INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet SOLIDOR

Construction de systèmes et d'applications distribués

Rennes, Rocquencourt

THÈME 1B

Rapport d'Activité

2000

Table des matières

1 Composition de l'équipe **3**

2 Présentation et objectifs généraux **4**

3 Fondements scientifiques **5**
3.1 Architectures logicielles de systèmes . . . 6
3.2 Temps-réel . . . 7
3.3 Tolérance aux fautes . . . 9
3.4 Informatique mobile . . . 10

4 Domaines d'applications **11**
4.1 Avionique modulaire embarquée . . . 11
4.2 Ordinateurs de poche . . . 12
4.3 Systèmes d'information pour ordinateurs de poche en environnement mobile . . 13

5 Logiciels **14**
5.1 Environnement pour l'adaptation systématique de middleware – Aster . . . 14
5.2 Environnement d'exécution pour l'avionique embarquée – Hades . . . 15
5.3 Environnement d'exécution Java pour architecture embarquée . . . 17

6 Résultats nouveaux **17**
6.1 Architectures logicielles de systèmes distribués . . . 17
6.1.1 Conception d'architectures de systèmes . . . 18
6.1.2 Analyse des propriétés non-fonctionnelles des systèmes distribués . . . 19
6.1.3 Synthèse d'architectures middleware pour systèmes client-serveur à base de composants . . . 19
6.1.4 Synthèse d'architectures de sites Web . . . 21
6.1.5 Nouvelles architectures de systèmes distribués . . . 21
6.2 Exécutifs pour applications temps-réel strict hautement disponibles . . . 22
6.2.1 Aspects tolérance aux fautes . . . 22
6.2.2 Aspects temps-réel strict . . . 23
6.3 Environnement Java embarqué pour ordinateur de poche . . . 25
6.3.1 Machine d'exécution Java modulaire Scratchy . . . 26
6.3.2 Ordonnancement multi-critère d'applications multimédias . . . 26
6.3.3 Ramasse-miettes adapté aux applications multimédia . . . 27
6.4 Systèmes d'information spontanés (SIS) . . . 27
6.4.1 Communication entre les nœuds d'un SIS . . . 28
6.4.2 Découverte et échanges d'informations au sein d'un SIS . . . 28
6.5 Informatique nomade et réseaux sans fil . . . 29
6.6 Systèmes adaptatifs . . . 30
6.7 Caches coopératifs pour systèmes d'information sur Internet . . . 32

7 Contrats industriels (nationaux, européens et internationaux) 33
7.1 Contrats nationaux 33
7.1.1 Hades (partenariat DGA/Dassault-Aviation) 33
7.1.2 Texas Instruments 33
7.1.3 Alcatel 34

8 Actions régionales, nationales et internationales 34
8.1 Actions européennes 34
8.1.1 Projet LTR C3DS 34
8.1.2 Projet IST CALIM 35
8.1.3 Projet IST DSoS 35
8.1.4 Projet ITEA Vivian 36
8.1.5 Projet IST BRAIN 36
8.2 Actions nationales 36
8.2.1 Projet RNTL Carlit 36
8.2.2 Groupes de travail nationaux 37
8.3 Relations bilatérales internationales 37
8.4 Réseaux et groupes de travail internationaux 37
8.4.1 Cabernet 37

9 Diffusion de résultats 38
9.1 Animation de la communauté scientifique 38
9.1.1 Comités de programme 38
9.1.2 Autres responsabilités sur un plan international 39
9.1.3 Autres responsabilités sur un plan national 39
9.2 Enseignement universitaire 39
9.3 Accueil de stagiaires 40
9.4 Participation à des colloques, séminaires, invitations 40
9.5 Dépôts de brevets 41

10 Bibliographie 41

1 Composition de l'équipe

Responsable scientifique

Michel Banâtre [DR Inria]

Assistantes de projet

Fabienne Cuyollaa [adjoint administratif]

Sylvie Loubressac [adjoint administratif, UR Rocquencourt]

Personnel Inria

Gilbert Cabillic [CR, depuis le 1/10/2000]

Valérie Issarny [CR, UR Rocquencourt]

Personnel CNRS

Jean-Paul Routeau [IR]

Personnel Université

Françoise André [Professeur, université de Rennes 1]

Erwan Demairy [ATER, université de Rennes 1, depuis le 1/10/2000]

Jean-Marc Menaud [ATER, université de Rennes 1, jusqu'au 31/08/2000]

Siegfried Rouvrais [ATER, ENSSAT, depuis le 1/09/2000, co-enc. projet Lande]

Maria-Teresa Segarra [ATER, université de Rennes 1, depuis le 1/09/2000]

Frédéric Weis [MC, IUT de Rennes, Antenne de Saint-Malo]

Personnel INSA

Isabelle Puaut [MC, INSA de Rennes, délégation CNRS depuis le 1/09/2000]

Ingénieurs experts Inria

Gilbert Cabillic [jusqu'au 30/09/2000]

Pascal Chevochot

Paul Couderc [depuis le 01/10/2000]

Pascal Eloy [jusqu'au 15/09/2000]

Jean-Philippe Lesot

Pierre Tiako [depuis le 15/10/2000]

Apostolos Zarras [UR Rocquencourt]

Chercheurs doctorants

Nazim Boudeffa [bourse Inria]

Malika Boulkenafed [bourse Inria/région IdF, UR Rocquencourt, depuis 1/10/2000]

Gregory Cobena [X-Télécom, UR Rocquencourt, co-encadrement projet Verso, depuis le 1/09/2000]

Antoine Colin [bourse DGA]

Paul Couderc [bourse Inria/région Bretagne, jusqu'au 30/09/2000]

David Decotigny [bourse DGA]

Erwan Demairy [bourse MENESR, jusqu'au 30/09/2000]

Teresa Higuera [bourse Inria, UR Rocquencourt]

Christos Kloukinas [bourse Inria, UR Rocquencourt]

Frédéric Le Mouël [bourse Inria]

Frédéric Parain [bourse Inria]

Siegfried Rouvrais [bourse MENESR jusqu'au 31/08/2000, co-enc. projet Lande]

Maria-Teresa Segarra [bourse MENESR, jusqu'au 30/08/2000]

David Touzet [bourse Inria, depuis le 1/10/2000]

Arnaud Troël [bourse Inria, depuis le 1/10/2000]

Autres personnels

Viet Khoi Nguyen [poste d'accueil jeune, UR Rocquencourt, depuis le 15/12/00]

David Mentré [Post-doctorant, UR Rocquencourt, depuis le 1/12/2000]

Stéphane Tudoret [scientifique du contingent, jusqu'au 31/08/2000]

Stéphane Tudoret [poste d'accueil jeune, depuis le 1/10/2000]

2 Présentation et objectifs généraux

Un système distribué est un élément logiciel qui gère des ressources matérielles (calculateurs reliés par un réseau de communications), et qui offre une infrastructure au-dessus de laquelle différentes applications peuvent être bâties. La gestion des ressources doit être non seulement correcte, afin que le comportement des applications l'utilisant soit bien défini et en adéquation avec leurs spécifications, mais également performante, afin que les applications puissent par exemple offrir aux utilisateurs des temps de réponse raisonnables.

Ce sont les multiples aspects de la construction de systèmes distribués qu'étudie le projet Solidor. Les activités de recherche du projet Solidor s'articulent autour de deux axes :

- *La construction de systèmes distribués par spécification abstraite.* Cet axe est abordé au travers de la notion d'*architecture logicielle*, qui décrit de façon abstraite l'organisation d'un système. Cette description se fait en énonçant les propriétés des composants formant le système, et en exprimant les propriétés des interconnexions entre les composants qui interagissent. L'abstraction permet d'ignorer les détails de mise en œuvre du système, permettant ainsi de mieux en appréhender le comportement. De plus, l'abstraction permet d'utiliser des méthodes de spécification formelles, et ainsi de bénéficier des outils de vérification automatique de propriétés. Cet axe de recherche, dont les résultats sont exposés dans le paragraphe 6.1 est essentiellement exploré à Rocquencourt.

– *La construction de systèmes distribués embarqués et/ou mobiles.* Les applications utilisant ces systèmes ont une large gamme d'exigences selon leur domaine d'utilisation (par exemple contraintes de temps-réel, de consommation mémoire, de consommation électrique, de mode de traitement des déconnexions en environnement sans-fil). Il nous est donc apparu nécessaire d'étudier plusieurs types de systèmes, chacun capable de satisfaire les exigences d'une classe d'applications bien particulière. Cet axe de recherche, essentiellement exploré à Rennes, peut par conséquent être divisé en différents sous-axes, chacun d'eux étant dédié à un environnement matériel et applicatif différent :

  – *Support pour applications critiques (§ 6.2).* L'objectif est ici la construction d'environnements d'exécution pour applications distribuées ayant à la fois des contraintes de *temps-réel strict* (impératif de respect des échéances) et de *tolérance aux fautes.* Par ailleurs, de manière à limiter le coût d'achat et de maintenance de tels environnements, nous nous intéressons à l'utilisation d'éléments *sur étagères* (composants à vocation générale, non dédiés à un domaine d'applications particulier).
  – *Support pour applications multimédia sur ordinateurs de poche (§ 6.3).* L'objectif est ici de supporter, via un environnement Java, l'exécution d'applications ayant des contraintes de *temps-réel souples* (vidéo-conférence, vidéo à la demande). Le matériel visé est l'ordinateur de poche, qui possède des *ressources limitées*, tant en ce qui concerne la capacité mémoire que l'autonomie en énergie, et est le plus souvent composé de ressources *hétérogènes*.
  – *Systèmes d'information spontanés (SIS) (§ 6.4).* Cet axe traite de la conception de systèmes d'information dits "spontanés" qui se créent lorsque deux ou plusieurs ordinateurs de poche, disposant d'un système de communication "courte portée", peuvent coopérer lorsqu'ils se trouvent dans un même voisinage physique ; cette coopération s'évanouit lorsqu'ils ne se trouvent plus à portée de communication. Les principaux problèmes que nous abordons ici concernent la prise en compte de la mobilité physique des utilisateurs pour la gestion des différentes ressources. Ils traitent aussi de la découverte de l'ensemble des informations accessibles sur les calculateurs appartenant au SIS.
  – *Développement d'applications en environnement mobile (§ 6.5).* Ce dernier axe concerne le développement d'applications intégrant la mobilité au dessus de systèmes de communication sans fil. Nous nous concentrons principalement sur la proposition de schémas adaptatifs qui prennent en compte la grande variation des ressources mises en jeu pour l'exécution des applications.

3 Fondements scientifiques

Mots clés : Architectures logicielles, temps-réel, tolérance aux fautes, informatique mobile, caches.

Résumé : *Le projet Solidor étudie les différentes facettes de la construction de systèmes répartis suivant les axes des architectures logicielles et des systèmes embarqués et/ou sans fil. Les thématiques de recherche qui en découlent recoupent les recherches actuelles en systèmes temps-réel, en tolérance aux fautes, en noyaux de systèmes d'exploitation et en informatique mobile. En complément, le point de vue des architectures logicielles fait appel à une méthodologie de construction de systèmes permettant de bâtir des systèmes corrects garantissant la satisfaction de contraintes applicatives.*

Nous présentons dans les paragraphes suivants un rapide panorama des recherches actuelles dans les domaines abordés par le projet.

3.1 Architectures logicielles de systèmes

Le domaine des architectures logicielles est apparu au début des années 90 et consiste à promouvoir le développement de systèmes logiciels à partir de la définition de leur architecture [SG96]. Une architecture logicielle décrit abstraitement l'organisation d'un système en terme d'interconnexion de composants caractérisant des unités de calcul ou de stockage, *via* des connecteurs caractérisant les protocoles d'interaction entre composants. Un avantage de cette approche est qu'en faisant abstraction des détails de mise en œuvre du système, elle permet d'appréhender plus simplement le comportement du système. En particulier, elle rend viable l'utilisation de méthodes formelles pour la spécification du comportement de systèmes complexes, et des outils associés pour la vérification automatique de propriétés sur les architectures.

De nombreux travaux complémentaires sont actuellement en cours sur la spécification formelle du comportement d'une architecture. Nous pouvons les distinguer suivant les propriétés de l'architecture qui sont privilégiées : (*i*) les *propriétés invariantes de l'architecture* qui décrivent l'organisation du système, (*ii*) les *propriétés fonctionnelles* qui décrivent le comportement des composants, (*iii*) les *propriétés d'interaction* qui décrivent le comportement des protocoles de communication (ou connecteurs), et (*iv*) les *propriétés non-fonctionnelles* qui décrivent la gestion de ressources. Chaque type de propriété est détaillé ensuite.

Propriétés invariantes de l'architecture Ces propriétés permettent de décrire l'organisation du système. Spécifier les propriétés invariantes d'une architecture facilite l'implantation correcte d'une architecture par raffinement. La nature des propriétés spécifiées permet d'identifier la famille d'architectures à laquelle appartient celle du système que l'on construit. Cela permet la réutilisation de conceptions existantes lorsque l'on rencontre des architecture partageant certaines caractéristiques. Différentes méthodes de spécification sont utilisées, comme la logique du premier ordre, le formalisme Z, le modèle Cham ou encore les grammaires de graphe.

[SG96] M. Shaw, D. Garlan, *Software Architecture: Perspectives on an Emerging Disciplines*, Prentice Hall, 1996.

Propriétés fonctionnelles La spécification des propriétés fonctionnelles d'une architecture permet de vérifier la correction d'une interaction entre deux composants au regard du comportement de l'action découlant de cette interaction. Elle permet en outre de spécifier le comportement global (du point de vue des propriétés fonctionnelles) de l'architecture en fonction de celui de chacun de ses composants et de leurs interconnexions. La spécification de propriétés fonctionnelles peut aussi être exploitée pour la recherche automatique de composants à partir de leur spécification, et permet ainsi la réutilisation de logiciels. La méthode de spécification formelle utilisée dans ce cadre s'appuie sur la logique de Hoare. Remarquons ici que les propriétés spécifiées sont utilisées pour vérifier la validité de l'interconnexion de composants et non la correction d'une mise en œuvre pour une spécification donnée. Ceci rend ainsi praticable l'emploi d'outils pour une automatisation des processus de vérification ou de recherche.

Propriétés d'interaction La spécification des propriétés d'interaction d'une architecture vise les vérifications de correction de l'architecture compte tenu des protocoles de communication attendus par les composants et de ceux effectivement implantés par le système de communication utilisé. Elle permet ainsi de vérifier la validité des interconnexions de composants via les connecteurs ou encore la satisfaction de propriétés globales comme l'absence d'interblocage. L'algèbre de processus fondée sur CSP est une méthode de spécification possible. Elle permet d'utiliser l'outil FDR pour automatiser les vérifications.

Spécification de propriétés non-fonctionnelles La spécification de propriétés non-fonctionnelles a fait l'objet de très peu d'attention dans la communauté des architectures logicielles. Les quelques travaux qui abordent ce type de propriétés traitent soit les performances à l'exécution, soit la sécurité du système logiciel. Toutefois, la prise en compte des propriétés non-fonctionnelles est cruciale lorsque l'on considère la construction de systèmes (d'exploitation) distribués puisqu'il s'agit de propriétés inhérentes à ces systèmes. Aussi, les travaux de recherche du projet Solidor dans ce cadre se concentrent sur la spécification de propriétés non-fonctionnelles de différente nature. Une des méthodes de spécification proposées s'appuie sur la logique temporelle linéaire. Cette solution permet d'appréhender le raffinement d'architectures logicielles du point de vue de la réalisation de propriétés non-fonctionnelles. Par ailleurs, du fait de la forme particulière des propriétés non-fonctionnelles, nous sommes à même de fournir des outils d'aide au raffinement d'architectures et de recherche de composants implantant des fonctions de gestion de ressources pour effectivement réaliser une propriété non-fonctionnelle.

