



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet aces

Informatique diffuse et systèmes embarqués

Rennes

THÈME 1B

*R*apport
d'Activité

2002

Table des matières

1. Composition de l'équipe	1
2. Présentation et objectifs généraux	1
3. Fondements scientifiques	3
3.1. Introduction	3
3.2. Systèmes d'exploitation embarqués	3
3.2.1. Contraintes de temps	3
3.2.1.1. Temps-réel strict.	3
3.2.1.2. Systèmes temps-réel souple.	4
3.2.1.3. Performance temporelle.	4
3.2.2. Consommation énergétique	4
3.2.3. Mémoire	5
3.2.4. Tolérance aux fautes	5
3.3. Informatique diffuse	6
3.3.1. Notion de contexte	6
3.3.1.1. Introduction	6
3.3.1.2. Système d'information spatial	7
3.3.2. Accès à l'information en environnement mobile	7
3.3.2.1. Adaptation à la mobilité	8
3.3.2.2. Masquage de la mobilité	8
3.3.3. Caractéristiques d'un Réseau ad hoc	8
3.3.3.1. Généralités	8
3.3.3.2. Bluetooth et le routage ad hoc	9
4. Domaines d'application	10
4.1. Introduction	10
4.2. Ordinateurs de poche pour applications multimédias embarquées	10
4.3. Systèmes d'information pour ordinateurs de poche en environnement mobile	10
4.4. Navigation Spatiale et navigation Web	12
5. Logiciels	12
5.1. Introduction	12
5.2. Environnement d'exécution Java pour architecture embarquée	12
5.3. Environnement pour la mise en œuvre des systèmes d'information spontanés	13
5.4. Heptane	13
5.5. Artisst	13
5.6. Spread	14
6. Résultats nouveaux	14
6.1. Introduction	14
6.2. Caractérisation de la consommation en ressources des logiciels embarqués	15
6.3. Environnement Java embarqué pour ordinateurs de poche	16
6.3.1. Machine d'exécution Java modulaire (Scratchy)	17
6.3.2. Ordonnancement multicritère d'applications multimédias	18
6.3.3. Disponibilité pour environnement d'exécution sans fil	19
6.4. Noyaux de systèmes d'exploitation embarqués	19
6.4.1. Mécanismes de base pour les systèmes d'exploitation embarqués	19
6.4.2. Système d'exploitation embarqué construit en Java	20
6.5. Evaluation de performances des systèmes embarqués	20
6.6. Systèmes d'information spontanés (SIS)	21
6.6.1. Gestion des interactions de proximité au sein d'un SIS	21

6.6.2.	Représentation des informations au sein d'un SIS	22
6.6.2.1.	Techniques d'accès à l'information au sein d'un SIS	23
6.7.	Systèmes d'informations spatiaux	24
6.7.1.	Structure d'un système d'information spatial	25
6.7.1.1.	Mécanismes d'adressage à base de tuple	25
6.7.1.2.	Accès à l'information spatiale	25
6.7.2.	Architecture	25
6.7.2.1.	Gestion de l'espace des tuples.	26
6.7.2.2.	Adressage spatial.	26
6.7.3.	Applications	26
6.7.4.	Problèmes ouverts	26
7.	Contrats industriels	26
7.1.	Contrats nationaux	26
7.1.1.	Texas Instruments	26
7.1.2.	Alcatel	27
8.	Actions régionales, nationales et internationales	27
8.1.	Actions européennes	27
8.1.1.	Programme d'action intégré Picasso	27
8.2.	Réseaux et groupes de travail internationaux	27
8.2.1.	CaberNet	27
9.	Diffusion des résultats	28
9.1.	Animation de la communauté scientifique	28
9.1.1.	Comités de programme	28
9.1.2.	Autres responsabilités sur un plan national	28
9.2.	Enseignement universitaire	28
9.3.	Accueil de stagiaires	29
9.4.	Participation à des colloques, séminaires, invitations	29
9.5.	Dépôts de brevets	30
10.	Bibliographie	30

1. Composition de l'équipe

Responsable scientifique

Michel Banâtre [DR Inria]

Assistantes de projet

Fabienne Cuyollaa [Technicien de Recherche, jusqu'au 31/10/2002]

Evelyne Livache [Technicien de la Recherche, depuis 1/11/2002]

Personnel Inria

Gilbert Cabillic [CR]

Personnel CNRS

Jean-Paul Routeau [IR]

Personnel Université

Frédéric Weis [MC, IUT de Saint-Malo]

Personnel INSA de Rennes

Isabelle Puaud [MC, HdR, en délégation CNRS, jusqu'au 30/09/2002]

David Decotigny [ATER, depuis 1/09/2002]

Ingénieurs experts Inria

Mathieu Becus [depuis 15/9/2002]

Paul Couderc

Jean-Philippe Lesot

Salam Majoul

Frédéric Parain

Stéphane Tudoret [jusqu'au 30/09/2002]

Grégory Watts

Chercheurs doctorants

Alexis Arnaud [bourse Inria, depuis 1/10/2002]

Carole Bonan [bourse Inria, depuis 1/10/2002]

David Decotigny [bourse DGA, jusqu'au 31/08/2002]

Julien Pauty [bourse MESR, depuis 1/10/2002]

Pushpendra Singh [bourse Inria]

David Touzet [bourse Inria]

Arnaud Troël [bourse Inria]

Autre personnel

Mathieu Avila [Ingénieur associé, depuis 1/10/2002]

2. Présentation et objectifs généraux

La principale raison d'être d'un système d'exploitation est de *gérer les ressources* de l'architecture matérielle sous-jacente en vue d'exécuter des applications, en offrant une *interface d'accès abstraite* aux ressources ("virtualisation" des accès aux ressources) afin de masquer l'accès au matériel.

Si cette problématique reste au cœur de la recherche en système d'exploitation, il n'en demeure pas moins qu'elle est en renouvellement permanent, compte tenu des évolutions constantes des architectures matérielles et des applications, et aussi de *l'intégration de plus en plus poussée du traitement et de la communication*. Cette dernière exploite simultanément la communication "filaire" et la communication mobile à longue ou courte portée. L'autre aspect important de ces architectures c'est leur "miniaturisation", qui rend possible leur implantation dans les endroits les plus divers (autrefois la "salle machine", et aujourd'hui l'appareil photo numérique ou le téléphone portable en passant par les calculateurs embarqués dans les automobiles ou encore les cartes à puces).

Ces évolutions se traduisent par l'émergence de nouveaux domaines d'applications, qui cohabitent avec ceux plus classiques du parallélisme (calcul scientifique) ou de la coopération ("client-serveur"), ainsi qu'à la démocratisation d'autres domaines d'application. C'est le cas des applications *embarquées* qui, jusqu'à un passé récent, étaient confinées aux applications hautement critiques (nucléaire, avionique), et que l'on rencontre aujourd'hui dans des applications "grand public" (téléphonie mobile, ordinateurs de poche). Elles ont ainsi permis l'éclosion d'un nouveau thème d'application, qui de notre point de vue va se répandre dans tous les domaines de la vie courante. C'est celui de "l'informatique diffuse" ou (*ubiquité numérique*), que l'on peut définir intuitivement comme l'application d'un traitement donné en fonction des contextes (par exemple, les positions géographiques) respectifs, des entités impliquées dans ce traitement. La nature de ce traitement est variable, elle peut être relative au traitement de données multimédia ou une requête de base de donnée.

