EXEMPLE BANK FOURNI PAR GNAT-GLADE

```
configuration Simcity is

   pragma Version (False);
   -- Don't want to check 'Version.

   pragma Starter (None);
   -- Bank_Client and Bank_Server are launched manually.

   pragma Boot_Server ("tcp", "localhost:5557");
   -- Bank_Server service has to be mapped on a given host and a
   -- given port.

   Bank_Server : Partition := (Server);
   -- There is only one RCI package. Note that this RCI package
   -- is similar to Bank package with only the limited features
   -- allowed to a client.

   Bank_Client : Partition;
   for Bank_Client'Termination use Local_Termination;
   -- A client should not wait for the termination of the whole
   -- distributed application to terminate.

   procedure Manager is in Bank_Server;

   procedure Client;
   for Bank_Client'Main use Client;
   -- The main subprogram of partition Bank_Client is not a
   -- dummy one.

   --   The zip filter works only on 32 bits machines, don't
   --   try it on Digital Unix/Alpha.
   --   Channel_1 : Channel := (Bank_Client, Bank_Server);
   --   for Channel_1'Filter use "zip";

end Simcity;
```

ANNEXE5 : FICHIER DE CONFIGURATION (SPIRAL.CFG) DE L'EXEMPLE ERATHO-SPIRAL

```
configuration Spiral4 is

   pragma Starter (Shell);

   pragma Boot_Server ("tcp", "localhost:5557");

   Partition1 : Partition := (Prime_1);
   -- la partition1 comporte uniquement l'unité de bibliothèque Prime_1
   Partition2 : Partition := (Prime_2);
   Partition3 : Partition := (Prime_3);

   for Partition2'Host use "pc-bana-daurat.cst.cnes.fr";
   -- la partition2 sera hébergée sur pc-bana-daurat
   for Partition3'Host use "localhost";

   for Partition'Storage_Dir use "bin";
   procedure Mainloop is in Partition1;

end Spiral4;
```