3.2 Temps-réel

De façon générale, un système temps-réel se distingue d'un système traditionnel par sa capacité à garantir que l'exécution de tâches respecte certaines échéances [Sta96]. Il existe deux grandes classes de systèmes temps-réels : les systèmes dits *temps-réel strict* et ceux dits *temps-réel souple*.

[Sta96] J. Stankovic, « Strategic directions in real-time and embedded systems », *ACM Computing Surveys 28*, 4, décembre 1996, p. 751–763.

Systèmes temps-réel strict. Les systèmes dits temps-réel strict sont ceux pour lesquels le système doit *impérativement* garantir le respect des échéances fixées pour l'exécution des tâches [But97]. Respecter impérativement des échéances est typiquement une contrainte inhérente aux applications à sûreté critique, comme le sont les applications de contrôle de centrales nucléaires ou de pilotage de fusées. Une telle contrainte signifie par exemple réagir *dans un délai maximum* à la variation d'une température au sein du cœur nucléaire.

Offrir *la garantie* que les échéances de toutes les tâches d'une application seront respectées est complexe et nécessite de respecter un ensemble de contraintes lors du développement des logiciels.

Il faut d'abord procéder à l'analyse complète des logiciels, pour déterminer les lois d'arrivée des tâches (sont-elles périodiques, apériodiques ou sporadiques), leur temps d'exécution au pire-cas (WCET ou *Worst-Case Execution Time*), l'échéance de chaque tâche et les ressources à mobiliser (mémoire, accès aux disques, etc.). Des outils (e.g. outils d'analyse statique de code pour l'obtention des WCET[PB00]) peuvent être utilisés pour assister le concepteur d'applications dans cette tâche d'analyse. Notons que ces contraintes doivent s'appliquer à tous les logiciels impliqués par l'exécution de l'application, c'est à dire non seulement l'application elle-même, mais également le système d'exploitation la supportant. Le but est de pouvoir prédire dans tous les cas de figure le comportement de toutes les parties du système (il faut donc disposer d'un système d'exploitation qui soit lui aussi *prévisible*).

Une fois l'analyse des logiciels effectuée, il est nécessaire de sélectionner un algorithme d'ordonnancement, chargé de décider de l'ordre d'exécution des tâches (par exemple EDF – *Earliest-Deadline First* – , RM[LL73] – *Rate Monotonic*). Il est alors possible d'appliquer un *test de faisabilité*. Un test de faisabilité est un test effectué hors-ligne, qui vérifie que toutes les échéances spécifiées seront respectées compte tenu de l'algorithme d'ordonnancement choisi et des propriétés sur les tâches ayant été capturées lors de la phase d'analyse. Ce test passé avec succès assure que l'application est faisable. Sinon, il faut modifier les caractéristiques de l'application (modifier les échéances par exemple), puis retester la faisabilité de l'application.

Systèmes temps-réel souple. Les contraintes temps-réel souple ont pour particularité de tolérer le non respect de certaines échéances lors de l'exécution des tâches. Cette tolérance est acceptée car les applications concernées ne relèvent pas du domaine des applications à sûreté critique. Le comportement erroné du système du point de vue des contraintes temps-réel n'a par conséquent pas d'incidence grave sur l'application. Les exemples typiques d'applications ayant des contraintes temps-réel souple sont les applications multimédias à flux de données continus. Le système doit assurer le respect de contraintes temporelles dans la délivrance des flux de données afin de garantir la qualité des images et du son. Toutefois, les contraintes imposées peuvent être adaptées puisque la qualité des données tolère une dégradation qui

[But97] G. Buttazzo, *Hard real-time computing systems: Predictable scheduling algorithms and applications*, Kluwer Academic Publishers, 1997.

[PB00] P. Puschner, A. Burns, « A Review of Worst-Case Execution-Time Analysis », *The Journal of Real-Time Systems 18*, 2, mai 2000, p. 115–128, Special issue on WCET analysis.

[LL73] C. Liu, J. Layland, « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment », *Journal of the ACM 20*, 1, janvier 1973, p. 46–61.

ne sera que faiblement perçue par l'utilisateur. Deux solutions existent pour satisfaire des contraintes temps-réel souple.

- La *réservation de ressources* où une application ayant des contraintes temps-réel effectue préalablement au lancement de l'exécution d'une tâche une réservation des ressources nécessaires à l'exécution de celle-ci pour en garantir la correction temporelle. Dans le cas où le système peut garantir la disponibilité des ressources demandées au regard de sa charge, il réserve les ressources afin que celles-ci ne soient plus utilisées pour le compte d'autres tâches et il notifie l'acceptation d'exécution à l'application. Dans le cas contraire, le système examine s'il peut toutefois exécuter la tâche en garantissant des échéances moins contraignantes. Il propose éventuellement ces échéances à l'application qui choisit ou non de lancer l'exécution de la tâche, laquelle exhibera un comportement dégradé. On parle alors de *négociation de qualité de service*.
- La *dégradation du comportement du système* compte tenu des contraintes temps-réel (plus simplement, *dégradation de qualité de service*) où le lancement de l'exécution de toute tâche ayant des contraintes temps-réel est systématique. Le système adapte ici la répartition des ressources entre les différentes tâches en fonction de sa charge globale. Dans le cas où les ressources disponibles ne permettent pas de satisfaire les contraintes temps-réel de certaines tâches, le système adapte ces contraintes de manière à pouvoir les satisfaire, ce qui conduit à un fonctionnement dégradé de l'application.

Le choix pour l'une ou l'autre solution dépend des applications et de l'environnement d'exécution. Par exemple, une application s'appuyant sur Internet reposera sur une dégradation de la qualité de service. En revanche, un service commercial de vidéo à la demande par câble devra s'appuyer sur une réservation de ressources.

Notons enfin que pour les deux solutions préconisées et à l'instar des systèmes garantissant le respect de contraintes temps-réel strict, le système implante dans tous les cas une politique d'ordonnancement des tâches qui privilégie l'exécution des tâches ayant des contraintes temps-réel, par rapport aux exécutions des autres tâches. De même, les temps d'exécution associés aux différentes fonctions du système doivent pouvoir être bornés dans le cas d'une politique de réservation de ressources.

3.3 Tolérance aux fautes

Afin de traiter les erreurs présentes dans un système informatique, trois fonctions sont distinguées[ABC+95] : (i) la *détection d'erreurs*, dont le rôle est d'identifier les états erronés comme tels ; (ii) le *diagnostic d'erreurs*, qui permet d'estimer les dommages créés par l'erreur qui a été détectée et par les erreurs éventuellement propagées avant la détection ; (iii) le *recouvrement d'erreurs*, qui permet de substituer un état exempt d'erreur à l'état erroné.

Les formes les plus couramment utilisées de mécanismes de détection d'erreur sont les codes détecteurs d'erreur, les méthodes de réplication et comparaison, les contrôles temporels et d'exécution, les contrôles de vraisemblance et les contrôles sur les données structurées.

[ABC+95] J. Arlat, J. P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J. C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, J. C. Laprie, C. Mazet, D. Powell, C. Rabéjac, P. Thévenod, *Guide de la sûreté de fonctionnement*, Cépaduès, 1995.

Les méthodes de recouvrement d'erreur reposent sur une *redondance* des données ou des calculs, par exemple par réplication de processus sur des machines différentes. Deux stratégies de réplication de processus sont utilisables :

- La *réplication active*, où chaque calcul est répliqué et exécuté simultanément sur n machines distinctes. Les copies doivent synchroniser leurs exécutions à chaque fois qu'un message est reçu ou envoyé afin d'assurer que toutes les copies réalisant un même calcul reçoivent les messages provenant d'autres processus dans le même ordre. Plusieurs modes de défaillance d'une copie existent, allant du silence sur défaillance (défaillance la plus simple) à la défaillance byzantine (défaillance la plus complexe). Dans le cas du silence sur défaillance, une copie défaillante ne produit plus aucun résultat. Disposer de n copies sur des machines distinctes permet donc d'absorber la défaillance simultanée de $n-1$ machines. Dans le cas de défaillances byzantines, une copie défaillante continue de produire des résultats mais ceux-ci sont erronés. Là, il est nécessaire de procéder à un vote sur les résultats pour élire le résultat correct produit par $\lceil (2n-1)/3 \rceil$ composants non défectueux.
- La *réplication passive*, où un composant logiciel est répliqué en n exemplaires, mais une seule des n copies effectue le calcul. Les $n-1$ autres copies sont passives et ne prennent la relève qui si la copie active est défaillante. Pour que cette stratégie fonctionne, il est nécessaire que la copie active transmette aux copies passives son état d'exécution. Cet état d'exécution est stocké par chacune des copies passives et constitue un *point de reprise*. Si la copie active est défaillante, l'une des copies passives est activée et reprend l'exécution du calcul à partir du dernier point de reprise enregistré. On dit que le processus effectue un retour arrière. Recréer un état d'exécution simule une remontée dans le temps en ramenant le calcul dans l'état qu'il occupait avant la manifestation de la défaillance.

Enfin, ces deux stratégies de réplication s'accompagnent obligatoirement d'un mécanisme d'échange *atomique* d'information entre les processus, propriété indispensable à la construction d'un système tolérant aux fautes.

3.4 Informatique mobile

L'informatique mobile pose aux concepteurs de systèmes des problèmes essentiellement liés à la prise en compte de la mobilité géographique, à la variabilité des conditions de communication et à l'énergie électrique limitée dont disposent les calculateurs portables [IK96]. Résoudre ces problèmes demande d'intervenir à de nombreux niveaux (système d'exploitation, couches chargées des communications, logiciels applicatifs).

De façon générale, les travaux en informatique mobile ont pour objet de rendre compatibles les applications et systèmes conçus dans des environnements traditionnels avec les caractéristiques particulières des environnements mobiles. Une approche vise à donner aux applications l'illusion d'un environnement stable, et tente de masquer la variabilité des performances physiques [NSN+97]. Une autre approche vise à adapter le système aux variations du contexte de

[IK96] T. Imielinski, H. Korth (éditeurs), *Mobile Computing*, Kluwer Academic Press, Boston, 1996.
[NSN+97] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, K. Walker, « Agile

l'utilisateur (position géographique par exemple). Celle-ci prône le développement d'environnement d'ubiquité informatique [Wei93]. Cependant, les prototypes actuellement développés ont une échelle souvent restreinte (couvrant un bâtiment ou un campus).

Ces approches adoptent des points de vue très différents : l'une étend les fonctionnalités des systèmes statiques à des environnements dynamiques ; l'autre exploite des informations contextuelles signifiantes pour les utilisateurs. La première cache la mobilité, la seconde l'exploite.

4 Domaines d'applications

Mots clés : Systèmes embarqués, applications mobiles, avionique modulaire embarquée, ordinateurs de poche, robotique embarquée, systèmes d'information spontanés, assistance à la conduite automobile.

Résumé : *Les activités du projet Solidor s'appliquent à des domaines différents, dans lesquels les applications ont pour principales propriétés d'être distribuées et pour la plupart embarquées. Les applications relevant de chaque domaine imposent chacune des contraintes très différentes sur le système distribué (contraintes de temps de réponse strict et de tolérances aux fautes pour les applications embarquées à sûreté critique, contraintes de mobilité pour les applications utilisant des communications sans fil, contraintes de consommation électrique pour les ordinateurs de poche).*

À chaque domaine présenté ci-après, et en fonction du degré d'avancement des recherches associées, correspond soit une action industrielle (stade avancé de recherches), soit une action de validation expérimentale.

4.1 Avionique modulaire embarquée

Mots clés : Avionique embarquée, sûreté critique, temps-réel strict, tolérance aux fautes, Hades.

Les systèmes actuels d'avionique embarquée doivent assurer des missions complexes et critiques dans des environnements agressifs. Les contraintes que doivent respecter ces systèmes et les objectifs qu'ils doivent atteindre imposent à leurs concepteurs de résoudre des problèmes combinés de tolérance aux fautes, de temps-réel strict, et de traitements distribués. La combinaison de ces problèmes entraîne souvent la mise au point de solutions spécifiques, ne fonctionnant que pour une application particulière. Chaque solution est généralement dotée de mécanismes de tolérance aux fautes matérielles ou logicielles spécifiques, et s'applique à une architecture matérielle donnée. Étant entièrement dédiée à une application, chaque solution est lourde et coûteuse.

Application-Aware Adaptation for Mobility », *in : Sixteen ACM Symposium on Operating Systems Principles*, p. 276–287, Saint Malo, France, octobre 1997.

[Wei93] M. Weiser, « Some Computer Science Issues in Ubiquitous Computing », *Communication of the ACM 36*, 7, juillet 1993, p. 75–83.

Afin de diminuer ces coûts, l'avionique modulaire embarquée tend à faire disparaître la notion d'équipement propriétaire, développé par des équipementiers spécialisés, pour lui substituer celle de composant standard intéropérable, tant matériel que logiciel, nommé communément composant *"sur étagères"* ou en anglais COTS, pour Commercial-Off-The-Shelf. Elle tend à favoriser la réutilisabilité, l'intéropérabilité et la réduction des coûts de développement.

L'action HADES, dont les résultats sont présentés dans le paragraphe 6.2 s'adresse à la construction d'exécutifs pour applications du domaine de l'avionique modulaire embarquée. Plus généralement, cette action fournit les briques de base pour l'exécution de toutes applications ayant des contraintes de temps-réel strict, de tolérance aux fautes, et de réutilisation de composants COTS (par exemple domaines de l'automobile, du spatial, de gestion de l'énergie ou de contrôle de procédés).

4.2 Ordinateurs de poche

Mots clés : Java, multiprocesseurs hétérogènes, temps-réel souple, équilibrage de charge, applications multimédias.

Aujourd'hui, l'utilisation des ordinateurs de poche *(appliances)* est cantonnée à la gestion des agendas, à la prise de notes ou au traitement du courrier électronique. Demain, l'intégration du traitement de données multimédias au sein de ces ordinateurs permettra la mise en place de nouvelles applications (vidéoconférence, *vidéo à la demande*), chargées dynamiquement à partir de fournisseurs de services. Il est clair que permettre l'exécution de telles applications nécessite de disposer, en local, d'une importante puissance de traitement. Celle-ci doit cependant rester compatible avec les contraintes liées à l'embarquabilité, en particulier pour tout ce qui à trait à la consommation électrique. C'est dans ce contexte que se place notre collaboration avec *Texas Instruments* (voir les paragraphes 6.3 et 7.1.2).

L'architecture actuellement retenue repose sur un multiprocesseur hétérogène (processeurs dédiés au traitement du signal et processeurs standard). Pour tenir compte des aspects liés au chargement dynamique des applications, ainsi qu'à leur mobilité potentielle, Java a été retenu comme langage de programmation. Dans ce cadre, nos travaux seront dédiés à l'étude de solutions pour l'exécution d'applications Java au dessus des architectures multiprocesseurs hétérogènes embarquées pour lesquelles la gestion de la consommation électrique est primordiale. Nous devrons tenir compte de l'exécution concurrente d'applications Java, dont certaines gèrent des données multimédias. Ceci suppose l'étude de solutions pour la gestion du temps-réel souple au niveau de la machine d'exécution (la *Java Virtual Machine* – JVM), ainsi que la proposition d'algorithmes de placement pour une utilisation optimale des ressources processeurs en tenant compte de la nature des applications (applications de traitement de signal, applications de calcul standard). De plus, des possibilités de traitement à distance seront offertes via des serveurs dédiés, afin de décharger le calculateur de poche de l'exécution de tâches non critiques et lui permettre d'accroître sa disponibilité par la sauvegarde, à distance, de points de reprise.