C'est dans ce contexte de l'évolution des architectures matérielles et des applications que se situent les objectifs scientifiques du projet ACES. De façon plus précise, les axes de recherche du projet ACES concernent :

- la conception des architectures et systèmes d'exploitation embarqués,
- les supports systèmes pour l'informatique diffuse.

Concernant le premier axe de recherches, la spécificité des systèmes d'exploitation embarqués réside dans l'impossibilité de s'abstraire des contraintes liées à l'environnement physique dans lequel ils s'exécutent, caractérisées le plus souvent par des ressources limitées, que ce soit au niveau de la mémoire, de l'énergie ou des capacités de traitement. De plus, compte tenu des fortes interactions des logiciels embarqués avec leur environnement, ces derniers sont confrontés à des contraintes de temps-réel ou de sûreté de fonctionnement plus ou moins fortes selon le type d'utilisation du système embarqué.

Plus précisément, les systèmes d'exploitation embarqués doivent :

- satisfaire des contraintes de temps-réel provenant des applications, qui peuvent être aussi bien des contraintes de temps-réel souples (par exemple dans le cadre d'applications multimédia) que des contraintes de temps-réel strict (par exemple dans le cadre de protocoles de communication).
- s'adapter à la nature périssable de certaines ressources (énergie), au fonctionnement aux limites du matériel (surchauffe), à des capacités de stockage limitées, et doivent intégrer l'instabilité des systèmes de communications utilisés (déconnexion, bande passante).
- offrir une puissance de traitement suffisante pour satisfaire les contraintes de temps d'exécution des applications.

Satisfaire l'ensemble de ces prérequis suppose des travaux de recherches pour d'une part caractériser les besoins en ressources (temps processeur, mémoire, énergie) des applications embarquées, et d'autre part concevoir des systèmes d'exploitation prenant en compte conjointement les contraintes provenant du matériel (ressources limitées) et des logiciels (temps-réel souple et/ou strict, sûreté de fonctionnement). Les travaux de recherche nécessaires relèvent des domaines de l'ordonnancement, de l'allocation de ressources, de la répartition de charge, ainsi que de l'organisation et l'accès aux données dans les nouvelles applications relevant de l'informatique diffuse. Pris de façon séparée, certains des problèmes ci-dessus ont déjà des solutions partielles. Le défi majeur est de les résoudre de façon globale, sachant que les mécanismes proposés par le système d'exploitation doivent être conçus en tenant compte à la fois de l'architecture matérielle et des applications envisagées.

Concernant le second axe de recherches, nos travaux s'orientent dans deux directions principales :

- l'étude des systèmes d'informations spatiaux dans lesquels les objets présents dans un espace physique constituent un système d'information. Les mécanismes utilisés pour construire les espaces physiques identifiés reposent sur de la communication radio à courte portée.

- la construction de systèmes d'informations spontanés (et implicites) dont l'existence (ou la disparition) ne dépend que de la proximité physique (ou de l'éloignement) d'entités communicantes. La proximité ou l'éloignement sont caractérisés par le système de communication de radio courte portée sous-jacent.

Dans ce cadre, nous sommes conduits à considérer plusieurs types de problèmes. Sans être exhaustifs, ceux-ci sont relatifs à la découverte des entités communicantes, à la définition des protocoles de communications, à la représentation des contextes spatiaux, à l'accès aux informations, à la construction d'architectures *réelles*.

3. Fondements scientifiques

3.1. Introduction

Mots clés : *Systèmes d'exploitation embarqués, Temps-réel strict, temps-réel souple, Consommation énergétique, Outils de conception, Ubiquité, Informatique diffuse, Navigation spatiale.*

Nous présentons dans les paragraphes suivants un rapide panorama des travaux sur lesquels s'appuient les recherches menées dans le groupe, sans nous limiter aux travaux du projet ACES.

3.2. Systèmes d'exploitation embarqués

Le terme système embarqué dénote un système autonome, disposant de l'ensemble des éléments physiques nécessaires à son fonctionnement (processeur, mémoire, périphériques d'entrées/sorties) et en forte interaction avec l'environnement extérieur dans lequel il évolue. Les contraintes lors de la construction de tels systèmes sont relatives au temps, imposées par l'interaction du système avec son environnement (paragraphe 3.2.1), et aussi au caractère limité et/ou périssable des ressources dont dispose le système (énergie (paragraphe 3.2.2), mémoire (paragraphe 3.2.3), bande passante des communications vers l'extérieur). À ces contraintes s'ajoutent des contraintes de sûreté de fonctionnement (paragraphe 3.2.4) plus ou moins importantes selon le type d'utilisation du système embarqué.

3.2.1. Contraintes de temps

Beaucoup de systèmes embarqués interagissent directement avec leur environnement via des capteurs/actionneurs ou un réseau de communications sans fil. Ces interactions contraignent les temps de réponse du système embarqué de manière plus ou moins forte selon le domaine d'applications visé. On parle alors de *système temps-réel* [32], dans le sens où le temps de livraison des résultats d'un calcul fait partie intégrante de la spécification de ce dernier, au même titre que le résultat lui-même.

3.2.1.1. Temps-réel strict.

Dans les systèmes temps-réel *strict*, ou *dur*, le non respect des contraintes temporelles du système, le plus souvent exprimées sous la forme d'échéances de terminaison, constitue une défaillance de l'application. Dans le cadre d'applications critiques, telles que par exemple le contrôle de centrales nucléaires, une telle défaillance peut avoir des conséquences catastrophiques, telles que la mise en danger de vies humaines ou des pertes financières importantes.

Étant données les conséquences du non respect d'une échéance dans les systèmes temps-réel strict, il est nécessaire (ou tout du moins fortement recommandé) pour de tels systèmes de pouvoir vérifier avant leur exécution que toutes les échéances seront toujours respectées. Cette vérification est le rôle des méthodes d'analyse d'ordonnabilité, qui requièrent une connaissance du pire comportement temporel du système. De très nombreux travaux en analyse d'ordonnabilité (théorie de l'ordonnement) ont été menés depuis maintenant plus d'une trentaine d'années [31][18]. Ils diffèrent notamment de par le modèle de tâches considéré (loi d'arrivée des tâches, modes de communication entre tâches), le critère à minimiser (nombre d'échéances non respectées, retard cumulé), ou encore le type d'informations généré, utilisé en-ligne pour l'exécution des tâches (plans, priorités). Notons ici que la performance brute du système n'est pas primordiale, ce qui prime étant que les performances pire-cas du système permettent à ses échéances d'être respectées.

Alors que l'analyse d'ordonnabilité est un des champs de recherche traditionnel dans le domaine des systèmes temps-réel, l'obtention des temps d'exécution au pire cas (WCET) n'a attiré l'attention de la communauté de la recherche en temps-réel qu'il y a environ dix ans, et constitue à l'heure actuelle un thème de recherche très actif[29]. Les WCETs doivent être connus de manière *sûre* (dans le sens où ils doivent être supérieurs aux temps d'exécution effectifs) tout en étant le moins élevés possibles, pour ne pas entraîner de sur-dimensionnement des ressources nécessaires à l'exécution du système. Par exemple, dans l'aéronautique, il n'est pas rare de trouver des processeurs jamais utilisés à plus de 20% de leurs capacités maximales.