4.3 Systèmes d'information pour ordinateurs de poche en environnement mobile

Mots clés : systèmes de communication sans fil, voisinage physique, protocole de communication, systèmes embarqués, robotique embarquée, assistance à la conduite automobile.

On peut envisager deux approches pour l'utilisation des ordinateurs de poche dotés d'interface de communication sans fil (*wireless appliances*). La première consiste à s'appuyer sur des infrastructures globales pour mener à bien toute communication. Cela nécessite l'utilisation de bornes réceptrices et émettrices, fixes, géographiquement dispersées. La présence de ces bornes, et l'existence de protocoles de communication adaptés, dissimulent aux applications les problèmes liés à la mobilité, et leur donne l'illusion que le réseau délivre un service « uniforme », quelle que soit la position géographique de l'utilisateur.

Nous étudions actuellement une autre approche pour l'utilisation de ces ordinateurs de poche, complémentaire de la précédente, en nous appuyant sur des interactions dites « de proximité ». Notre démarche est la suivante : il est possible d'établir des communications directes entre entités mobiles physiquement proches, en l'absence de toute infrastructure de communication fixe (comme le sont les bornes). En s'appuyant sur la notion de proximité physique, deux calculateurs mobiles, dotés chacun d'un module de communication, et se trouvant à portée de communication, peuvent échanger directement des messages. Dès que l'échange est possible, les deux calculateurs impliqués forment un système d'information, qui présente la particularité d'être spontané. Cette spontanéité signifie que le système d'information n'existe que par la rencontre fortuite de deux calculateurs, qu'il n'existe que tant que ces deux calculateurs sont suffisamment proches l'un de l'autre, et qu'il disparaîtra une fois les calculateurs éloignés (voir la section 6.4). Des technologies de communication sans fil telles que Bluetooth doivent permettre la mise en œuvre de tels systèmes dans un futur proche.

Afin d'illustrer le principe des Systèmes d'Information Spontanés (SIS), considérons l'exemple suivant : un groupe de personnes assiste à une conférence. Plusieurs exposés se déroulent simultanément, et les participants peuvent se déplacer librement d'une session à l'autre, en fonction des sujets abordés et de leurs centres d'intérêts. Il est envisageable que dans un futur proche, chacune de ces personnes soit équipée avec un ordinateur de poche capable d'interactions de proximité. Nous appelons de tels ordinateurs des W-PDAs (Wireless Personal Digital Assistant, assistant personnel numérique sans fil). Un W-PDA est en mesure de stocker toutes sortes d'informations : l'identité de son possesseur, ces centres d'intérêt, une copie de sa présentation, des références bibliographiques, etc. Dans le contexte de la conférence, chaque rencontre physique entre deux personnes peut être perçue comme une opportunité d'échanger (de glaner) spontanément des données. Bien entendu, dans la mesure où les utilisateurs sont mobiles, ces informations ne pourront être transmises que pendant une période limitée, dont la durée n'est pas connue a priori. De plus, pour construire un tel système, il faut prendre en compte le fait que les utilisateurs ne se connaissent pas forcément au moment où la rencontre se produit. C'est à partir de ce type d'interactions que nous étudions les mécanismes nécessaires à la mise en œuvre des Systèmes d'Informations Spontanés.

En plus des systèmes de communication de personne à personne, les systèmes d'information spontanés sont applicables aux « systèmes à commandes » dans lesquels nous plaçons la robotique mobile et l'assistance à la conduite automobile. Dans ces deux cas, chaque robot ou chaque voiture dispose d'un calculateur doté d'un émetteur-récepteur, et chacun évolue dans un environnement où n'existe aucune infrastructure de communication. En robotique mobile, chaque robot est autonome et coopère avec d'autres robots physiquement proches pour accomplir ensemble une tâche commune. La formation spontanée d'un système d'information par la proximité de plusieurs robots élimine tout besoin de contrôle centralisé (comme le demandent les approches actuelles), et tend à favoriser la résistance aux défaillances et améliore la flexibilité du système. En automobile, le calculateur de chaque véhicule coopère avec ceux des véhicules voisins dans le but d'assurer la sécurité des personnes transportées : le système d'information spontané, créé par un petit groupe de voitures proches, s'échange des informations de vitesse, de freinage ou de direction, et substitue ainsi aux approches actuelles de détection passive et de reconnaissance de mouvements une approche active où les véhicules échangent des informations spontanément. De plus, dans le cas de l'automobile, aucune infrastructure de voirie n'est nécessaire.

Les bases de données peuvent également exploiter le principe des systèmes d'information spontanés. L'idée est la suivante : l'entrée de la base et la ressource qu'elle représente (par exemple une pièce en cours de fabrication) sont confondues. C'est le rapprochement des pièces, et donc la création d'un SIS, qui forme de manière temporaire, une base de données. L'éloignement d'une pièce entraîne bien entendu la sortie de la base de données. Nous estimons qu'une telle approche présente un réel intérêt dans les domaines de l'aéronautique ou de l'automobile, où les processus de fabrication impliquent une multitude de pièces, et où se posent d'importants problèmes de traçabilité. L'évolution future des étiquettes électroniques (en terme de capacité mémoire et capacité de communications) actuellement utilisées dans l'industrie pour marquer et suivre les pièces, nous permet de penser qu'une telle classe d'applications pourrait voir le jour.

5 Logiciels

Résumé : *Les travaux de recherche conduits dans le projet Solidor donnent lieu au développement de nombreux logiciels. Ces logiciels sont réalisés dans le cadre de partenariats avec des industriels, et par conséquent attachés aux domaines d'application couverts par le projet (voir paragraphe 4). Ces partenariats sont détaillés dans le paragraphe 7.1.*

5.1 Environnement pour l'adaptation systématique de middleware – Aster

Participants : Valérie Issarny [correspondante], Christos Kloukinas, Apostolos Zarras.

Dans le cadre de notre activité de recherche sur l'exploitation de la spécification d'architectures logicielles pour la conception, l'analyse et la construction de systèmes distribués, nous avons entrepris la réalisation d'un prototype d'environnement de développement d'applications distribuées. Précisément, cet environnement vise le développement de systèmes client-serveur à base de composants, et prend en charge la construction de *middlewares* adaptés aux besoins des applications, en termes des propriétés non-fonctionnelles fournies, à partir de la description architecturale des applications. L'approche mise en œuvre consiste à identifier une plate-forme *middleware* de base, ainsi que les services de gestion de la distribution complémentaires, nécessaires pour garantir les propriétés non-fonctionnelles requises par l'application. Par exemple, si l'on considère une application distribuée exigeant la mise en œuvre de propriétés transactionnelles, un *middleware* adapté est une implantation de l'ORB (*Object Request Broker*) de CORBA combinée avec les services de transaction et de verrouillage (*i.e.*, les services OTS –*Object Transaction Service*– et CCS –*Concurrency Control Service*). L'environnement comprend les éléments suivants :

- Un langage de description d'architectures (ou ADL pour *Architecture Description Language*) qui permet notamment de spécifier les propriétés non-fonctionnelles, requises ou fournies par les composants architecturaux.
- Un système de stockage, mémorisant l'historique des conceptions d'architectures *middleware*, incluant les architectures concrètes, mises en œuvre à partir de plate-formes *middleware* disponibles. La description des architectures ainsi mémorisées intègre en particulier la spécification des propriétés non-fonctionnelles fournies. Cette spécification est exprimée au moyen de la logique temporelle linéaire, ce qui permet d'organiser le stockage des architectures suivant une relation de raffinement entre les propriétés qu'elles fournissent, mais aussi d'offrir des fonctions pour la recherche et l'ajout systématiques d'architectures dans le système. Nous utilisons le démonstrateur de théorèmes STeP de Stanford, pour la mise en œuvre de ces différentes fonctionnalités.
- Un outil pour l'intégration automatique d'une architecture *middleware* sélectionnée, au sein d'une application, à partir de la description architecturale de cette dernière.

Notre environnement est indépendant de toute plate-forme *middleware* et peut par conséquent être exploité pour des infrastructures comme CORBA, *Enterprise Java Beans* ou encore DCOM. Toutefois, nous avons principalement centré nos expérimentations autour de l'infrastructure CORBA, ce qui nous a conduits à développer des services *middleware* particuliers mettant en œuvre des propriétés de tolérance aux fautes. L'environnement Aster fait l'objet de divers enrichissements liés à l'expérimentation de nos résultats de recherche dans le domaine de la conception, analyse et construction de systèmes logiciels distribués à partir de leur description architecturale. Nous nous intéressons notamment à l'introduction de méthodes et outils pour la combinaison systématique d'architectures *middleware* offrant différentes propriétés non-fonctionnelles. Nous examinons par ailleurs l'extension de l'environnement Aster afin de permettre l'analyse quantitative de propriétés de fiabilité et de performance d'architectures de systèmes distribués, en nous appuyant sur des outils de modélisation existants.

5.2 Environnement d'exécution pour l'avionique embarquée – Hades

Participants : Michel Banâtre, Pascal Chevochot, Antoine Colin, David Decotigny, Stéphane Tudoret, Isabelle Puaut [correspondante].

Les aspects contractuels liés à ce logiciel sont détaillés dans le paragraphe 7.1.1.

Les travaux réalisés dans le cadre de l'action HADES, visant à développer un environnement d'exécution pour applications critiques (à temps de réponse strict et en fort besoin en tolérance aux fautes) (voir paragraphe 6.2) font l'objet d'une expérimentation *via* la réalisation d'un prototype. Ce prototype comporte d'une part un environnement dédié au développement et l'analyse des applications, et d'autre part un environnement permettant l'exécution de ces applications. Les applications visées par le prototype, de par le contexte de partenariat avec la DGA et la société Dassault-Aviation (cf. § 7.1.1), sont des applications d'avionique embarquée, ayant les propriétés d'être distribuées, de comporter des contraintes de temps-réel strict et de haute disponibilité.

L'environnement de développement d'applications comprend un ensemble d'outils exécutés hors-ligne (i.e. avant exécution des applications). Ces outils incluent :

- Un environnement graphique de développement d'applications, permettant de saisir les applications sous la forme de graphes orientés acycliques ;
- Des tests de faisabilité permettant de tester le respect des échéances temporelles des applications en considérant le pire scénario de charge possible [16],
- Un outil de réplication automatique d'applications permettant de rendre les applications tolérantes aux fautes avec le minimum d'intervention du concepteur d'applications [36]. Cet outil transforme les graphes des applications en répliquant (totalement ou partiellement) leurs traitements, et ceci selon différentes méthodes de réplication (réplication active, passive, semi-active et temporelle)
- Un outil d'analyse des temps d'exécution au pire-cas de programmes [6], permettant par analyse statique de code C de donner leur pire temps d'exécution, en prenant en compte les propriétés de l'architecture pour éviter le pessimisme de l'analyse. Cet outil utilise l'outil Salto, développé par le projet CAPS, pour obtenir les informations concernant l'architecture interne du processeur.

L'environnement d'exécution (ou support d'exécution) est une couche logicielle *middleware* qui s'appuie sur un micro-noyau temps réel sur étagères (à ce jour, le support d'exécution a été porté sur les noyaux Chorus et RTEMS). Ce support d'exécution [36] offre :

- Un ensemble de services pour l'exécution d'applications distribuées temps-réel dans un environnement sujet aux défaillances. Ces services incluent notamment un service de communication fiable temps-réel, un service d'ordonnancement des processus, un service de gestion de groupes et un service de synchronisation d'horloges. Par ailleurs, le support d'exécution définit un ensemble de mécanismes de détection d'erreurs permettant d'assurer que chaque calculateur soit fonctionne correctement, soit s'arrête de manière silencieuse (silence sur défaillances).
- Un outillage pour l'examen du comportement du support d'exécution. En particulier, un logiciel d'injection de fautes a été développé pour évaluer la couverture de l'hypothèse de silence sur défaillances.

L'architecture logicielle du support d'exécution est conçue de manière à pouvoir remplacer tout service par un service ayant la même interface mais ayant des propriétés ou une implémentation différente, ce qui permet notamment de faciliter le portage du support d'exécution sur de nouveaux noyaux temps-réel.

Une application du domaine de l'avionique (conduite de tir de missiles) a été portée avec succès sur le support d'exécution.

5.3 Environnement d'exécution Java pour architecture embarquée

Participants : Michel Banâtre, Gilbert Cabillic [correspondant], Valérie Issarny, Frédéric Parain, Teresa Higuera, Jean-philippe Lesot, Jean-Paul Routeau, Pierre Tiako, Pascal Eloy.

Le contrat correspondant est détaillé dans le paragraphe 7.1.2.

L'objectif de ces développements est de concevoir et réaliser un environnement d'exécution Java qui puisse prendre en compte les contraintes d'embarquabilité liées à un *appliance* (i.e. ordinateur de poche disposant d'un moyen de communication sans fil). Ce type de matériel est plus particulièrement caractérisé par le fait d'une capacité mémoire limitée et d'une autonomie d'énergie limitée. Vu de l'utilisateur, un appliance permet l'exécution d'applications différentes telles qu'un agenda, un accès Internet, de la téléphonie, mais aussi des applications multimédias (reconnaissance vocale, lecture de sons ou de séquences vidéos diffusées, vidéoconférence, etc).

Cet environnement permet l'exécution de différents types d'applications, en offrant un ensemble d'APIs Java tout en tenant compte des contraintes d'embarquabilité d'un appliance. En outre, il est conçu pour s'exécuter sur différents types de systèmes en définissant une couche d'abstraction du matériel qui permette de le porter d'un système à l'autre. Ainsi, le système temps-réel VxWorks de la société WindRiver est supporté, mais également des systèmes d'exploitation plus généralistes comme Windows ou Linux.

Enfin, les aspects liés à la performance de cet environnement d'exécution sont traités. Notre approche est d'avoir décomposé en modules l'ensemble de l'environnement d'exécution Java. Cette décomposition modulaire permet tout d'abord de comprendre les problèmes réels de performance d'un module et ensuite de pouvoir se concentrer sur les améliorations à apporter sur ce module. Afin de réaliser cette modularité, nous avons été amenés à écrire la totalité de notre environnement d'exécution Java.

6 Résultats nouveaux

Résumé : *Ce paragraphe présente les résultats des recherches menées par le projet Solidor au cours de l'année 2000. Ces recherches correspondent aux activités ayant été brièvement introduites dans le paragraphe 2.*

6.1 Architectures logicielles de systèmes distribués

Participants : Malika Boulkenafed, Gregory Cobena, Erwan Demairy, Valérie Issarny, Viet Khoi Nguyen, David Mentré, Christos Kloukinas, Siegfried Rouvrais, Apostolos Zarras.

Comme indiqué dans la section 3.1, la description de l'architecture d'un système logiciel permet de s'abstraire des détails d'implémentation de bas niveau et ainsi de mieux appréhender la conception et la construction de systèmes logiciels complexes. En particulier, une telle approche à la mise en œuvre de systèmes distribués rend possible l'utilisation de méthodes formelles pour l'analyse et la conception de ces systèmes.

Dans le cadre des activités de recherche du projet SOLIDOR s'effectuant à l'UR de Rocquencourt, nous nous intéressons à l'exploitation de la description d'architectures logicielles (*i.e.*, définition de notations, et de méthodes et outils associés), ou plus généralement à la spécification déclarative de systèmes, pour mécaniser l'analyse, la conception et la construction de systèmes distribués de bas niveau, *i.e.*, les systèmes logiciels implantant des propriétés non-fonctionnelles pour les applications, tels que les *middleware*. Jusqu'ici, nous nous sommes principalement intéressés aux architectures de systèmes client-serveur à base de composants, et, plus récemment, aux architectures de sites Web.

Nous étudions également la définition de nouvelles architectures logicielles de systèmes distribués, liées à l'utilisation croissante de l'Internet depuis des terminaux de différentes capacités, incluant les ordinateurs de poche sans fil. Les architectures des systèmes distribués visés doivent en particulier permettre d'offrir une qualité de service acceptable aux utilisateurs tout en tenant compte des ressources relativement limitées de certains des terminaux utilisateurs, de la variation de la bande passante disponible, et de la charge globale du système. Nous donnons ci-après un récapitulatif de nos résultats de recherche ainsi qu'un aperçu de nos futurs travaux, dans le domaine des architectures logicielles de systèmes distribués.

6.1.1 Conception d'architectures de systèmes

La description d'architectures logicielles de systèmes repose sur trois éléments principaux : (*i*) les composants caractérisant abstraitement les unités de calcul ou de stockage du système, (*ii*) les connecteurs caractérisant abstraitement les protocoles d'interaction entre composants et (*iii*) les configurations décrivant les structures de (sous-)systèmes en termes de compositions d'instances de composants *via* des instances de connecteurs. Les notations utilisées pour les spécifications de ces différents éléments dépendent alors de l'exploitation visée de l'architecture (*e.g.*, analyse, construction). L'essentiel est de maintenir un niveau d'abstraction élevé afin de faciliter l'appréhension du comportement du système à partir de la description de son architecture. En général, la description d'architectures est abordée d'un point de vue fonctionnel, mettant en avant les fonctionnalités des composants du système et les protocoles d'interaction exploités pour leur composition. Toutefois, la mise en œuvre de propriétés non-fonctionnelles est tout aussi fondamentale. Une caractéristique intéressante de ces dernières propriétés est qu'elles peuvent souvent être implantées en réutilisant des "services systèmes" existants. Il est donc possible de considérer une spécification abstraite des propriétés non-fonctionnelles, pouvant être exploitées pour mécaniser l'intégration des services nécessaires à leur satisfaction, au sein des architectures d'applications. Cette problématique motive un certain nombre de nos travaux, présentés dans les paragraphes suivants. Néanmoins, il est également nécessaire de prendre en compte l'intégration de solutions à la mise en œuvre de propriétés non-fonctionnelles, qui sont spécifiques à l'application. Ces solutions peuvent alors être soit masquées au niveau des éléments de l'architecture (*e.g.*, implantées au sein des composants), soit implantées au niveau architectural dans le cas où la configuration du système se voit affectée. Cette dernière considération s'applique notamment au cas de la tolérance aux fautes où l'on peut envisager une reconfiguration du système suite à l'occurrence d'exceptions. En collaboration avec Jean-Pierre Banâtre, nous avons proposé une première solution au traitement d'exceptions au niveau architectural. Cette proposition complémente le traitement d'exceptions interne aux éléments de l'architecture, et facilite la réutilisation de composants logiciel [22]. Plus généralement, nous nous intéressons au problème de la mise en œuvre de solutions de tolérance aux fautes, spécifiques aux applications, au niveau architectural.

6.1.2 Analyse des propriétés non-fonctionnelles des systèmes distribués

Dans le cadre de l'analyse de systèmes distribués à partir de leur description architecturale, notre travail s'est d'abord orienté sur l'analyse des propriétés non-fonctionnelles des architectures de systèmes, d'un point de vue qualitatif. Notre objectif est ici de promouvoir la réutilisation de solutions architecturales existantes pour la mise en œuvre d'une propriété non-fonctionnelle donnée. Succinctement, notre approche repose sur une description des architectures logicielles de systèmes intégrant la spécification abstraite des propriétés non-fonctionnelles fournies, ce qui permet de définir une relation de raffinement entre les architectures, du point de vue de la satisfaction de ces propriétés. Cette relation de raffinement est alors exploitée pour mécaniser la sélection d'architectures répondant aux spécificités d'un système donné, à partir des conceptions antérieures d'architectures. Notre solution s'appuyait initialement sur une spécification des propriétés au moyen de la logique temporelle linéaire. Nous examinons à présent une solution qui soit plus aisément appréhendable par les concepteurs de systèmes. Nous étudions en particulier la combinaison de notre solution avec le développement de systèmes s'appuyant sur UML.

Depuis plus récemment, nous nous intéressons à l'analyse des propriétés non-fonctionnelles de systèmes au niveau architectural, d'un point de vue quantitatif. Ceci permet notamment de dimensionner correctement le système au regard de l'infrastructure sous-jacente. Notre travail de cette année s'est orienté vers l'analyse quantitative de la fiabilité et des performances de systèmes distribués [35]. Ceci nous a conduits à proposer une solution à la spécification de propriétés de performance et de fiabilité au sein de la description d'architectures. Les architectures ainsi définies sont ensuite traduites en des modèles de systèmes pouvant être exploités par des outils existants d'évaluation de performance et de fiabilité. Du fait de l'utilisation croissante d'UML pour le développement de systèmes, nous nous appuyons sur une modélisation de l'architecture au moyen de cette notation. Nous examinons par ailleurs l'application de notre méthode au développement de systèmes d'information distribués.

6.1.3 Synthèse d'architectures middleware pour systèmes client-serveur à base de composants

Nos travaux dans le domaine de l'analyse qualitative d'architectures de systèmes distribués du point de vue des propriétés non-fonctionnelles fournies, ont mis en avant qu'il était possible d'identifier des solutions architecturales réutilisables pour les applications. Afin de permettre une exploitation pratique de ces résultats, nous avons entrepris, dès l'origine de nos travaux dans le domaine des architectures logicielles pour la construction de systèmes distribués, la conception et l'implantation d'un environnement de développement de systèmes distribués intégrant nos résultats les plus récents. L'environnement proposé s'appuie sur la technologie *middleware*, qui définit une couche logicielle qui se situe entre l'application et le système d'exploitation. Une infrastructure *middleware* (*e.g.* CORBA, DCOM, *Enterprise Java*

Beans) fournit des solutions réutilisables aux problèmes traditionnellement rencontrés dans la construction de systèmes distribués comme l'hétérogénéité, l'interopérabilité, la sécurité, la tolérance aux fautes ou encore les transactions. En général, une infrastructure *middleware* comprend un système de communication de base (ou *broker*) et un ensemble de services implantant des propriétés non-fonctionnelles évoluées comme celles mentionnées ci-avant. Toutefois, si la construction d'un système distribué à partir d'une infrastructure est simplifiée du fait des solutions réutilisables offertes, la conception du système pour répondre au besoin d'une application particulière reste une tâche complexe nécessitant la maîtrise des différents éléments des infrastructures disponibles. L'environnement de développement que nous fournissons propose une solution à ce problème [4]. Cet environnement s'appuie sur un langage de description d'architectures et un ensemble d'outils associés pour : (*i*) la conception d'architectures *middleware* qui satisfont les exigences des applications, (*ii*) l'intégration d'une architecture *middleware* adaptée à une application donnée, au sein de l'architecture de cette application, et (*iii*) maintenir cette architecture *middleware* au regard des évolutions relatives aux exigences de l'application ou des services *middleware* disponibles.

De manière générale, la conception d'une architecture *middleware* répondant aux exigences d'une application requiert de combiner des architectures offrant des propriétés non-fonctionnelles distinctes (*e.g.*, exigences de sécurité et de tolérance aux fautes). Dans ce cadre, guider le développement du système nécessite de mettre en évidence des compositions d'architectures *middleware* valides. Nous avons proposé une première étape de solution à ce problème ; celle-ci s'appuie sur une composition d'architectures *middleware* au niveau structurel afin d'en limiter la complexité [26]. Nous examinons à présent l'enrichissement de cette solution de manière à supporter l'analyse des compositions d'architectures *middleware* qui sont valides d'un point de vue structurel, et ainsi faciliter l'identification de l'architecture la mieux adaptée à l'application visée.

Nous avons jusqu'ici indiqué des exemples connus d'infrastructures *middleware*. Afin d'être pleinement exploitable, notre approche doit également pouvoir s'accommoder des infrastructures offrant des fonctionnalités avancées. Dans ce cadre, nous avons examiné, en collaboration avec Gordon et Lynne Blair de L'Université de Lancaster (GB), l'utilisation de notre environnement pour l'adaptation dynamique de *middleware*, étant donnée une plate-forme offrant des fonctionnalités de réflexivité [15]. Parmi les plate-formes middleware émergentes, nous identifions également celles supportant l'exécution d'agents mobiles. Toutefois, l'intérêt d'utiliser des agents mobiles plutôt que des modes d'interaction classiques comme l'appel de procédure à distance dépend en grande partie des propriétés non-fonctionnelles du système (*e.g.*, propriétés de sécurité des différents acteurs). En collaboration avec Pascal Fradet du projet LANDE (UR Rennes), nous avons entrepris une étude sur la caractérisation abstraite des interactions entre les entités du système distribué, qui sont ensuite raffinées en des modes d'interaction concrets supportés par la plate-forme middleware sous-jacente (*e.g.*, agent mobile, appel de procédure à distance) au regard des propriétés non-fonctionnelles du système global et des éléments le constituant (*e.g.*, performance, sécurité) [20].

6.1.4 Synthèse d'architectures de sites Web

La spécification déclarative de systèmes pour simplifier leur développement fait l'objet de différents travaux dans le domaine de la production de sites Web mettant à disposition de grands volumes de données extraits dynamiquement de bases de données. Dans ce contexte, la spécification de la structure et du contenu du site est séparée de celle de sa représentation graphique. La structure et le contenu d'un site Web sont alors définis au moyen d'un modèle logique (en général, un graphe) et la correspondance de ce modèle avec les données de la base de données est spécifiée *via* un langage déclaratif. Ensuite, étant données cette spécification du site Web ainsi que celle de sa représentation graphique, il est possible d'automatiser la génération du site Web à partir de composants existants (*e.g.*, générateur de pages HTML à partir de fragment XML et de feuilles de style XSL). Cette approche, notamment examinée au sein du projet CARAVEL (UR Rocquencourt), a pour avantage de simplifier le développement mais aussi la maintenance des sites Web.

Toutefois, les sites Web considérés ont comme inconvénient de souvent offrir des temps de réponses peu acceptables à l'utilisateur, en partie du fait des interactions avec le système de gestion de base de données. En collaboration avec Daniela Florescu, Khaled Yagoub et Patrick Valduriez du projet CARAVEL, nous avons étudié une solution à la spécification déclarative de sites Web, supportant la génération de sites offrant de bonnes performances aux utilisateurs. Notre solution repose sur un système de gestion de la matérialisation des données, permettant de mettre en caches des résultats de requêtes au niveau du système de gestion de base de donnée, des fragments XML et des pages HTML. La gestion de ces différents caches est alors spécialisée en fonction de la spécification du comportement attendu du site Web d'un point de vue non-fonctionnel, *e.g.*, taux de mise à jour, taux d'accès [34]. Les résultats de ces travaux ont fait l'objet d'une expérimentation *via* le développement du système Weave pour la gestion de sites Web [33]. Le système Weave permet la spécification déclarative des sites Web, incluant des informations non-fonctionnelles relatives au site. Cette spécification est ensuite exploitée pour générer le site Web et notamment spécialiser la gestion des caches. Différentes perspectives sont envisagées pour l'évolution du système Weave ; indiquons notamment le support de la mise à jour de la base par les clients du site Web, l'accès aux sites Web depuis des terminaux de différentes capacités, ou encore l'adaptation de la politique de matérialisation des données au regard du comportement effectif du site Web par opposition à la spécification de ce comportement par le développeur.

6.1.5 Nouvelles architectures de systèmes distribués

L'utilisation croissante de l'Internet, celle attendue des ordinateurs de poche sans fil, ainsi que les évolutions technologiques en matière de réseaux conduisent à une évolution significative des systèmes distribués. La notion d'ubiquité informatique qui était déjà d'actualité au début des années 90, devient une considération majeure dans le domaine des systèmes distribués. Dans ce contexte, l'architecture du système distribué doit pouvoir être composée dynamiquement, en intégrant notamment les différents acteurs impliqués et les ressources à leur disposition. La conception d'architectures de systèmes distribués supportant la notion d'ubiquité fait l'objet d'une partie de nos travaux de recherche depuis cette année. Nos études sur la construction de systèmes à partir de la spécification de leur architecture, présentées dans les paragraphes précédents, font partie intégrante des bases de solutions au problème visé. La composition dynamique d'architectures de systèmes requiert une mécanisation que permet *a priori* une approche reposant sur une spécification déclarative du système. Toutefois, il est également nécessaire que la composition des architectures de base des systèmes distribués soit adaptée à la mécanisation. Par exemple, les architectures *middleware* des systèmes client-serveur traditionnels sont peu adaptées à ce processus de composition. Par ailleurs, ces architectures n'intègrent pas, en général, l'existence de variations significatives tant au niveau des différentes machines interagissant qu'au niveau du réseau d'interconnexion. Nous avons axé nos premiers travaux sur les aspects précités, sur la définition d'architectures *middleware* intégrant des sites de différentes capacités et notamment des ordinateurs de poche sans fil. Nous considérons par ailleurs le couplage de cette étude avec nos travaux sur la définition des architectures de sites Web, lesquels constituent des sites d'accès privilégiés pour les utilisateurs quel que soit le terminal qu'ils ont à leur disposition.

6.2 Exécutifs pour applications temps-réel strict hautement disponibles

Participants : Nazim Boudeffa, Michel Banâtre, Pascal Chevochot, Antoine Colin, David Decotigny, Isabelle Puaut.

Nous nous intéressons ici à la construction d'environnements d'exécution pour applications distribuées ayant à la fois des contraintes de *temps-réel strict* (impératif de respect des échéances) et de *tolérance aux fautes*. Par ailleurs, de manière à limiter le coût d'achat et de maintenance de tels environnements, nous nous intéressons à l'utilisation d'éléments, tant matériels que logiciels, sur étagères (en anglais COTS, pour Commercial-Off-The-Shelf), c'est à dire des composants à vocation générale, non dédiés à un domaine d'applications particulier.

Nous orientons la présentation des résultats de nos recherches de l'année 2000 selon les deux classes de contraintes exhibées par les applications visées, à savoir les contraintes de tolérance aux fautes et de temps-réel strict, le dénominateur commun entre ces deux points étant l'utilisation de composants sur étagères. Une vue d'ensemble de ces résultats pourra être trouvée dans les articles [17, 36].

Notons que dès l'origine de nos travaux sur les exécutifs pour applications temps-réel strict hautement disponibles, nos résultats de recherche ont été intégrés dans un prototype nommé par la suite Hades (acronyme de *Highly Available Distributed Embedded System*), dont la construction a été soutenue par la DGA et la société Dassault-Aviation (cf.§ 7.1.1). Nous ne détaillons pas plus avant ce support d'exécution, ce dernier étant présenté dans le paragraphe 5.2.

6.2.1 Aspects tolérance aux fautes

Les noyaux temps-réel sur étagères n'intègrent pas en général de mécanismes de tolérance aux fautes (détection ou recouvrement d'erreurs). Par conséquent, ils doivent être enrichis avec de tels mécanismes, augmentant ainsi la complexité des supports d'exécution résultants. De ce fait, montrer que les échéances des applications seront toujours respectées dans un tel support d'exécution n'est pas une tâche aisée. Notre travail de l'année dans ce cadre à consisté à étendre l'analyse d'ordonnancement distribuée holistique de Tindell et Clark[TC94] pour pouvoir prendre en compte des systèmes plus complexe qu'à l'origine (présence d'appels entre tâches, de priorités identiques pour plusieurs tâches). Les résultats de ce travail sont relatés dans [16].