L'analyse statique de programmes permet d'obtenir des estimations sûres des WCET. Ces méthodes doivent à la fois caractériser les chemins d'exécution les plus longs dans les programmes, en opérant sur leur structure syntaxique[24][20] ou leur graphe de flot de contrôle[30] (analyse de *haut niveau*) et obtenir les temps d'exécution des séquences d'instruction machine en prenant en compte les caractéristiques du matériel, tels que la présence de mémoires caches[16], d'exécution pipelinée[36] ou de prédiction de branchement⁶ (analyse de *bas-niveau*). L'un des défis actuels est de prendre en compte dans les analyseurs des modèles temporels de processeurs qui soient les plus proches possibles de la réalité, et facilement extensibles pour intégrer leurs évolutions aussi rapidement que possible.

3.2.1.2. *Systèmes temps-réel souple.*

Dans les systèmes temps-réel souple, bien que l'instant de livraison d'un résultat soit important, la violation des contraintes temporelles du système est acceptable si elle reste rare. Cette tolérance est acceptée car les applications concernées ne relèvent pas du domaine des applications critiques. Les exemples typiques d'applications ayant des contraintes temps-réel souples sont les applications multimédias à flux continus. Le système vise au respect des contraintes temporelles dans la délivrance des flux de données afin de garantir la qualité des images et du son ; toutefois, les contraintes temporelles peuvent être adaptées puisque une dégradation de la qualité des données ne sera que faiblement perçue par l'utilisateur.

L'ordonnancement des calculs dans les systèmes temps-réel souple est important car il contribue au respect des contraintes temporelles des tâches. Toutefois, contrairement aux systèmes temps-réel strict, la violation d'une échéance a des contraintes moins sévères. On a alors recours à des méthodes d'ordonnancement qui diffèrent de plusieurs points de vue. D'une part, les données temporelles exploitées pour déterminer quel calcul effectuer ne sont plus nécessairement des données pire-cas ; on peut se contenter de connaître les caractéristiques temporelles du système de manière statistique. Par exemple, on utilisera les temps d'exécution moyens des calculs plutôt que des temps d'exécution au pire-cas. D'autre part, l'ordonnancement n'ayant pas à garantir le respect des échéances, une stratégie d'ordonnancement au meilleur effort permet d'exploiter au mieux les capacités de traitement des processeurs.

3.2.1.3. *Performance temporelle.*

À cause de la demande toujours croissante de puissance de traitement des applications, la performance globale temporelle du système embarqué est également un critère de conception important. Aussi, dans la conception d'un système embarqué la performance temporelle des mécanismes est une préoccupation constante. Au niveau logiciel ce travail porte à la fois sur les algorithmes afin de diminuer leur complexité, mais aussi sur l'optimisation des séquences fréquentes de code d'un mécanisme particulier. Parfois, l'utilisation de processeurs spécialisés peut également permettre d'atteindre la performance désirée. Une autre approche consiste à proposer un ordonnancement qui permette de réduire le temps de réponse des traitements temps-réel. Conjointement à ces travaux, le couplage fort des mécanismes de synchronisation à la stratégie d'ordonnancement (par exemple la gestion des sémaphores) contribue à diminuer ce temps de réponse.

3.2.2. *Consommation énergétique*

Une grande majorité des systèmes embarqués (téléphones cellulaires, ordinateurs de poche,...) sont confrontés au problème de l'autonomie. Aussi, afin d'étendre l'autonomie de fonctionnement d'un système deux approches complémentaires sont possibles : augmenter la capacité de stockage des batteries ou réaliser un système embarqué à faible consommation énergétique. Dans le cadre de cette dernière approche, plusieurs méthodes sont alors envisagées qui touchent à la fois le domaine de l'électronique et du logiciel : la conception

de composants électroniques consommant le minimum d'énergie, l'optimisation du logiciel afin de diminuer le coût énergétique de son exécution, et enfin la conception de stratégies logicielles exploitant les fonctionnalités du matériel. Nous nous intéressons plus particulièrement aux deux dernières méthodes (méthodes logicielles). *L'optimisation du logiciel* consiste à privilégier les instructions moins gourmandes en énergie afin de diminuer la consommation énergétique de l'exécution du programme[33]. Pour l'optimisation des applications de type protocoles réseau sans fil, elle se porte sur la réduction du volume des communications afin de diminuer la consommation d'énergie occasionnée par l'utilisation du réseau. De plus, dans un contexte d'ordinateur de poche sans fil, l'exécution distante de traitements permettra de diminuer la consommation locale de l'exécution du traitement pour peu que le mécanisme de réalisation de l'exécution distante consomme moins que le traitement proprement dit[21].

La conception d'une *stratégie logicielle exploitant des fonctionnalités du matériel* afin de diminuer la consommation énergétique de l'ordinateur touche principalement à l'ordonnancement. Tout d'abord, certains processeurs offrent différents modes d'exécution. Outre le mode d'exécution nominal, un mode veille permet à la fois d'endormir à faible coût le processeur et également de le réveiller de manière rapide. Le système d'exploitation aura donc dans ce cas la responsabilité de décider de passer le processeur en mode veille lors des périodes d'inactivité du système[25]. Une autre approche possible consiste à adapter dynamiquement la vitesse d'exécution du processeur en agissant par exemple sur sa tension d'alimentation[34]. En effet, le fait de réduire la tension du processeur réduit considérablement la consommation énergétique. L'ordonnancement doit donc, en fonction de la charge du processeur, décider de la tension la plus basse permettant d'exécuter les traitements. Considéré dans un contexte temps-réel, le problème est d'autant plus difficile que les contraintes temporelles des applications sont à prendre en compte[28].

3.2.3. Mémoire

La mémoire est une ressource limitée dans un grand nombre de systèmes embarqués (de quelques Kilo-octets dans une carte à puce à quelques Méga-octets dans un téléphone portable), et par conséquent une bonne utilisation de la ressource mémoire est cruciale pour ces systèmes. Les méthodes permettant d'utiliser des systèmes à faible capacité mémoire vont de l'utilisation de codes interprétés compacts, comme par exemple le bytecode Java, à l'utilisation d'algorithmes de compression[19], en passant par des algorithmes d'allocation dynamique de mémoire optimisés pour limiter le morcellement de la mémoire, ou fragmentation[23]. Une difficulté supplémentaire dans les systèmes embarqués est que la gestion de la mémoire soit compatible avec les contraintes temps-réel des applications, qu'elles soient souples ou strictes. Bien que de nombreux systèmes temps-réel se cantonnent à une gestion statique de la mémoire, quelques études récentes visent à utiliser des techniques de gestion de mémoire élaborées (allocation [9] et libération automatique de mémoire - garbage collection[26], espaces d'adressages séparés[17]) dans un contexte temps-réel embarqué.

3.2.4. Tolérance aux fautes

Certains systèmes embarqués doivent pouvoir remplir leurs fonctions malgré la présence de fautes, qu'elles soient d'origine physique ou humaine. Les moyens pour la sûreté de fonctionnement, et plus spécifiquement les méthodes de tolérance aux fautes, permettant au système de remplir ses fonctions en dépit des fautes pouvant affecter ses composants, ont fait l'objet de travaux fournis dans les trente dernières années pour des systèmes généralistes[15]. Par exemple, en ce qui concerne les fautes d'origine physique, il est nécessaire de détecter les erreurs, par l'utilisation de méthodes telles que les codes détecteur d'erreurs, les contrôles de vraisemblance ou encore le diagnostic en-ligne, puis d'effectuer un recouvrement d'erreur permettant au système de continuer à remplir ses fonctions malgré l'erreur, que ce soit par reprise de son exécution à partir d'un état sauvegardé au préalable (point de reprise) ou par compensation exploitant la redondance présente dans le système (par exemple la duplication active de tâches). Les difficultés issues du contexte embarqué sont relatives aux contraintes de temps des logiciels embarqués, ainsi qu'aux ressources limitées de l'architecture. Ceci contraint les types de méthodes de tolérance aux fautes utilisables dans un contexte embarqué temps-réel. En particulier, dans les systèmes temps-réel strict, il est nécessaire d'intégrer les mécanismes de tolérance aux fautes dans l'analyse d'ordonnabilité du système.

3.3. Informatique diffuse

Les bases de l'ubiquité numérique (ou informatique diffuse) ont été posées par Marc Weiser dans son article de 1993 intitulé "Some Computer Science Issues in Ubiquitous Computing"[35]. Il y définit l'ubiquité numérique comme une technologie invisible à des utilisateurs avec lesquels elle entretient des interactions permanentes. Dans cet article, Weiser dresse le constat suivant : exploiter les capacités d'un ordinateur nécessite aujourd'hui toute l'attention de son utilisateur. Les systèmes nomades actuels, conçus pour être utilisés par des utilisateurs mobiles, ne peuvent s'accommoder de ce type de contrainte.

Un nouveau mode d'utilisation des machines, plus intuitif et ne nécessitant pas d'interactions explicites, devient alors nécessaire afin de permettre aux utilisateurs de consacrer la totalité de leur attention aux tâches qui les occupent. Concevoir les environnements systèmes qui vont supporter de telles applications s'appuient sur des travaux dans les domaines complémentaires que sont la caractérisation des contextes dans lesquels évoluent les utilisateurs, la gestion des données accédées par ces derniers et les réseaux ad hoc.

3.3.1. Notion de contexte

3.3.1.1. Introduction

L'objectif de l'ubiquité numérique est de réaliser l'intégration transparente des environnements numériques au monde physique. Cette intégration des mondes physiques et numériques doit offrir aux utilisateurs des modes d'interaction plus naturels avec leur milieu. À terme, ces interactions devraient avoir lieu sans que l'utilisateur ait conscience d'utiliser les services de calculateurs avoisinants (on parle d'interactions invisibles).

Afin d'intégrer un calculateur à son environnement, il est nécessaire de le munir d'outils dédiés à la perception de ce dernier. Cette perception doit, de plus, être accompagnée d'une structuration appropriée des informations captées afin que celles-ci puissent être exploitées. La détection et l'analyse des situations sont deux tâches inconscientes que nous effectuons tout au long d'une journée. Par exemple, en rentrant dans une pièce, nous pouvons sans effort découvrir combien de personnes s'y trouvent, leur identité si elle nous est connue, ou encore la fonction de la pièce (bureau, salle de réunion ...). Cependant, lorsque ces tâches sont confiées à un ensemble de calculateurs, le traitement devient nettement plus délicat. En effet, la complexité du monde réel est telle qu'il est difficile pour un calculateur d'identifier les sources d'information pertinentes. Les applications d'ubiquité numérique délèguent donc les tâches d'acquisition de l'environnement d'exécution, également appelé **contexte**, à des systèmes spécialisés tels que le GPS (pour la localisation des utilisateurs). Les types de contexte gérés par de tels systèmes ne constituent qu'un sous-ensemble limité du contexte perçu par les humains.

Quatre classes de contexte sont généralement exploitées par les applications d'ubiquité numérique :

- le contexte *numérique*, qui regroupe les paramètres tels que la présence (ou l'absence) d'accès réseau, la bande passante disponible, les périphériques accessibles (imprimante, écran) ...
- le contexte *utilisateur*, qui traite des informations relatives à un utilisateur : son identité, ses préférences, sa localisation ...
- le contexte *physique*, relatif à l'environnement de l'utilisateur (conditions climatiques, niveau de bruit, luminosité ...), et le paramètre temps, qui fait référence à l'heure et la date. Ce dernier type de contexte, combiné aux types précédents, permet l'obtention d'historiques de contextes.

Pour illustrer cette notion de contexte, prenons l'exemple d'un musée équipé d'un système de guide virtuel où chaque visiteur équipé d'un calculateur portable de type PDA peut lire (sur son PDA) les informations relatives aux œuvres situées à sa proximité. Pour ce faire, le système doit être capable de détecter devant quelle(s) œuvre(s) se trouve le visiteur. L'ensemble des œuvres à proximité d'un utilisateur constitue, pour ce type de service, le contexte utile à l'application.

L'architecture générale d'un système pour l'ubiquité numérique repose sur trois éléments principaux :

- L'environnement réel, qui correspond aux entités du monde réel (objets, personnes, lieux...).
- L'environnement virtuel, constitué par des systèmes d'informations, comme le Web par exemple.
- Les interfaces qui permettent le couplage entre les deux environnements.

Le système d'information connaît l'état du monde réel grâce au "sondage" de l'environnement. Il peut s'agir de la position d'une personne (acquise grâce à un système de localisation, comme le GPS), des informations météorologiques en un lieu (acquises par des capteurs spécialisés), de l'état de fonctionnement d'un appareil etc. Le sondage de l'environnement recouvre donc toutes les techniques permettant de mettre à jour automatiquement les informations numériques concernant des événements ou des entités du monde réel.

À l'inverse, des interfaces permettent d'agir sur le monde réel à travers l'environnement numérique. Par exemple, les fenêtres d'un bâtiment ou d'une voiture peuvent être éventuellement automatiquement fermées s'il pleut.

Dans un environnement d'ubiquité informatique, l'utilisateur accède à l'espace d'information de façon implicite, par ses actions dans l'environnement réel. Cela suppose qu'il dispose en permanence d'un ordinateur (sur lui, ou à proximité), capable d'enrichir l'environnement d'informations utiles et de faciliter les manipulations du monde réel. Cet aspect recouvre d'une part les techniques d'interfaces utilisateurs nécessaires pour l'ubiquité numérique, comme la réalité enrichie, et d'autre part l'informatique mobile, afin d'assurer la proximité d'un ordinateur à l'utilisateur. L'ubiquité numérique nécessite aussi des systèmes de communications sans fil. En effet, l'environnement peut comporter de très nombreux objets étendus électroniquement et possédant une interface avec l'espace d'information ; utiliser des câbles pour connecter tous ces objets potentiellement mobiles n'est donc pas envisageable. Toujours dans un souci d'assistance implicite, l'accès aux informations doit exploiter au maximum la connaissance du contexte (acquise en particulier avec le sondage de l'environnement), afin d'éviter à l'utilisateur d'avoir à fournir explicitement des informations.