Par ailleurs, de nombreux mécanismes de recouvrement d'erreurs, comme par exemple le recouvrement arrière, supposent des calculateurs à *silence sur défaillance* (qui soit fonctionnent correctement, soit s'arrêtent sans produire de résultats incorrects). Les composants sur étagères, tant au niveau matériel qu'au niveau logiciel, ne sont généralement pas silencieux sur défaillances. C'est pourquoi nous avons défini un ensemble de *mécanismes de détection d'erreurs* [36], permettant de détecter des erreurs (temporelles et par valeur) et d'arrêter le système dès qu'une erreur est détectée. Ces mécanismes sont entièrement implantés par logiciel, sous la forme d'une couche logicielle intermédiaire intercalée entre un système d'exploitation temps-réel sur étagères et le code applicatif. En outre, ils ont un coût borné en terme de temps d'exécution, de manière à être compatibles avec des contraintes de temps-réel strict. L'efficacité de ces mécanismes (couverture de l'hypothèse de silence sur défaillances) a été évaluée, en utilisant une technique expérimentale : l'injection de fautes. Un injecteur de fautes, opérant par insertion d'erreurs dans l'état du système (injection logicielle) a été réalisé pour les besoins de l'évaluation. Les résultats de l'évaluation montrent une couverture de 80% de l'hypothèse de silence sur défaillances quand aucun mécanisme de détection d'erreurs n'est utilisé (autre que ceux du processeur ou du système d'exploitation). Cette couverture dépasse 99% quand tous les mécanismes de détection d'erreurs sont utilisés [37].

6.2.2 Aspects temps-réel strict

Évaluation de temps d'exécution au pire cas par analyse statique. La connaissance des temps d'exécution *au pire cas* des tâches est nécessaire pour leur prise en compte dans des tests de faisabilité (des informations probabilistes sur les temps d'exécutions ne sont pas adaptées). Nous nous intéressons ici à obtenir de tels temps par analyse statique de programmes, et examinons dans quelle mesure une telle approche est compatible avec l'utilisation de matériel et logiciel sur étagères. Les contributions de notre approche, détaillées ci-dessous, sont d'une part d'intégrer dans un outil d'analyse statique, une modélisation fine de la structure interne du processeur, de manière à obtenir des temps d'exécution au pire-cas qui soient le moins sur-évalués possible, et d'autre part d'appliquer l'analyse statique de programmes sur le code de systèmes d'exploitation.

Une des difficultés dans le domaine de l'analyse de temps d'exécution au pire cas de programmes est d'éviter l'obtention de temps qui soient une estimation trop pessimiste des temps d'exécution effectifs, ce qui entraînerait alors une sur-estimation des ressources matérielles nécessaires à l'exécution des programmes, et donc une sous-utilisation de ces ressources lors de l'exécution. Les processeurs actuels incluent des mécanismes élaborés (pipelines, caches, prédiction de branchement), dont le but est d'améliorer les temps d'exécution moyens des programmes. Ces mécanismes doivent être pris en compte lors de l'analyse de temps d'exécution au pire-cas pour réduire le pessimisme de l'analyse. Notre contribution concernant la prise en compte de l'architecture interne des processeurs a porté sur la modélisation et la prise en

[TC94] K. Tindell, J. Clark, « Holistic Schedulability Analysis for Distributed Hard Real-Time Systems », *Microprocessors and Microprogramming 40*, 1994, p. 117–134.

compte au niveau de la méthode d'analyse du mécanisme de prédiction de branchement, en particulier pour un processeur Pentium [6].

Un outil d'analyse de temps d'exécution au pire-cas de programmes C ayant comme cible un processeur Pentium, et nommé HEPTANE (pour *Hades Embedded Processor Timing ANalyzEr*) a été développé (voir paragraphe 5.2). Il prend en compte les mécanismes de caches d'instruction, de pipeline et de prédiction de branchement afin de réduire le pessimisme de l'analyse. HEPTANE utilise l'outil Salto, développé par le projet CAPS, pour obtenir les informations concernant l'architecture interne du processeur.

Nous avons utilisé une méthode d'analyse statique (via l'outil Heptane) pour obtenir les temps d'exécution au pire-cas d'un noyau temps-réel sur étagères, le noyau RTEMS. Ce travail [CP99] montre que les restrictions sur le langage source imposées par les outils d'analyse statique sont, tout du moins en ce qui concerne le noyau RTEMS, tolérables. Par ailleurs, cette expérience a permis de mettre en exergue : (i) des composants du système d'exploitation dont le temps d'exécution au pire-cas dépend des conditions d'exécution (contexte d'appel des fonctions, arrivée d'interruptions) (ii) des composants ayant des temps d'exécution au pire-cas très longs et très différents de leur temps d'exécution moyen, à cause de l'utilisation d'algorithmes non adaptés aux contraintes temps-réel strict (algorithmes d'allocation dynamique de mémoire notamment). Pour solutionner le point (i), nous appliquons actuellement sur le code de RTEMS un outil d'évaluation partielle (l'outil Tempo, développé par le projet Compose) au préalable à l'analyse de temps au pire-cas, et évaluons les gains obtenus.

Prise en compte de conditions opérationnelles évolutives. Dans le cadre des systèmes temps-réel strict, la démarche usuelle est de définir de manière statique l'environnement et la composition du système lors de sa conception, afin de garantir toutes les échéances avant exécution. Ceci conduit à un sur-dimensionnement du système pour parer aux conditions de fonctionnement extrêmes. Le présent travail vise à terme à définir une extension de ces systèmes selon laquelle plusieurs hypothèses de fonctionnement peuvent être supportées par le même système (hypothèses sur les lois d'arrivée d'événements extérieurs et les durées d'exécutions des tâches, hypothèses de défaillance). En fonctionnement, le système doit être capable d'identifier l'hypothèse de fonctionnement valable dans le contexte d'exécution courant, et éventuellement d'initier une reconfiguration quand l'hypothèse de fonctionnement n'est plus valide. L'objectif est ici de garantir des caractéristiques minimales strictes pour le service rendu, tout en permettant de les améliorer quand les ressources disponibles l'autorisent. Afin d'atteindre cet objectif, nous construisons actuellement une infrastructure de simulation qui permettra d'évaluer plusieurs mécanismes système existants. Ces mécanismes proposent tous de superviser de façon dynamique la configuration du système en réponse aux évolutions du comportement de l'environnement. Pour l'évaluation de ces mécanismes, il s'agit de définir quelles sont les métriques de comparaison, et quels sont les échantillons de test permettent d'obtenir des résultats pertinents pour les comparer (échantillons statistiques ou utilisation de traces réelles d'exécution). Les résultats obtenus seront confrontés à ceux obtenus avec une réalisation du mécanisme que nous proposerons.

[CP99] A. COLIN, I. PUAUT, « Worst-Case Timing Analysis of the RTEMS Real-Time Operating System », *rapport de recherche n° 1277*, IRISA, novembre 1999, Soumis à publication.

Caractérisation du comportement temporel de composants sur étagères. Utiliser un système d'exploitation temps-réel sur étagères pour construire un système temps-réel critique nécessite de connaître le temps de réponse de ses appels système dans le pire scénario d'exécution possible. Deux classes de méthodes existent pour obtenir de tels temps de réponse: (i) par analyse statique du code source du système d'exploitation ; (ii) par exécution de jeux de tests (benchmarks) permettant de mesurer dans des conditions de test particulières le temps de réponse du système d'exploitation. Bien que la première approche soit plus sûre, elle nécessite de disposer du code source du système d'exploitation. D'autre part, pour obtenir des temps de réponse qui ne sont pas sur-dimensionnés, elle nécessite une connaissance fine du comportement temporel du processeur utilisé, ce qui est de moins en moins facile à obtenir étant donnée la complexité grandissante des processeurs actuels. La deuxième approche ne présente pas ces inconvénients, mais en revanche pose le problème d'identifier les conditions provoquant le pire temps de réponse du système. Nous débutons actuellement une étude visant à explorer cette deuxième voie en générant les jeux de tests de manière à ce qu'ils soient le plus représentatifs possible du pire scénario d'exécution, et ceci en restant autant que possible indépendant d'un noyau temps-réel et d'une architecture matérielle particuliers.

6.3 Environnement Java embarqué pour ordinateur de poche

Participants : Michel Banâtre, Gilbert Cabillic, Teresa Higuera, Valérie Issarny, Jean-Philippe Lesot, Frédéric Parain, Pierre Tiako.

Ces travaux s'inscrivent dans le cadre de la conception et la réalisation d'un environnement d'exécution Java embarqué adapté à un ordinateur de poche disposant d'un moyen de communication sans fil [25]. Cet environnement doit tout d'abord prendre en compte les contraintes liées à l'architecture matérielle qui est dans notre cas un multiprocesseur hétérogène à mémoire partagée. L'architecture possède également d'autres contraintes : une capacité mémoire limitée, une capacité de traitement limitée par processeur, et une autonomie d'énergie limitée.

L'utilisation de ces ordinateurs de poche sans fil, aujourd'hui cantonnée à la gestion des agendas, à la prise de notes ou au traitement du courrier électronique est en pleine évolution. Demain, l'intégration du traitement de données multimédias au sein de ces ordinateurs permettra la mise en place de nouvelles applications, telles que la vidéo-conférence ou la vidéo à la demande, chargées dynamiquement à partir de fournisseurs de services.

Nos travaux actuels se focalisent sur trois axes de recherches :

- la conception et le développement de la machine d'exécution Java modulaire distribuée Scratchy,
- la proposition d'une stratégie d'ordonnancement multi-critère d'applications multimédias écrites en langage Java conciliant la gestion du temps-réel souple et la minimisation de la consommation énergétique,
- la définition d'une technique de ramasse miette adaptée aux applications multimédias.

Nous décrivons dans la suite de ce paragraphe les résultats obtenus autour de ces différents axes.

6.3.1 Machine d'exécution Java modulaire Scratchy

Les travaux menés autour de la machine d'exécution Scratchy doivent apporter une solution à plusieurs problèmes décrits ci-après. Tout d'abord, ils doivent permettre de comprendre les problèmes de performances de la machine d'exécution Java liés à l'interprétation de bytecode. Puis, il faut pouvoir supporter l'exécution d'une application Java au-dessus d'une architecture multiprocesseur et ce, de manière transparente pour le concepteur de l'application Java. Enfin, cette machine d'exécution doit pouvoir s'exécuter sur différents types de processeurs et différents systèmes d'exploitation temps-réel.

Comme nous l'avons indiqué dans le paragraphe 5.3, notre approche est d'avoir décomposé de manière modulaire l'ensemble de l'environnement d'exécution Java afin de pouvoir travailler indépendamment sur chaque module. Afin de réaliser cette modularité nous avons été amenés à écrire la totalité de notre environnement d'exécution Java. Nous avons également défini un module permettant d'interfacer notre environnement d'exécution Java avec de nombreux systèmes temps-réel.

6.3.2 Ordonnancement multi-critère d'applications multimédias

Afin de permettre l'ordonnancement d'applications multimédias qui puisse concilier la gestion du temps-réel souple et la minimisation de la consommation énergétique, nous avons été amenés à introduire tout d'abord un modèle d'application multimédia permettant d'exprimer les contraintes temporelles souples des applications écrites dans le langage Java. Ensuite, nous avons défini une politique d'ordonnancement multi-critère compatible avec ce modèle. Enfin, afin d'obtenir les informations propres à chaque application nécessaires à la réalisation de la politique d'ordonnancement (estimation du temps d'exécution et estimation de la consommation énergétique des traitements temps-réel d'une application), nous examinons la conception d'une technique de caractérisation (*"profiling"*) d'applications multimédia.

Nous décrivons successivement ces trois points dans la suite de ce paragraphe.

Écriture d'applications multimédias dans le langage Java. Pour autoriser l'écriture d'applications multimédias dans le langage Java, nous avons défini un modèle d'application multimédia. Ce modèle a pour objectif de permettre la caractérisation des contraintes temporelles des applications multimédias à flux continus. Nous avons ainsi introduit la notion d'application décomposée en un ensemble de traitements périodiques, chaque traitement étant décomposé en une ou plusieurs unités d'exécution (correspondant à l'exécution d'une partie d'un traitement). Partant de ce modèle, nous avons ensuite défini une API Java.

Ordonnancement multi-critère. Le travail de l'ordonnanceur [31] est réalisé via l'exécution d'une règle de placement exécuté par chaque processeur qui permet de choisir la tâche à exécuter parmi l'ensemble des tâches Java prêtes à s'exécuter. Cette règle est actuellement en cours de définition et doit permettre d'atteindre plusieurs objectifs :

- prise en compte dynamique d'une nouvelle application temps-réel,
- introduction de parallélisme de manière transparente pour l'utilisateur au niveau de l'exécution en utilisant l'ensemble des processeurs,

- Eeécution *au meilleur effort* des applications multimédias ;
- prise en compte de l'hétérogénéité des processeurs au niveau temporel, en réalisant le placement en fonction du temps d'exécution des traitements qui varient d'un processeur à l'autre,
- prise en compte de l'hétérogénéité des processeurs au niveau énergétique, en réalisant le placement en fonction de la consommation énergétique des traitements afin de minimiser la consommation énergétique globale du multiprocesseur.

Caractérisation (profiling) d'application multimédia. L'ordonnanceur se base sur les estimations temporelles et énergétiques propre à chaque application. Aussi, la définition d'un outil de caractérisation (profiling) permettant d'obtenir ces estimations est à l'étude. Le point de départ de notre approche consiste à se focaliser sur le profiling au niveau des bytecodes des traitements temps-réel d'une application Java en définissant une technique de profiling hors ligne d'une application. L'ajustement de manière dynamique, lors de l'exécution de l'application, de ces estimations est également à l'étude.

6.3.3 Ramasse-miettes adapté aux applications multimédia

Au niveau de l'environnement d'exécution Java, il est nécessaire de disposer d'un ramasse-miettes qui libère l'espace mémoire alloué aux objets Java devenus inutiles. Dans notre cas, ce ramasse-miettes a la particularité de ne pas devoir perturber l'exécution des applications multimédias. Contrairement aux approches existantes qui définissent une politique de ramasse-miettes pour un ensemble d'applications temps-réel, notre approche consiste à définir une politique de ramasse-miettes pour chaque application multimédia. Il est aussi possible de prendre en compte de manière fine tant le comportement temps-réel des traitements de l'application que les besoins mémoire spécifiques des traitements. Les politiques de ramasse-miettes que nous proposons s'appuient sur l'adaptation d'algorithmes de ramasse-miettes temps-réel existants dans la littérature.

6.4 Systèmes d'information spontanés (SIS)

Participants : Michel Banâtre, Paul Couderc, Jean-Marc Menaud, David Touzet, Arnaud Troël, Stéphane Tudoret, Frédéric Weis.

L'informatique mobile et la communication sans fil sont actuellement en plein essor. Bien que de nouvelles mises en œuvre tentent d'exploiter les possibilités offertes par ces nouvelles technologies, l'objectif final est simplement de rendre transparent la mobilité des entités au sein de l'architecture de communication. Notre approche est ici de montrer que dans un contexte particulier, celui du voisinage physique des entités communicantes, il est possible de tirer partie des facultés de mobilité et de la communication sans fil, pour créer de façon spontanée des systèmes d'information, que nous appelons SIS (Système d'Information Spontanés).

Durant cette année 2000, nous avons travaillé principalement sur deux axes : (i) la définition et l'évaluation d'un mécanisme de communication entre les nœuds d'un SIS, (ii) la découverte et les échanges d'informations au sein d'un SIS [13].