3.3.1.2. *Système d'information spatial*

Nous introduisons la notion de système d'information spatial (implicite) pour caractériser des informations obtenues automatiquement à partir d'interactions entre entités du monde réel. Le principe consiste à exploiter des propriétés existant dans le monde réel (comme la position ou la proximité physique) pour en déduire automatiquement un système d'information. La principale propriété spatiale utilisée dans les systèmes sensibles au contexte est la position relativement à l'espace physique. Celle-ci constitue un facteur important du contexte. En effet, l'espace physique est très organisé : un logement, une entreprise, un magasin, une bibliothèque, une ville constituent tous des structures spatiales. Par exemple, dans un magasin ou une bibliothèque, les éléments sont organisés dans l'espace par thème.

Ce type d'information peut être obtenu via un système de la localisation cartésienne, ou par de la localisation topologique. Dans le premier cas, on cherche à obtenir la position d'une entité relativement à un système de coordonnées. C'est ce que proposent les technologies comme la localisation satellite (GPS), ou la localisation par triangulation par rapport à des stations de base dans un réseau cellulaire. La localisation topologique consiste à localiser l'entité par son appartenance à un domaine défini logiquement, cette appartenance à un domaine étant définie par la proximité physique d'un marqueur (balise ou autre) construits notamment avec des balises infrarouges ou à radio-fréquence tel que la technologie RFID.

Un autre indicateur contextuel intéressant est l'identité des entités environnantes, qui peut être considérée comme dépendant de la position dans l'espace. En effet, cet indicateur est lié à notion de proximité dans l'espace physique, et donc à la position dans l'espace physique. On peut d'ailleurs remarquer que la localisation topologique exploite directement cette propriété.

Suivant l'environnement où l'on se trouve, d'autres informations spatiales sont parfois exploitées dans les systèmes sensibles au contexte : la vitesse, l'accélération et l'orientation peuvent ainsi affiner la description de la situation spatiale de l'entité. Elles sont soit acquises à partir de capteurs dédiés, soit dérivées à partir de l'évolution de la position.

3.3.2. *Accès à l'information en environnement mobile*

La mise en œuvre d'applications en environnement mobile repose en partie sur des techniques d'accès aux données localisées sur les différentes entités participants aux applications considérées. Celles-ci peuvent être regroupées en deux grandes catégories : d'un côté celles qui permettent aux applications de s'adapter à la mobilité, de l'autre celles qui cherchent à la masquer. Nous explicitons brièvement ces deux techniques.

3.3.2.1. Adaptation à la mobilité

Les communications sans fil sont exposées à d'importantes, et parfois brutales, variations de bande passante qui peuvent être à l'origine d'une perturbation, voire d'une interruption de service. Considérons l'exemple d'un utilisateur désirant visualiser une vidéo, stockée sur un serveur fixe, à partir d'un terminal mobile. Dès que la bande passante disponible n'est plus suffisante pour transmettre au terminal mobile les images suivantes, le service doit être interrompu.

Une des façons de surmonter ce type de difficulté est, lorsque c'est possible, d'adapter la qualité de l'information transférée à la bande passante disponible. La qualité d'une information peut être définie ici comme son *degré de fidélité* à une copie de référence (non dégradée). Cette technique ne peut cependant être appliquée qu'à des données structurées, c'est-à-dire de façon à ce qu'elles puissent être dégradées sans altération de sens. Les images et les vidéos peuvent être considérées comme telles.

Odyssey[27] est le principal système d'adaptation pour la mobilité. Bien que se voulant générique, il est principalement axé sur les variations de bande passante. Il se présente comme une surcouche du système d'exploitation à laquelle est confiée la gestion de l'adaptation pour les différentes applications d'un terminal mobile. Odyssey associe un composant, appelé surveillant, à chaque type de donnée dégradable. Les applications confient au surveillant concerné leur fenêtre de tolérance aux variations de la bande passante. Lorsqu'un surveillant constate que la valeur de cette dernière sort de l'une des fenêtres considérées, il avertit l'application concernée. La politique d'adaptation est à la charge de l'application. C'est elle qui décide, en fonction des retours fournis par les surveillants, de la fidélité des données à manipuler. Elle doit également adapter ses fenêtres de tolérance aux variations de la bande passante.

3.3.2.2. Masquage de la mobilité

Une connectivité intermittente peut isoler temporairement un utilisateur du reste du système. Dans une telle situation, le service proposé ne peut plus être assuré, dans la mesure où il est impossible d'accéder à de nouvelles informations distantes. Pour palier aux "trous" de connectivité, les systèmes nomades s'appuient sur des techniques de pré-chargement (hoarding), qui permettent d'anticiper le chargement sur un terminal mobile des documents qui seront ultérieurement nécessaires à son utilisateur. De cette façon, le système peut rester opérationnel en l'absence de connectivité (puisque les données sont stockées localement). Les déconnexions deviennent transparentes aux applications. La sélection des données à pré-charger s'effectue généralement à partir d'une combinaison d'informations explicitement fournies par l'utilisateur et d'informations acquises automatiquement (telles qu'un historique des documents accédés).

3.3.3. Caractéristiques d'un Réseau ad hoc

3.3.3.1. Généralités

Un réseau ad hoc (Mobile Ad Hoc NETWORKS -MANET-) est un réseau sans fil, exploitant les interfaces radio courte portée (Short Distance Wireless -SDW-), multi-saut. Il se distingue des réseaux câblés, des réseaux cellulaires et des réseaux locaux sans fil par les caractéristiques suivantes :

- Le réseau est mobile : les nœuds le composant sont généralement des appareils portatifs, fixés sur des objets se déplaçant dans l'espace (véhicules par exemple) ou encore sur des objets fixes, mais qui peuvent être déplacés sans qu'aucune intervention au niveau de la configuration du réseau ne soit nécessaire (imprimante, vidéoprojecteur par exemple).
- La topologie du réseau est dynamique : des nœuds peuvent entrer et sortir du réseau à tout moment. Le réseau doit être capable de gérer le routage des informations entre des nœuds dont l'existence peut être relativement éphémère.
- Le routage est multi-saut : cela signifie que des communications entre deux nœuds doivent pouvoir s'effectuer même si ceux-ci sont hors de portée de communication directe. La connaissance réciproque de leur existence et les échanges d'information doivent être possibles en traversant d'autres nœuds du réseau

- Chaque nœud du réseau est à la fois hôte et routeur : chaque nœud du réseau contribue au bon acheminement des informations dans le réseau. Pour ce faire chaque nœud essaie de posséder une connaissance partielle du réseau la plus étendue possible pour jouer le rôle de routeur pour lui-même et pour tout autre nœud qui le lui demandera.
- Les nœuds ont des contraintes énergétiques : les nœuds sont des appareils mobiles sans fil qui se doivent d'être autonomes au niveau énergétique. La gestion de l'énergie influe sur les algorithmes de routage.

L'échange robuste des données entre deux entités via un réseau ad hoc s'appuie sur des algorithmes de routage capables de détecter les changements topologiques et de s'y adapter. De manière générale, on distingue deux classes de protocoles pour traiter le problème du routage dans les réseaux ad hoc : les protocoles proactifs, et les protocoles réactifs (ou à la demande).