6.4.1 Communication entre les nœuds d'un SIS

Un SIS est un système instable : les nœuds entrent et sortent dynamiquement du système, en fonction de leur mobilité et de leur position relative. On ne peut donc a priori pas faire d'hypothèse sur le temps de communication entre les nœuds. De plus, les échanges au sein d'un SIS s'appuient sur une communication sans fil, soumise à de perpétuelles perturbations (grandes variations de performances en fonction de l'environnement, variation de la qualité du signal reçu, ...). Dans ce cadre, les déconnexions lors de transferts d'informations entre nœuds voisins doivent être considérées comme des événements normaux, et les paramètres caractérisant la mobilité des nœuds voisins ne doivent pas être masqués au système embarqué. Ceci nous amène à proposer des approches permettant d'estimer le temps de communication « restant » entre deux entités mobiles au sein d'un SIS. C'est ce que nous appelons un schéma prédictif de communication. Ce dernier s'appuie sur l'utilisation de paramètres gérés par le système de transmission sans fil, et qui caractérisent la mobilité des nœuds : distance, qualité et puissance du signal reçu, taux d'erreurs.

Le schéma prédictif permet donc de déterminer si une information qu'une application souhaite transmettre dans le cadre d'un SIS, peut être envoyée, compte tenu du temps de communication restant. Nous avons fait l'hypothèse qu'une application construite au sein d'un SIS, transmet ces informations sous la forme de blocs caractérisés pour chacun d'entre eux par une taille et une priorité. Ce choix nous a amenés à étudier différentes politiques d'ordonnancement visant à sélectionner le ou les blocs d'informations pouvant être transmis, en fonction du temps de communication fourni par le schéma prédictif d'une part, et les tailles et les priorités des blocs proposés par l'application d'autre part.

L'ensemble des schémas prédictifs étudiés sont actuellement évalués grâce à l'environnement de simulation NS (Network Simulator) développé par l'université de Berkeley. Cet environnement de simulation a été adapté au contexte de notre étude, afin de prendre en compte la mobilité des entités au sein d'un même espace géographique, et de pouvoir embarquer au sein de chaque entité simulée l'architecture de communication, propre aux systèmes d'informations spontanés.

6.4.2 Découverte et échanges d'informations au sein d'un SIS

Un SIS est constitué d'un ensemble d'entités dont le nombre varie perpétuellement. Chacune de ces entités est porteuse d'un ensemble d'informations, qu'elle est susceptible de transmettre dans le cadre du SIS auquel elle participe. La difficulté de la mise en œuvre de tels mécanismes au sein des SIS vient de leur nature totalement distribuée. En effet, chaque entité est autonome. Il est donc impossible de faire jouer à l'une d'entre elles le rôle d'un annuaire centralisé (à la façon par exemple dont le lookup service centralise les services au sein du système JINI).

Il est nécessaire de réaliser un système de découverte des informations entièrement réparti, où chacune des entités est chargée de présenter elle-même les informations qu'elle détient aux entités du SIS qui en font la demande. Des premiers éléments de solution à ce problème ont été proposés.

Il s'agit d'abord de réaliser une indexation des informations stockées au niveau de chaque entité du SIS. Nous proposons ainsi d'adjoindre à chaque objet une enveloppe dans laquelle sont stockées toutes les informations utiles le concernant (en particulier une liste de mots clefs décrivant le contenu de l'objet). Cette séparation entre l'objet et sa représentation a pour but de faciliter ensuite la recherche d'informations sur les entités distantes au sein d'un SIS. À partir d'une définition, les informations sont regroupées par domaine, en utilisant une technique dérivée du *data mining*. Cette dernière permet, par l'association des mots clefs apparaissant fréquemment ensemble, de distinguer les principaux centres d'intérêts d'un utilisateur. Nous avons également défini des profils utilisateurs, c'est-à-dire les sujets sur lesquels ce dernier souhaite récupérer de l'information au sein d'un SIS. Le protocole de découverte d'information que nous sommes en train d'étudier permet donc à une entité de découvrir, parmi l'ensemble des entités disponibles sur les entités distantes du SIS, toutes celles qui correspondent à son profil.

6.5 Informatique nomade et réseaux sans fil

Participants : Françoise André, Maria-Teresa Segarra.

Les propriétés de l'informatique mobile, où sont impliqués des ordinateurs portables utilisant des réseaux sans fil, diffèrent de celles définies pour des contextes plus traditionnels, comme celui des réseaux locaux filaires par exemple. Ces différences ont des répercussions sur les applications, notamment parce que celles-ci doivent être mises en œuvre avec des ressources limitées (faibles débits réseau par exemple). De plus, les propriétés du nomadisme informatique combinées à celles des réseaux sans fil rendent envisageable la mise au point de nouveaux modes de fonctionnement pour les applications, comme par exemple la possibilité (parfois la nécessité) de travailler en mode déconnecté.

Pour un utilisateur d'applications, le nomadisme présente de nombreux avantages. Pour les concepteurs d'applications, ou encore pour les concepteurs de systèmes d'exploitation répartis, le nomadisme introduit une complexité supplémentaire. Les applications existantes, non pensées en vue d'être utilisées dans un contexte nomade, risquent fort de ne pas fonctionner. Les problèmes qu'elles peuvent rencontrer sont par exemple dus aux mécanismes internes qu'elles utilisent, et qui peuvent être incompatibles avec la mobilité. Les techniques de verrouillage traditionnelles aux bases de données et utilisées pour protéger l'accès concurrent aux données partagées sont un exemple d'incompatibilité. Si l'application ayant posé des verrous se déconnecte, la base peut rester partiellement inaccessible, et cela sans limite dans le temps. A l'identique, un système d'exploitation réparti traditionnel peut ne pas être capable d'offrir un support d'exécution adéquat à des applications mobiles, si le nomadisme n'a pas fait partie de sa conception dès l'origine.

L'activité Molène au sein du projet Solidor a pour but de fournir un ensemble de services système génériques permettant à des concepteurs de rendre des applications ou des systèmes d'exploitation répartis compatibles avec les propriétés de l'informatique nomade. Molène est une couche logicielle fondée sur le modèle objet qui se place entre un système d'exploitation réparti traditionnel et une application elle aussi traditionnelle (c'est-à-dire initialement non prévues pour fonctionner dans un contexte nomade). Molène permet au système d'exploitation réparti et à l'application d'ignorer certains problèmes liés à la mobilité, et leur donne l'illusion que chacun fonctionne dans un cadre traditionnel. La modélisation objet utilisée dans Molène permet de spécialiser aisément un comportement particulier (gestion de la cohérence de données en présence de déconnexions par exemple), et d'introduire des stratégies d'adaptation dynamiques afin d'obtenir un comportement mieux adapté aux conditions d'exécution courantes.

L'année 2000 a vu la mise au point d'un prototype de Molène et des expériences d'utilisation de celui-ci ont été conduites dans des domaines d'application divers. Ces expériences nous ont permis de valider l'intérêt des mécanismes proposés et notamment de l'adaptation dynamique à l'environnement [3]. Ce travail se poursuit en étudiant notamment la coordination lors de l'adaptation simultanée de plusieurs composants. Plusieurs perspectives sont ouvertes en particulier pour régler les problèmes d'adaptation dans le cadre de l'utilisation conjointe de divers types de réseaux mobiles.

Cette année a également vu la fin de l'action Charcot, qui nous a permis d'étudier l'intérêt des mécanismes d'adaptation dynamique sur les systèmes transactionnels. Cette action avait pour objectif de permettre à un médecin de saisir la prescription d'un patient au chevet de celui-ci grâce à un ordinateur portable relié au système d'information hospitalier par un réseau local sans fil. Ce type de réseau offre de meilleures performances qu'un réseau de type GSM : il y a peu de déconnexions et elles sont de courte durée, la qualité de communication maximale est suffisante pour ce genre d'applications mais elle varie fortement au cours des déplacements de l'utilisateur. Afin de s'adapter à cette variation Charcot exécute les transactions sur le serveur lorsque la qualité de communication est bonne et il les exécute sur le poste mobile lorsque la qualité est mauvaise. Charcot dispose de mécanismes pour passer d'un mode d'exécution à l'autre ; il dispose également d'un service de préchargement de données qui détermine par anticipation les données qu'il faut ajouter ou supprimer sur le poste mobile. Il est associé à un service de réplication qui permet de copier des données du serveur sur le poste mobile et d'assurer leur mise à jour dans les deux sens (serveur vers client et client vers serveur).

6.6 Systèmes adaptatifs

Participants : Françoise André, Michel Banâtre, Paul Couderc, Frédéric Le Mouël.

Le développement de l'informatique nomade conjugué à la croissance exponentielle de l'Internet met en évidence un crucial besoin d'adaptation dans le développement de systèmes distribués. Ainsi, en fonction de ses déplacements, l'environnement informatique d'un utilisateur mobile est susceptible de varier considérablement en terme de latence, bande passante, terminal ou services disponibles par exemple. Notre objectif dans ce cadre consiste à fournir un environnement de conception de systèmes et d'applications permettant de répondre aux besoins d'adaptation mis en évidence dans les environnements extrêmement variables. Ainsi, l'information doit pouvoir se décliner en de multiples représentations afin de fournir à un utilisateur mobile la représentation la plus appropriée à son contexte informatique. Nous définissons le contexte d'une application comme étant une liste de couples (attribut, valeur) permettant de caractériser un environnement à un instant donné.

Infrastructure générique pour une distribution adaptative et dynamique en environnement mobile. Une facette de ces travaux repose sur la distribution des applications en fonction d'un environnement d'exécution mobile et hautement variable. En effet, les environnements mobiles présentent des caractéristiques différentes des environnements classiques : (i) les terminaux mobiles offrent peu de ressources et celles-ci sont limitées (batterie), (ii) les réseaux sans fil offrent, en général, une bande passante plus faible que celle des réseaux filaires et celle-ci est sujette à d'importantes variations et de fréquentes déconnexions (interférences, changement de cellule, perte de couverture), (iii) le contexte d'exécution change suivant les déplacement du mobile avec l'apparition et la disparition de stations, ressources, périphériques, etc.

La distribution des applications "mobiles" permet de pallier la pauvreté des ressources du terminal mobile par une utilisation au mieux des ressources de l'environnement. Selon la disponibilité des ressources, la distribution améliore les performances ou en évite la dégradation. Pour s'adapter aux fluctuations de l'environnement mobile, la distribution doit prendre en compte les différents critères du cadre mobile : batterie, bande passante, latence, probabilité de déconnexion, probabilité de reconnexion, pour assurer le dynamisme aussi bien au niveau du placement que des politiques de distribution elles-mêmes.

Ces différentes fonctionnalités sont mises en œuvre dans le système AeDEn (*Adaptive Distribution Environment*) au sein de *services*. Trois services principaux assurent les fonctionnalités : (i) le service de Détection et Notification détecte et notifie les services concernés des différents changements des ressources matérielles et logicielles de l'environnement, (ii) le service de Gestion de l'Environnement maintient l'état courant des différentes ressources présentes dans l'environnement et (iii) le service de Distribution décide, selon les besoins de l'application et selon les ressources de l'environnement, quels composants de l'application doivent être distribués et où ils doivent l'être.

Il existe de nombreuses *politiques* correspondant à des choix de mise en œuvre pour ces différents services. Les choix de politiques sont très importants puisqu'ils induisent un coût plus ou moins important sur l'utilisation des ressources et que l'on dispose de ressources limitées et variables. Pour pouvoir adapter dynamiquement les politiques de distribution à l'environnement, AeDEn propose des *politiques dynamiques*. Une politique dynamique peut changer d'implantation en cours d'exécution lorsque des changement surviennent dans l'environnement. Différentes politiques classiques sont implantées et mises à disposition des applications. Les applications peuvent également implanter leur propre politique et l'inclure dynamiquement. Ces travaux sont présentés dans [29, 11, 28, 30].

L'adaptation face à la vitesse d'évolution des composants logiciels. Le déploiement logiciel, c'est-à-dire les techniques mises en œuvre pour amener le logiciel depuis le site où il est développé vers les sites où il sera utilisé, est aujourd'hui profondément modifié par l'Internet. D'une part l'Internet facilite la réutilisation de code existant en rendant accessible aux développeurs de vastes bases de composants, ce qui tend à accroître, pour un composant donné, sa dépendance par rapport à des composants externes dont l'évolution n'est pas contrôlée. D'autre part l'Internet accroît considérablement la fréquence des mise-à-jour visibles pour les sites utilisateurs, ce qui rend difficile l'administration des configurations logicielles sur ces sites du fait de dépendances complexes existant entre les composants formant ces configurations. En effet, la configuration d'un système logiciel, autrefois principalement déterminée par le développeur lorsque l'ensemble du système était distribué de façon monolithique (sous forme d'une archive par exemple), devient déléguée aux administrateurs qui ont à décider des composants à installer et à mettre à jour, ces derniers étant disponibles et mis à jour indépendamment les uns des autres sur le réseau.

De notre point de vue, un mécanisme automatique et transparent est nécessaire pour le déploiement incrémental. En effet, l'augmentation de la fréquence des mises-à-jour et du nombre de dépendances entre les composants pose deux problèmes majeurs : d'une part la détermination d'une nouvelle configuration valide à chaque mise-à-jour est complexe pour l'administrateur, et d'autre part, une fois la nouvelle configuration déterminée, son activation nécessite l'arrêt du système utilisant la configuration précédente. Pour résoudre ce problème, nous proposons une solution exploitant le langage Java : notre système permet le déploiement incrémental de composants Java en respectant les dépendances de versions. Une nouvelle version d'un composant peut être intégrée dans une configuration active, grâce à un mécanisme de remplacement dynamique, qui fonctionne y compris dans le cas où des instances de l'ancienne version sont en cours d'utilisation. Les contraintes imposées au développeur d'une classe pour exploiter le mécanisme sont limitées à la spécifications des dépendances ainsi que la fourniture d'un transformateur permettant l'instanciation d'un composant à partir d'une instance de la version précédente. Le principal bénéfice du système est de rendre complètement transparent le déploiement et la maintenance des logiciels exploitant ce mécanisme.

Tous ces travaux sont détaillés dans [1].

6.7 Caches coopératifs pour systèmes d'information sur Internet

Participants : Michel Banâtre, Valérie Issarny, Jean-Marc Menaud.

Les systèmes d'information sont notamment caractérisés par : *(i)* des volumes importants d'informations qui transitent vers les utilisateurs, *(ii)* le nombre important de ces utilisateurs, *(iii)* la nature multimédias des informations et *(iv)* les contraintes de qualités de service à respecter (temps de réponse, disponibilité, etc.).

Nous avons principalement travaillé sur l'amélioration du temps de réponse dans les systèmes d'information grande échelle. Nous montrons, après étude, que cet objectif peut être atteint en utilisant une méthode de distribution et réplication dynamique d'informations (appelée *mise en cache*) au niveau du réseau informatique. Un cache réseau permet de sauvegarder dans un espace local un ensemble d'informations accédées par ses utilisateurs proches. En interceptant les requêtes des utilisateurs et en fournissant l'information recherchée, si cette dernière est présente dans l'espace local, le cache réseau permet de diminuer le temps de réponse aux informations, de réduire l'utilisation du réseau, et de soulager la charge du service en-ligne. Malheureusement, les caches réseaux ne possèdent qu'une faible efficacité.

Pour pallier ce problème, nos propositions consistent à *augmenter virtuellement la taille*, à *affiner la politique de remplacement d'objets* et à *augmenter virtuellement le nombre de clients* d'un cache réseau [2, 8]. La première solution (LRU-QoS) réside en la dégradation des informations, sauvegardées localement, les moins accédées. La seconde proposition (LRU-HOT) consiste à affiner la politique de remplacement d'objets d'un cache en déterminant les informations possédant un fort potentiel de ré-accès, par intégration des fréquences d'accès de ces dernières, délivrées par le service en-ligne les maintenant. Finalement, l'accroissement du nombre de clients est géré en réalisant une coopération entre différents caches réseaux du système d'informations. La principale difficulté a été la définition d'un protocole de coopération extensible, nommé Scoops, car le bénéfice de l'approche ne se fait sentir que pour un très grand nombre de caches. L'ensemble de ces travaux est décrit dans les documents [2, 8, 23].