Les protocoles proactifs tentent de maintenir de l'information de routage vers toutes les destinations connues, et ce dans tous les nœuds. Ce type de protocole oblige à un échange régulier des modifications topologiques, pour que le routage reste cohérent. L'avantage principal est qu'on a tout de suite connaissance d'un chemin lorsqu'on veut débiter un transfert vers une destination précise. L'inconvénient majeur est que beaucoup de ressources sont utilisées pour la découverte et la maintenance de routes, qui ne seront finalement pas utilisées.

Les protocoles réactifs tentent d'éviter la dépense d'énergie inutile qu'induisent les protocoles proactifs. Avec un protocole réactif, lorsqu'un nœud veut débiter un transfert, une requête de route est diffusée (*broadcast*). Lorsqu'elle atteint la destination, elle revient. On peut ainsi découvrir plusieurs routes, et choisir celle qui semble la meilleure. Par contre, ce procédé induit un effet désastreux appelé "Broadcast Storm Problem". Pour éviter cela, diverses méthodes peuvent être employées, délai aléatoire de retransmission du message de broadcast, requêtes limitées en nombre de sauts,...

3.3.3.2. Bluetooth et le routage ad hoc

Une interface Bluetooth consiste en un émetteur/récepteur radio utilisé à des fréquences autour de 2,4 GHz. Un canal est représenté par une séquence pseudo-aléatoire de sauts de fréquence entre les 79 ou 23 fréquences disponibles (selon les pays). Le canal est divisé en périodes de temps constantes. Chacune de ces périodes correspond à l'utilisation d'une fréquence de la séquence de sauts définie.

Les unités Bluetooth communiquant par un même canal forment un piconet. Une unité est maître, les autres sont esclaves. La séquence de saut est unique pour le piconet, elle est déterminée par le maître du piconet. Au maximum, sept unités esclaves peuvent communiquer avec le maître dans le piconet à un instant donné. Mais bien plus d'unités esclaves peuvent être répertoriées par le maître et se tenir dans un état parking (park state). Plusieurs piconets ayant des unités en commun forment un scatternet¹. Chaque piconet ne peut avoir qu'un seul maître. Donc, des unités esclaves peuvent appartenir à plusieurs autres piconets en tant qu'esclave également, mais à un seul autre en tant que maître. De même, une unité maître ne peut appartenir à d'autres piconets simultanément qu'en tant qu'esclave. Chaque piconet d'un scatternet possède son propre canal ; les différents piconets ne devraient pas se trouver synchronisés en temps et/ou en fréquence.

Le but premier de cette technologie est de remplacer le câblage de la plupart des appareils (clavier d'ordinateur, écouteur,...). Pour ce faire, une puce Bluetooth est capable d'établir des liaisons point-à-point avec d'autres interfaces Bluetooth.

Cette technologie est a priori intéressante à utiliser dans le monde des réseaux mobiles. Son faible coût annoncé et certaines de ses caractéristiques (formation de piconets et de scatternets) en font un bon candidat pour la plupart des applications existantes et annoncées pour les réseaux mobiles. Cependant l'établissement des liaisons, la consommation d'énergie, les performances en terme de débit et la taille des paquets constituent autant de limitations lorsqu'il s'agit d'effectuer du routage ad hoc. La recherche de solutions réalistes à ces problèmes fait partie des préoccupations de la "communauté Bluetooth".

¹ Voir <http://www.bluetooth.com/>

4. Domaines d'application

4.1. Introduction

Mots clés : *Systèmes embarqués, applications mobiles, ordinateurs de poche, systèmes d'information spontanés.*

Les activités du projet ACES s'appliquent à des domaines différents, dans lesquels les logiciels ont pour principale propriété d'être embarqués. Etant donnée l'utilisation grandissante de l'informatique embarquée, nous ne tentons pas d'être exhaustifs sur les domaines d'applications de nos travaux. Citons à titre d'exemples les applications bancaires (carte à puces), de télécommunications (téléphonie mobile notamment), automobile, avionique, production d'énergie, contrôle-commande, électronique grand-public (décodeurs), ordinateurs de poche, SoC (Systems on Chip).

Nous détaillons dans les paragraphes qui suivent uniquement les domaines dans lesquels nos travaux de recherches sont actuellement et effectivement appliqués dans le cadre de collaborations industrielles, en passant volontairement sous silence les domaines d'utilisation potentiels de nos travaux.

4.2. Ordinateurs de poche pour applications multimédias embarquées

Mots clés : *Java, multiprocesseurs hétérogènes, ordonnancement 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 énergétique. C'est dans ce contexte que se place notre collaboration avec Texas Instruments (voir les paragraphes 6.3 et 7.1.1).

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 sont 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 devons 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 l'environnement d'exécution Java, 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, nous travaillons sur le problème de performance de l'environnement d'exécution Java et une approche basée sur un couplage fort matériel/logiciel est à l'étude.

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 donnent 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 le paragraphe 6.6). 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, ses 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 ordinateur 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 ordinateur 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.

4.4. Navigation Spatiale et navigation Web

Mots clés : *Navigation spatiale.*

Les systèmes d'informations grande échelle comme le Web sont aujourd'hui potentiellement accessibles depuis n'importe où, grâce à l'informatique et aux communications mobiles. Cependant, leur exploitation par les utilisateurs mobiles est complexe lorsque l'utilisateur doit rechercher des informations en rapport avec sa situation réelle, comme par exemple l'heure d'un vol en arrivant dans un aéroport. En effet, les informations sur le Web sont liées entre elles par des liens reflétant une proximité thématique, ce qui permet à l'utilisateur de naviguer autour d'un sujet après avoir trouvé une première page intéressante, mais la localisation de cette première page est problématique, nécessitant souvent une recherche qui peut être longue et fastidieuse, procédure inadaptée dans le cas d'une utilisation mobile : le système d'information est découplé de l'espace physique dans lequel évolue l'utilisateur (aéroport, gare, musée...).

La navigation spatiale telle que nous la définissons permet de proposer une approche radicalement différente à ce problème : le système d'information est construit directement à partir d'objets physiques ; plus précisément, le contenu et la structure des informations sont déduits automatiquement à partir de la disposition d'objets dans l'espace. Par exemple, dans le cas d'un musée, les informations d'orientation, de description des œuvres, de liens thématiques entre les œuvres constituent son système d'information. Cette solution permet sa mise en œuvre et son exploitation implicite, une fois les objets mis en place dans les salles d'exposition. Elle offre l'avantage unique de suivre dynamiquement les évolutions apportées aux expositions : déplacement, ajout, suppression d'objets.

Ce domaine est en plein développement actuellement du fait de l'intégration de plus en plus importante "d'intelligence" dans l'environnement et les objets. Une expérimentation dans les locaux de l'IRISA, à base de PDAs et de système radio "courte portée" montre la simplicité de réalisation d'un système "Web contextuel", directement dérivé de l'espace physique (bureaux, équipes, personnes...), et son exploitation par un piéton qui navigue naturellement à travers les informations en se déplaçant. Le navigateur Web couple la recherche d'information sur le Web traditionnel (liens hypertextes et proximité thématique) à une navigation "physique" (proximité physique).

Ce modèle de système d'information pourrait rapidement trouver des applications réelles que ce soit dans l'environnement urbain (renseignements sur les magasins, édifices etc.) ou plus localement par exemple dans un musée comme cela a été décrit ci-dessus. Notons que ce système est particulièrement bien adapté au profilage fin du comportement des utilisateurs : il est possible de connaître les chemins suivis par les utilisateurs, ou le temps de présence dans les différentes zones couvertes par le système. Il est également possible de personnaliser le système d'information et de recommander des circuits particuliers, en fonction du profil de l'utilisateur.

5. Logiciels

5.1. Introduction

Les travaux de recherche conduits dans le projet ACES donnent lieu au développement de nombreux logiciels. Ces logiciels sont pour la plupart réalisés, ou tout du moins initialisés dans le cadre de partenariats avec des industriels, et par conséquent attachés aux domaines d'application couverts par le projet.

5.2. Environnement d'exécution Java pour architecture embarquée

Participants : *Michel Banâtre, Gilbert Cabillic [correspondant], Jean-philippe Lesot, Salam Majoul, Frédéric Parain, Jean-Paul Routeau.*

Statut du logiciel : *Interne au projet.*

Le contrat correspondant est détaillé dans le paragraphe 7.1.1. 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). Cet environnement permet de construire, à l'aide d'un ensemble d'APIs Java (bibliothèques) et d'exécuter différents types d'applications.

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. Outre la performance il est également possible d'agir sur l'expansion mémoire introduite par la machine Java et sur la consommation énergétique de ses traitements. Afin de réaliser cette modularité, nous avons été amenés à écrire la totalité de notre environnement d'exécution Java ainsi que l'outil de génération associé.

En outre, un module particulier 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 compatibles POSIX comme Windows ou Linux.

5.3. Environnement pour la mise en œuvre des systèmes d'information spontanés

Participants : *Michel Banâtre, Gilbert Cabillic, Paul Couderc, David Touzet, Arnaud Troël, Stéphane Tudoret, Grégory Watts, Frédéric Weis [correspondant].*

Statut du logiciel : *Interne au projet.*

Le contrat correspondant est détaillé dans le paragraphe 7.1.2. L'objectif de ces développements est de mettre en œuvre une plate-forme matérielle et logicielle permettant d'expérimenter les mécanismes propres aux SIS, notamment la gestion efficace des interactions de proximité, la construction dynamique de l'espace visible au sein du SIS, les interactions spontanés au sein de cette espace visible.

La plate-forme matérielle est constituée de PDAs fonctionnant sous Windows CE, et disposant de cartes de communication sans fil 802.11b. Cette configuration s'enrichira à terme de cartes Bluetooth, aptes à mettre en œuvre le principe de communication par voisinage spécifique aux SIS (elles offrent une portée de communication de l'ordre d'une dizaine de mètres). Les modules prenant en charge les mécanismes de base des SIS (décrit dans le paragraphe 6.6) ont été réalisés et expérimentés : détection de présence des nœuds voisins dans le voisinage physique, maintien dynamique de la liste des nœuds (en fonction des entrées et des sorties) présents dans le SIS, construction et présentation de l'espace visible à l'utilisateur.

5.4. Heptane

Participants : *Mathieu Avila, Isabelle Puaut [correspondante].*

Statut du logiciel : *En cours de dépôt à l'APP en vue d'une diffusion en open-source.*

Heptane (Hades Embedded Processor Timing ANalyzEr) est un logiciel d'obtention automatique de temps d'exécution au pire cas. Heptane, par analyse statique du code source d'un programme C, retourne une borne supérieure de son temps d'exécution sur une architecture Intel Pentium. Heptane exploite une modélisation de la micro-architecture du processeur (exécution pipelinée, cache d'instructions, prédicteur de branchement) de manière à évaluer les temps de manière la plus précise possible tout en donnant une borne supérieure du temps d'exécution.

La conception de ce logiciel a été initialisée lors de la coopération Inria/DGA menée en collaboration avec la société Dassault-Aviation. Le logiciel est développé principalement en Objective Caml et s'exécute sur Solaris et Linux.

5.5. Artisst

Participants : *David Decotigny, Isabelle Puaut [correspondante].*

Statut du logiciel : *En cours de dépôt à l'APP, en vue d'une diffusion en open-source.*

Artisst (Artisst is a Real-Time System Simulation Tool) est une infrastructure de simulation à événements discrets permettant d'évaluer les performances de systèmes (application et système d'exploitation) temps-réel

mono ou multiprocesseurs. Une simulation est découpée en modules, qui communiquent par échanges de messages. Les modules sont de trois types :

- Des modules d'entrée modélisant l'environnement extérieur au système et générant des messages à partir de lois statistiques ou de fichiers de trace ;
- des modules "système" représentant les logiciels s'exécutant sur chacun des calculateurs composant le système temps-réel (application et système d'exploitation).
- des modules de sortie, permettant d'exploiter les messages générés par les modules systèmes en générant des fichiers de trace ou représentant ces messages sous forme graphique.

Artisst fournit par défaut un ensemble de modules, qui peuvent être étendus et/ou remplacés (par exemple, changement de l'ordonnanceur du système d'exploitation, ajout de primitives de synchronisation entre tâches) grâce à la conception orientée-objets d'Artisst. Le logiciel s'exécute sur Solaris et Linux.

5.6. Spread

Participants : Michel Banâtre[correspondant], Mathieu Bécus, Paul Couderc.

Statut du logiciel : Interne au projet.

Le logiciel SPREAD (Spatial PRogramming Environment for Ambient computing Design) permet la construction simple d'applications dans le domaine de l'ubiquité numérique. Il permet de définir de nouvelles abstractions de programmation dont les propriétés qui les caractérisent dépendent des propriétés de l'espace physique considéré. Ceci nous permet de définir un modèle de programmation spatiale dans lequel la structure et les flots de contrôle dépendent directement des entités physiques associées et de leurs relations de dépendance spatiales (e.g. "...proximité de..."). Comme cela est détaillé dans la section 6, ce modèle s'appuie sur la notion de tuple space initialement proposée dans LINDA[22].

L'architecture de SPREAD s'articule autour de trois composants principaux :

- Le gestionnaire de l'espace de tuple qui gère le stockage des tuples et l'accès à ces tuple au travers des requêtes émises par les applications.
- Un serveur de tuples qui traite des accès aux espaces de tuples distants au moyen d'opérations get/put.
- Le système de découverte qui traite de l'évolution spontanée des espaces de tuples en fonction de la portée des systèmes de communication sous-jacents et de la mobilité des utilisateurs.

La plate-forme matérielle actuelle est constituée de iPaq fonctionnant sous Windows CE, et disposant de cartes de communication sans fil 802.11b. Le logiciel SPREAD est écrit en C, le système de découverte et l'accès aux tuples distants reposent sur le protocole UDP.