7 Contrats industriels (nationaux, européens et internationaux)

7.1 Contrats nationaux

7.1.1 Hades (partenariat DGA/Dassault-Aviation)

Participants : Michel Banâtre, Pascal Chevochot, Isabelle Puaut, Stéphane Tudoret.

- Numéro de la convention : 3198C4980031303011.
- Intitulé : Conception et réalisation d'un support d'exécution réparti hautement disponible pour applications temps-réel.
- Activité de recherche concernée : § 6.2.
- Partenaires : DGA, Dassault-Aviation.
- Période : février 1999 à août 2001.
- Financement : DGA.

L'évolution des systèmes d'avionique fait que, dans le contexte économique actuel, l'utilisation de composants matériels et logiciels *sur étagère* devient une nécessité, principalement pour des raisons de complexité grandissante (avionique modulaire embarquée). Cette action a pour objectif de concevoir et mettre en œuvre un support d'exécution distribué tolérant aux fautes pour les applications du domaine de l'avionique. Plus précisément, les objectifs visés sont: (i) fournir des mécanismes de tolérance aux fautes qui soient compatibles avec des échéances temps-réel strictes et qui soient transparentes pour le concepteur d'application ; (ii) fournir des mécanismes permettant de récupérer en-ligne les ressources inutilisées ; (iii) s'adapter à des demandes de qualité de service variées (échéances temporelles critiques, besoins en terme de tolérance aux fautes).

Le support d'exécution construit dans le cadre de le contrat avec la DGA sera utilisé à des fins d'évaluation par la société Dassault-Aviation. L'utilisation du support d'exécution par Dassault-Aviation, ainsi que celle par l'IRISA d'une application d'avionique, sont régies par un contrat de collaboration bilatéral avec Dassault-Aviation (convention numéro 098C3040031303012), de durée et de période identiques à celles du partenariat DGA.

7.1.2 Texas Instruments

Participants : Michel Banâtre, Gilbert Cabillic, Pascal Eloy, Teresa Higuera, Valérie Issarny, Jean-Philippe Lesot, Frédéric Parain, Jean-Paul Routeau, Pierre Tiako.

- Numéro de la convention : 198C2730031303202
- Intitulé : Real time Java Distributed Processing Environment
- Activité de recherche concernée : § 6.3.

- Partenaire : *Texas Instruments.*
- Financement : *Texas Instruments.*
- Date de début : 01/10/1998, date de fin : 30/09/2001.

L'objectif de ce partenariat est la conception et la mise en œuvre d'un environnement Java temps-réel au-dessus d'architectures multiprocesseurs hétérogènes embarquables (comme par exemple les ordinateurs de poche), supportant l'exécution concurrente d'applications multimédias.

7.1.3 Alcatel

Participants : Michel Banâtre, Frédéric Weis, Paul Couderc, Stéphane Tudoret, Arnaud Troël, David Touzet.

- Numéro de la convention : en cours de notification,
- Intitulé : logiciels et applications pour ordinateurs de poche sans fil utilisant des communications par voisinage physique.
- Activité de recherche concernée : § 6.4.
- Partenaire : *Alcatel.*
- Financement : *Alcatel.*
- Date de début : 01/10/2000, date de fin : 30/09/2003.

L'objectif de cette action est (i) de fournir une *architecture logicielle* pour les ordinateurs de poches sans fil permettant à ces derniers de coopérer dynamiquement et spontanément quand ils sont physiquement proches ; (ii) de fournir des *applications* qui exploitent cette architecture logicielle.

8 Actions régionales, nationales et internationales

8.1 Actions européennes

8.1.1 Projet LTR C3DS

Participants : Erwan Demairy, Valérie Issarny, Christos Kloukinas, Siegfried Rouvrais.

- Numéro de la convention : 197A93200000MC005.
- Intitulé : *Convention Esprit Ltr – C3DS*, Environnement pour le développement de services distribués complexes.
- Activité de recherche concernée : §6.1.
- Partenaires : Université de Newcastle (UK – coordinateur), Imperial College (UK), Inria-Rhône-Alpes, Bull SA (Grenoble).
- Période : 01/98–12/2000.
- Financement : CEE–ESPRIT.

L'action Esprit Ltr C3DS (*Control and Coordination of Complex Distributed Services*) porte sur la conception et la réalisation d'un environnement de développement de services distribués. Cet environnement s'appuie sur les notions suivantes : (*i*) la description des architectures logicielles des services, (*ii*) l'emploi d'agents, éventuellement mobiles, pour la réalisation des actions au sein des composants distribués constituant un service, et (*iii*) l'emploi de la technique de *workflow* pour coordonner les actions relatives à un même service.

8.1.2 Projet IST CALIM

Participants : Valérie Issarny, Apostolos Zarras.

- Numéro de convention : 100D0052,
- Intitulé : IST Calim – Corba Architecture for Legacy Integration and Migration
- Activité concernée : §6.1
- Période : [Janvier 2000 - Janvier 2002]
- Partenaires : EADS CCR (Suresne) – Coordinateur, Fiat-CRF (Italie), Informate (Belgique), Steria (Toulouse).

L'action CALIM porte sur la définition d'un processus de développement d'architectures de systèmes et des méthodes et outils associés pour faciliter l'évolution des systèmes d'information s'appuyant sur des systèmes logiciels complexes existants. L'approche retenue repose sur une intégration puis une évolution des systèmes logiciels existants, au sein du système d'information, en s'appuyant sur une plate-forme Corba.

8.1.3 Projet IST DSoS

Participants : Valérie Issarny, Christos Kloukinas, Viet Khoi Nguyen.

- Numéro de convention : 100D0150,
- Intitulé : IST DSoS – Dependable Systems of Systems
- Activité concernée : §6.1
- Période : [Avril 2000 - Mars 2003]
- Partenaires : Université de Newcastle (RU) – Coordinateur, Cnrs-Laas (Toulouse), Dera (RU), Lri (Orsay), Université d'Ulm (Allemagne), Université technique de Vienne (Autriche).

L'action DSoS vise à fournir des solutions facilitant la composition de systèmes de systèmes sûrs de fonctionnement, à partir de systèmes informatiques autonomes. Cette action se concentre notamment sur la conception, le placement et les propriétés des interfaces des systèmes à composer. Elle examine également la définition de méthodes et outils pour la construction effective des systèmes à partir de la spécification des interfaces des sous-systèmes à composer, ainsi que pour l'évaluation et la validation de la sûreté de fonctionnement des systèmes composés.

8.1.4 Projet ITEA Vivian

Participants : Malika Boulkenafed, Valérie Issarny, David Mentré.

- Intitulé : ITEA VIVIAN – Opening mobile platforms for the development of component-based applications
- Activité concernée : §6.1
- Numéro de convention : En cours de négotiation avec le ministère de l'industrie, labellisé ITEA en Janvier 2000.
- Période : [Septembre 2000 - Aout 2002]
- Partenaires : Nokia NRC (Finlande) – Coordinateur, CAS Software (Allemagne), CIME France, ERICSSON (Suède), HUT (Finlande), INT (Evry), ISOFT (Allemagne), MEMODATA (Caen), Paravent Oy (Finlande), Philips (Pays Bas), Unicom Consulting Ltd (Finlande).

L'action Vivian porte sur la définition d'une plate-forme pour ordinateurs de poche personnels, sans fil (*e.g.* Psion serie 5, *Communicator* de Nokia), afin de faciliter la construction d'applications logicielles distribuées pour ces systèmes. La solution retenue s'appuie sur une plate-forme *middleware* à base de composants, de type CORBA. Dans ce cadre, les problèmes abordés portent sur la définition et répartition des services composant l'architecture *middleware* sachant que les services devant s'exécuter sur les ordinateurs personnels doivent s'accommoder des ressources limitées disponibles.

8.1.5 Projet IST BRAIN

Participants : Michel Banâtre, Jean-Marc Menaud, Frédéric Weis.

- Numéro de convention : 100D0053
- Intitulé : on BRAIN (Broadband Radio Access for IP Based Networks)
- Période : [Janvier 2000-Mars 2001]
- Partenaires : Siemens (Allemagne) – Coordinateur, Btelecom (Angleterre), Dtelecom T-NOVA (Allemagne), Nokia (Finlande), Ericsson (Suède), NTT DoCoMo (Japon), Sony Europe (Allemagne), Agora (Espagne).

L'action BRAIN (Broadband Radio Access for IP Based Networks) porte sur la définition d'une architecture système pour terminaux mobiles, permettant de se connecter aussi bien à des réseaux respectant le standard HiperLan 2, qu'à des réseaux plus globaux comme l'UMTS.

8.2 Actions nationales

8.2.1 Projet RNTL Carlit

Participants : Nazim Boudeffa, Pascal Chevochot, Isabelle Puaut.

- Intitulé : RNTL Carlit – CARactérisation des Logiciels InTermédiaires

- Activité concernée : § 6.2.
- Numéro de convention : En cours de négotiation avec le ministère de l'industrie, labellisé RNTL en Juin 2000.
- Durée : 30 mois [Décembre 2000 - Juin 2003]
- Partenaires : Aérospatiale Matra Airbus, EDF, France Télécom R&D, IRISA, LAAS (coordinateur), LRI, Technicatome, Thomson-CSF.

L'utilisation de composants logiciels COTS (Commercial-Off-The Shelf) est désormais une dimension incontournable dans de nombreux secteurs d'activité. Elle se concrétise notamment par un impact très fort sur les logiciels intermédiaires ou "intergiciels" s'intercalant entre les logiciels exécutifs et les logiciels applicatifs. Cependant, cette évolution ne va pas sans poser des problèmes aux intégrateurs, tant du point de vue de la sûreté de fonctionnement que des comportements temporels, notamment en regard des exigences exprimées par les documents normatifs. L'objectif du projet RNTL Carlit est d'étendre significativement les possibilités de caractérisation des logiciels intermédiaires, actuellement disponibles, sur les plans de la sûreté de fonctionnement, du temps-réel (respect d'échéances temporelles) et de la performance.

8.2.2 Groupes de travail nationaux

Pascal Chevochot et Isabelle Puaut participent au groupe de travail GDR-ARP “Systèmes Temps-réel et Stochastiques” (STS).

8.3 Relations bilatérales internationales

P. Chevochot et I. Puaut participent à une coopération bilatérale Franco-Turque avec l'ICI (International Computing Institute) de l'EGE University (Bornova, Turquie), soutenue par le CNRS. Le thème de cette coopération est la conception et l'évaluation de performance d'un système de gestion de groupes extensible, temps-réel strict et tolérant aux fautes.

8.4 Réseaux et groupes de travail internationaux

8.4.1 Cabernet

- Numéro de la convention : en cours d'attribution
- Intitulé : Esprit : réseau d'excellence CABERNET (Network of Excellence in Distributed and Dependable Systems)
- Partenaires :
  - Université de Newcastle (coordinateur),
  - Alcatel Austria, Technische Universitaet Wien (Autriche),
  - Université Catholique de Louvain (Belgique),
  - INRIA, LAAS-CNRS, Sun Microsystems International, Université Joseph Fourier (France)
  - GMD, Universités d'Aachen, de Dortmund, d'Essen, de Hamburg, de Kaiserslautern, de Karlsruhe, de Stuttgart, Technische Universitaet Hamburg-Harburg, Friedrich Alexander University (Allemagne),

  - Foundation for research and technology d'Hellas (Grêce),
  - Trinity College Dublin (Irlande),
  - CNUCE-CNR, Instituto di Elaborazionre dell'informazione - NCR, Politecnico di Milano, Scuola superiore S. Anna, Tecnopolis CSATA Novus Ortus, TNO Bari, Universités de Bologne et Pise (Italie),
  - Universités de Twente et de Vrije (Hollande),
  - Critical software SA, Universités de Lisbonne et de Porto (Portugal),
  - Universités de Catalogne, de Madrid, de Valence (Espagne),
  - Universités de Chalmers, Maelardalen Hoegskola (Suède),
  - École polytechnique de Lausanne (Suisse),
  - Universités de Cambridge, Lancaster, British Telecom, City University, Hewlett-Packard, Imperial College of science technology and medecine, Microsoft research (Royaume-uni).
- Durée : 3 ans

Cabernet est un réseau d'excellence dans le domaine des systèmes distribués et des systèmes sûrs de fonctionnement. Sa mission est de coordonner la recherche européenne de haut niveau dans ces domaines, dans le but de la rendre visible aux gouvernements et aux acteurs industriels d'une part, et d'améliorer la qualité de l'enseignement d'autre part.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

9.1.1 Comités de programme

- F. André est membre du comité de programme de la conférence CARI'2000 - 5ième Colloque Africain sur la Recherche en Informatique, Antananarivo, Madagascar, octobre 2000.
- M. Banâtre est membre du comité de programme de la conférence ICDNS'2000 - International Conference on Dependable Systems and Networks, New-York, États-Unis, juin 2000.
- V. Issarny est membre du comité de programme de RENPAR'12 – Douzièmes Rencontres Francophones du Parallélisme. Juin 2000, Besancon, France
- V. Issarny est membre du comité de programme du Ninth ACM SIGOPS European Workshop – Beyond the PC : New Challenges for the Operating Systems, Danemark, septembre 2000.
- V. Issarny est membre du comité de programme de NOTERE 2000 – Colloque Francophone sur les NOuvelles TEchnologies de la REpartition. Novembre 2000, Paris, France.
- V. Issarny est membre du comité de programme de WPDRTS'2001 – 9th workshop on parallel and distributed real-time systems. Avril 2001, San Francisco, USA.
- V. Issarny est membre du comité de programme de CFSE'2 – seconde conférence francaise sur les systèmes d'exploitation. Avril 2001, Paris, France.

- V. Issarny est membre du comité de programme de RENPAR'13 – Treizièmes Rencontres Francophones du Parallélisme. Avril 2001, Paris, France.
- V. Issarny est membre du comité de programme de ISORC'2001 – The 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Mai 2001, Magdeburg, Allemagne.
- V. Issarny est membre du comité de programme de ICSE 2002 – the 24th International Conference on Software Engineering, May 2002, Buenos Aires, Argentine.
- V. Issarny est membre du comité de pilotage de WIESS, Workshop on industral experiences with system software, co-sponsorisé par USENIX, IEETCOS et ACM-SIGOPS.
- I. Puaut est membre du comité de programme de WPDRTS'2000 - Workshop on Parallel and Distributed Real-Time Systems, Cancun, Mexique, Mai 2000.
- I. Puaut est membre du comité de programme de ISORC'2K - Object-Real-Time Distributed Computing, Newport Beach, USA, mars 2000.

9.1.2 Autres responsabilités sur un plan international

- V. Issarny est vice-présidente de l'ACM SIGOPS.

9.1.3 Autres responsabilités sur un plan national

- F. André est membre élue du conseil de l'Ifsic.
- F. André est membre de la commission de spécialistes de la 27^e section de l'Université de Rennes 1.
- V. Issarny est vice-présidente du prix ASF de la recherche en système.
- I. Puaut est membre élue de la commission de spécialistes de la 27^e section de l'Insa.
- I. Puaut est membre du conseil de laboratoire et du comité des projets de l'IRISA.
- I. Puaut est membre du jury du titre d'ingénieur DPE (Diplômés Par l'État), spécialité informatique, de l'Insa de Rennes.

9.2 Enseignement universitaire

Les chercheurs et enseignants/chercheurs du projet Solidor coordonnent et participent à des enseignements de premier, second et troisième cycle à l'Ifsic, l'Insa de Rennes et l'IUT de St-Malo. Nous détaillons ci-dessous uniquement les interventions des membres du projet en *troisième cycle*.

- F. André assure une formation de système d'exploitation en DESS Compétences Complémentaires en Informatique (CCI) à l'Ifsic.
- M. Banâtre est responsable du cours de systèmes d'exploitation répartis ($5^{ème}$ année Insa de Rennes option informatique).
- M. Banâtre assure une formation de systèmes répartis à l'École des Mines de Nantes (filière FI4).
- M. Banâtre, I. Puaut et F. Weis interviennent dans le module SYCR (SYstèmes d'exploitation et de Calculs Répartis) du DEA d'informatique.

- M. Banâtre et F. Weis assurent une formation de systèmes répartis en Diic 3 ARC à l'Ifsic.
- F. Weis assure une formation sur les réseaux en DESS Domotique et Réseaux Intérieurs l'université de Rennes I (UFR Structure et Propriétés de la Matière).

9.3 Accueil de stagiaires

Pendant l'année 2000, les membres du projet ont accueilli et encadré des stagiaires venant de différents établissements :

- J. Allard, INSA de Rennes (stage de fin d'études)
- H. Bacha, Université de Versailles, Saint Quentin (stage de DEA)
- T. Colibert et J. Gloaguen, INSA de Rennes (stage de 4*ieme* année)
- F. Houdusse, T. Lautier, D. Touzet, IFSIC (stages de DEA)
- A. Troël (mémoire d'ingénieur de l'Institut d'Informatique d'Entreprise - IEE)

9.4 Participation à des colloques, séminaires, invitations

Des membres du projet ont participé à des conférences et « workshops » ; on se reportera à la bibliographie pour en avoir la liste. Certains ont également donné des séminaires :

- F. André. *MolèNE : un système générique pour la construction d'applications en environnements mobiles*. Exposé invité au 5ième CARI'2000, Antananarivo, Madagascar, octobre 2000.
- P. Chevochot. *Conception de systèmes distribués temps-réel strict tolérants aux fautes.* Réunion de travail du groupe GDR-ARP "Systèmes Temps-réel et Stochastiques" (STS), mars 2000.
- P. Chevochot, I. Puaut. *Utilisation d'éléments "sur étagères" (COTS) pour les systèmes distribués temps-réel strict critiques*, LAAS, Toulouse, février 2000, et Irccyn, Nantes, février 2000.
- V. Issarny. *Architectures middleware : De la conception à l'implémentation*, LIP6, Paris, France. Mai 2000.
- V. Issarny. *Software architecture for the design and construction of distributed systems*, Nokia NRC, Helsinki, Finlande. Juin 2000.
- V. Issarny. *Architectures logicielles de systèmes évolutifs*, Gemplus, Gemenos, France. Novembre 2000.
- V. Issarny. *Caching strategies for data-intensive Web sites*, University of Newcastle. Novembre 2000.
- V. Issarny. *Architectures de systèmes pour le Web*, EMN, Nantes, France. Décembre 2000.
- I. Puaut. *Design of a run-time support for dependable hard real-time applications: the Hades project*, International Computing Institute, Bornova, Turquie, mai 2000.
- I. Puaut. *Exécution d'applications temps-réel strict tolérantes aux fautes dans le système Hades*, réunion de bilan de l'ARC Tolère, Inria Rocquencourt, octobre 2000.

– F. Weis. *Spontaneous Information Systems.* 2nd Concertation Meeting of Mobile/Wireless/Satellite IST projects. Bruxelles, mai 2000.

Des membres du projet ont également eu l'occasion de présenter des posters dans le cadre de manifestations scientifiques :

– A. Colin, *Analyse statique des temps d'exécution au pire cas de systèmes temps-réel*, journées doctoriales DGA, octobre 2000.
– D. Decotigny, *Mécanismes système pour applications temps-réel strict soumises à des conditions opérationnelles évolutives*, journées doctoriales DGA, octobre 2000.
– D. Touzet, *Découverte et échange d'informations dans les Systèmes d'Informations Spontanés.* Journées doctorales « informatique et réseaux » JDIR'2000, novembre 2000.

9.5 Dépôts de brevets

– M. Banâtre, P. Couderc. « Télephonie mobile à périmètres de traitement sélectif ». Demande d'extension pour la protection à l'étranger du brevet Français PCT/FR00/01350, 18 mai 2000.
– M. Banâtre, F. Weis. « Poste mobile de traitement de données à module de communication locale ». Demande d'extension pour la protection à l'étranger du brevet Français PCT/FR99/02315, octobre 2000.
– M. Banâtre, P. Couderc. « Procédé et dispositif de téléphonie mobile permettant l'accès à un service contextuel exploitant la position et/ou l'identification de l'usager ». Demande de dépot de brevet, numéro 0012611, 3 octobre 2000.

10 Bibliographie

Thèses et habilitations à diriger des recherches

[1] P. Couderc, *Une approche pour le déploiement incrémental de composants logiciels sur Internet*, thèse de doctorat, Université de Rennes I, décembre 2000, À Paraître.

[2] J. M. Menaud, *Système de caches coopératifs pour les systèmes d'informations distribuées à grande échelle*, thèse de doctorat, Université de Rennes I, janvier 2000.

[3] M. Segarra, *Une plate-forme à composants adaptables pour la gestion des environnements sans fil*, thèse de doctorat, Université de Rennes I, novembre 2000.

[4] A. Zarras, *Configuration systématique de middleware*, thèse de doctorat, Université de Rennes I, mars 2000.

Articles et chapitres de livre

[5] F. André, M. Segarra, « MolèNE : un système générique pour la construction d'applications mobiles », *Calculateurs Parallèles. Numéro spécial «Evolution des plates-formes orientées objets répartis» 2*, 1/2000, 2000, p. 9–29.

[6] A. Colin, I. Puaut, « Worst Case Execution Time Analysis for a Processor with Branch Prediction », *The Journal of Real-Time Systems 18*, 2, mai 2000, p. 249–274.

[7] P. Couderc, A. Kermarrec, M. Banâtre, « Approches adaptatives en mobilité », *Techniques et Sciences informatiques (TSI) 19*, 4, 2000, p. 481–514.

[8] J.-M. Menaud, V. Issarny, M. Banâtre, « A Scalable and Efficient Cooperative System for Web Caches », *In IEEE Concurrency 8*, 3, July-September 2000, p. 56–62.

[9] C. Morin, A.-M. Kermarrec, M. Banâtre, A. Gefflaut, « An Efficient and Scalable Approach for Implementing Fault-Tolerant DSM Architectures », *IEEE Transactions on Computers 49*, 5, mai 2000, p. 414–430.

[10] F. Parain, G. Cabillic, M. Banâtre, V. Issarny, T. Higuera, J. Lesot, « Techniques de réduction de la consommation dans un système embarqué temps-réel », *Techniques et Sciences informatiques (TSI)*, 2000, à paraître. (également paru en rapport de recherche IRISA 1332 en mai 2000).

Communications à des congrès, colloques, etc.

[11] F. André, A. Kermarrec, F. L. Mouël, « Improvement of the QoS via an Adaptive and Dynamic Distribution of Applications in a Mobile Environment », *in : Proc. of the 19th International IEEE Symposium on Reliable Distributed Systems*, Nürnberg, Allemagne, octobre 2000.

[12] F. André, M. Segarra, « A Generic Approach to Satisfy Adaptability Needs in Mobile Environments », *in : Proc. of the 33rd Annual Hawaii International Conference on System Sciences*, Maui, Hawaii, Etats Unis, janvier 2000.

[13] M. Banâtre, P. Couderc, J. Menaud, F. Weis., « Proximate Interactions for Wireless Appliances », *in : Proceedings of the 9th ACM SIGOPS European Workshop - Beyond the PC: New Challenges for the Operating System*, Kolding, Denmark, septembre 2000.

[14] M. Banâtre, P. Couderc, F. Weis, « Spontaneous Communications », *in : Mobile Communications Summit*, Galway, Irlande, octobre 2000.

[15] G. Blair, L. Blair, V. Issarny, P. Tuma, A. Zarras, « Role of Software Architecture in Constraining Adaptation in Component-based Middleware Platforms », *in : Proceedings of Middleware 2000 – IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, *Lecture Notes in Computer Sciences*, *1795*, Springer-Verlag, Hudson River Valley (NY), USA, avril 2000.

[16] P. Chevochot, I. Puaut, « Holistic Schedulability Analysis of a Fault-Tolerant Real-Time Distributed Run-time Support », *in : Proc. of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA'00)*, Cheju Island, South Korea, décembre 2000.

[17] A. Colin, D. Decotigny, P. Chevochot, I. Puaut, « Are COTS suitable for building distributed fault-tolerant real-time systems ? », *in : 2000 Workshop on parallel and distributed real-time systems (WPDRTS)*, *Lecture Notes in Computer Sciences*, *1800*, Springer-Verlag, p. 699–705, Cancun, Mexico, mai 2000.

[18] D. Florescu, V. Issarny, P. Valduriez, K. Yagoub, « Caching Strategies for Data-Intensive Web Sites », *in : Proceedings of the International WWW Conference*, Amsterdam, Pays Bas, mai 2000. (Poster).

[19] D. Florescu, V. Issarny, P. Valduriez, K. Yagoub, « WEAVE: A Data-Intensive Web Site Management System », *in : Proceedings of the Conference on Extending Database Technology (EDBT)*, Konstanz, Allemagne, mars 2000. (Démonstration de logiciel).

[20] P. Fradet, V. Issarny, S. Rouvrais, « Analyzing Non-Functional Properties of Mobile Agents », *in : Proceedings of FASE 2000 – Foundational Aspects of Software Engineering*, *Lecture Notes in Computer Sciences*, *1783*, Springer-Verlag, p. 319–333, Berlin, Allemagne, mars 2000.

[21] T. Higuera, V. Issarny, F. Parain, G. Cabillic, M. Banâtre, J. Lesot, « Java Embedded Real-Time Systems: an Overview of Existing Solutions », *in : 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2k)*, p. 171–176, mars 2000.

[22] V. Issarny, J.-P. Banâtre, « Architecture-based Exception Handling », *in: Proceedings of the 34th Hawaii International Conference on Systems Sciences (HICSS)*, Maui (Hawaii), USA, janvier 2001. À paraître.

[23] V. Issarny, M. Banâtre, B. Charpiot, J.-M. Menaud, « Quality of Service and Electronic Newspaper: The Etel Solution », *in: Recent Advances in Distributed Systems*, S. Krakowiak, S. Shrivastava (éditeurs), *Lecture Notes in Computer Sciences*, *1752*, Springer-Verlag, 2000.

[24] V. Issarny, L. Bellissard, M. Riveill, A. Zarras, « Component-based Programming of Distributed Applications », *in: Recent Advances in Distributed Systems*, S. Krakowiak, S. Shrivastava (éditeurs), *Lecture Notes in Computer Sciences*, *1752*, Springer-Verlag, 2000.

[25] V. Issarny, G. Cabillic, M. Banâtre, F.Weis, P. Couderc, F. Parain, T. Higuera, « Providing an Embedded Software Environment for Wireless PDAs », *in: Proceedings of the 9th ACM SIGOPS European Workshop - Beyond the PC: New Challenges for the Operating System*, p. 49–54, Kolding, Denmark, septembre 2000.

[26] C. Kloukinas, V. Issarny, « Automating the Composition of Middleware Configurations », *in: Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'2000)*, Grenoble, France, septembre 2000.

[27] J.-M. Menaud, V. Issarny, M. Banâtre, « Improving Effectiveness of Web Caching », *in: Recent Advances in Distributed Systems*, S. Krakowiak, S. Shrivastava (éditeurs), *Lecture Notes in Computer Sciences*, *1752*, Springer-Verlag, 2000.

[28] F. L. Mouël, F. André, « AeDEn: un cadre général pour une distribution adaptative et dynamique des applications en environnements mobiles », *in: Actes du 3ème Colloque International sur les NOuvelles TEchnologies de la REpartition (NOTERE'2000)*, Paris, France, novembre 2000.

[29] F. L. Mouël, F. André, « Distribution over Mobile Environments », *in: Proc. of the 2000 ACM Symposium on Applied Computing*, p. 568–569, Como, Italy, mars 2000. Papier court.

[30] F. L. Mouël, M. Segarra, F. André, « Improving Mobile Computing Performance by Using an Adaptive Distribution Framework », *in: Proc. of the 7th International Conference on High Performance Computing*, Bangalore, Inde, décembre 2000.

[31] F. Parain, G. Cabillic, M. Banâtre, T. Higuera, V. Issarny, J. Lesot, « Increasing Appliance Autonomy using Energy-Aware Scheduling of Java Multimedia Applications », *in: Proceedings of the 9th ACM SIGOPS European Workshop - Beyond the PC: New Challenges for the Operating System*, p. 171–176, Kolding, Denmark, septembre 2000.

[32] M. Segarra, F. André, « A Framework for Dynamic Adaptation in Wireless Environments », *in: Proc. of TOOLS Europe 2000*, p. 336–347, Mont St. Michel, St. Malo, France, juin 2000.

[33] K. Yagoub, D. Florescu, C. C. Andrei, V. Issarny, « Building and Customizing Data-intensive Web Site using Weave », *in: Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, Le Caire, Egypte, septembre 2000. (Démonstration de logiciel).

[34] K. Yagoub, D. Florescu, V. Issarny, P. Valduriez, « Caching Strategies for Data-Intensive Web Sites », *in: In Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, Le Caire, Egypte, septembre 2000.

[35] A. Zarras, V. Issarny, « Assessing Software Reliability at the Architectural Level », *in: Proceedings of the 4th International Software Architecture Workshop*, Limerick, Ireland, juin 2000.

Rapports de recherche et publications internes

[36] P. Chevochot, I. Puaut, G. Cabillic, A. Colin, D. Decotigny, M. Banâtre, « Hades: a distributed system for dependable hard real-time applications built from COTS components », *rapport de recherche n° 1257*, IRISA, octobre 2000.

[37] P. Chevochot, I. Puaut, « Experimental evaluation of the fail-silent behavior of a distributed real-time run-time support built from COTS components », *rapport de recherche n° 1370*, IRISA, décembre 2000.

Divers

[38] « État d'avancement à 18 mois, collaboration INRIA-Texas Instruments », mars 2000.

[39] « État d'avancement à 24 mois, collaboration INRIA-Texas Instruments », octobre 2000.

[40] « Fourniture 1.4 du marché 98.34.375.00.470.75.65 INRIA/DGA intitulé "Conception et réalisation d'un support d'exécution réparti hautement disponible pour applications temps-réel" (rapport final de la tranche 1) », février 2000.

[41] « Fourniture 2.1 du marché 98.34.375.00.470.75.65 INRIA/DGA intitulé "Conception et réalisation d'un support d'exécution réparti hautement disponible pour applications temps-réel" (état d'avancement des travaux) », mai 2000.

[42] « Fourniture 2.2 du marché 98.34.375.00.470.75.65 INRIA/DGA intitulé "Conception et réalisation d'un support d'exécution réparti hautement disponible pour applications temps-réel" (rapport intermédiaire) », août 2000.

[43] « Fourniture 2.3 du marché 98.34.375.00.470.75.65 INRIA/DGA intitulé "Conception et réalisation d'un support d'exécution réparti hautement disponible pour applications temps-réel" (état d'avancement des travaux) », novembre 2000.

[44] « Charcot. Conception et réalisation d'un environnement pour l'exécution d'applications hospitalières mobiles », Rapport final du contrat ITR Convention INRIA-Région Bretagne, numéro 198C5630031303061., juillet 2000.