6. Résultats nouveaux

6.1. Introduction

Le programme de recherches du projet ACES s'articule autour de deux axes principaux ayant été brièvement introduits dans le paragraphe 2.1. Le premier axe concerne la conception de systèmes embarqués. Une architecture embarquée est le plus souvent caractérisée à la fois par une forte interaction avec l'environnement extérieur dans lequel il évolue, imposant des contraintes de temps-réel plus ou moins fortes selon le type d'utilisation de l'architecture, ainsi que par le caractère limité des ressources dont elle dispose, que ce soit en terme de capacité de stockage, d'énergie, ou de bande passante pour les communications vers le monde extérieur. Un premier axe d'étude autour de la construction de systèmes d'exploitation embarqués concerne la caractérisation des ressources consommées par les logiciels embarqués, en terme de mémoire, d'énergie et de temps de réponse (paragraphe 6.2). Cette caractérisation est nécessaire non seulement pour le dimensionnement des architectures embarquées, mais aussi à la mise en place de politiques de gestion

de ressources dans les systèmes d'exploitation qui sont adaptées à leur caractère limité et aux interactions avec l'environnement extérieur. La construction effective de systèmes d'exploitation embarqués constitue un deuxième axe d'étude (paragraphe 6.4), dans lequel nous examinons à la fois les couches basses des systèmes d'exploitation (noyaux de systèmes, avec par conséquent une forte connexion avec l'architecture embarquée sous-jacente) et des couches exécutives de type intergiciel (middleware) comme la machine virtuelle Java. Enfin, l'évaluation des performances des systèmes embarqués mérite une attention toute particulière. Nous présentons nos travaux sur ce thème dans le paragraphe 6.5. La seconde activité est orientée vers l'étude des supports systèmes pour l'ubiquité numérique, avec deux axes de recherche, un premier autour de la création de systèmes d'information spontanés (voir paragraphe 6.6) et un deuxième autour des systèmes d'information spatiaux (voir paragraphe 6.7).

6.2. Caractérisation de la consommation en ressources des logiciels embarqués

Participants : Alexis Arnaud, David Decotigny, Isabelle Puaut, Gilbert Cabillic.

Les ressources limitées des architectures embarquées, ainsi que les fortes interactions des logiciels embarqués avec l'environnement extérieur rendent nécessaire, pour une gestion de ressources appropriée, de *caractériser*, de manière la plus automatisée possible, les logiciels embarqués, d'un point de vue comportement temporel, énergie et mémoire consommées. Cette caractérisation peut aider aux choix et au dimensionnement de l'architecture matérielle (puissance de calcul, capacité mémoire et de stockage). Par ailleurs, dans le cadre de logiciels embarqués ayant des contraintes de temps-réel strict, la caractérisation du comportement temporel *pire-cas* des logiciels est nécessaire pour l'analyse d'ordonnabilité du système, qui est critique dans ce cas pour prouver le respect des échéances du système. De manière plus générale, la caractérisation des logiciels embarqués peut être utilisée dans une démarche d'optimisation de ces logiciels (hors-ligne, par exemple intégrée au processus de compilation des applications, ou en-ligne, dans le système d'exploitation, pour adapter la gestion des ressources de l'architecture embarquée aux besoins des applications).

Une partie de nos travaux a porté sur la caractérisation de la consommation en ressources de logiciel embarqué par instrumentation de l'exécution de l'application (méthode de profiling). À ce jour, ce type de méthode a été utilisé pour estimer les temps de réponse moyens des logiciels embarqués, et sont de ce fait adaptés à des logiciels temps-réel souple. Nous étendons actuellement ces travaux à l'estimation de la consommation énergétique d'applications multimedia.

Une deuxième partie de nos travaux a porté sur la caractérisation du comportement temporel *pire-cas* de logiciels (obtention de temps d'exécution au pire-cas de tâches, ou WCET pour Worst-Case Execution Time). Ces travaux s'appliquent donc aux logiciels ayant des contraintes de temps-réel *strict*. Les WCETs sont ici obtenus par *analyse statique* du code source et objet des logiciels. Une des difficultés si l'on utilise ce type de méthode est d'éviter l'obtention de WCET 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.

Nos contributions dans le domaine de l'obtention de WCET par analyse statique résident à la fois dans la définition d'une *méthode* d'obtention de WCET sûrs et précis, et dans le développement d'un *outil* mettant en œuvre cette méthode. La méthode proposée est une méthode à base d'arbre (*tree-based*). Elle consiste à calculer le WCET d'un programme en effectuant un parcours hiérarchique de bas en haut de son arbre syntaxique, le WCET d'un nœud de l'arbre étant utilisé pour calculer le WCET de son père. Les apports de la méthode sont de réduire les deux principales sources de pessimisme de l'analyse, provenant respectivement de l'analyse de haut et de bas niveau :

- La première source de pessimisme est la difficulté d'identifier précisément le pire chemin d'exécution dans le programme, en particulier le nombre maximum d'itérations des boucles. Nous proposons à cet effet une méthode d'annotation des boucles, qui permet de décrire de manière précise le comportement des boucles imbriquées dites non rectangulaires.

- La deuxième cause du pessimisme est due aux éléments d'architectures, qui améliorent les performances moyennes du processeur, mais posent des difficultés pour identifier de manière sûre mais pas trop pessimiste son comportement temporel au pire cas. Nous proposons des méthodes de prise en compte pour trois de ces éléments (le cache d'instructions, la prédiction de branchement et le pipeline), et adaptons la méthode de calcul des WCET par parcours de l'arbre syntaxique du programme de manière à prendre en compte de manière conjointe ces trois éléments d'architecture.

La méthode proposée a été intégrée dans un outil nommé Heptane (voir paragraphe 5.3), analysant des programmes C et calculant leur WCET sur une architecture Intel Pentium. La structure modulaire de l'analyseur est conçue pour pouvoir aisément l'adapter à de nouvelles architectures cibles, nouvelles méthodes de prise en compte de la micro-architecture ou nouveaux langages de programmation. Cet outil a été utilisé pour analyser des logiciels temps-réel, allant de petits programmes de tests à un noyau de système temps-réel embarqué [9], permettant ainsi d'évaluer les performances de la méthode d'analyse.

Ces travaux sur l'analyse statique de WCET seront prolongés dans plusieurs directions :

- *Analyse de bas niveau : meilleure prise en compte de l'architecture matérielle.* Les processeurs embarqués, bien que de structure plus simple que les calculateurs haute performance, suivent les évolutions de ces derniers avec quelques années de retard. Ainsi, il semble important de prendre en compte dans l'estimation des WCETs des mécanismes avancés présents dans les processeurs haute performance de manière à disposer des modèles correspondants dès leur intégration dans les processeurs embarqués. Des exemples d'éléments non pris en compte dans les analyses statique de WCET sont les caches de trace, l'exécution multiflot, l'exécution spéculative, les prédicteurs de branchement élaborés ou encore l'exécution non ordonnée. Ces travaux seront menés en partie en collaboration avec l'équipe Apara de l'IRIT (P. Sainrat, C. Rochange) avec lesquels des travaux commun ont déjà débuté (ces travaux [13] font le point sur les méthodes d'analyse statique de WCET avec un accent particulier sur les aspects architecture).
- *Extension de la méthode d'analyse à d'autres métriques.* D'autres ressources que le temps processeur, comme par exemple la consommation énergétique, sont primordiales dans les systèmes embarqués. Une voie de recherche à explorer est de transposer les méthodes d'analyse statique de WCET pour obtenir la connaissance a priori de mesures comme la consommation énergétique au pire-cas de logiciels embarqués.

Enfin, obtenir les WCET des programmes par analyse statique présente deux inconvénients qui peuvent être jugés inacceptables selon le logiciel à évaluer et l'architecture cible. D'une part, l'utilisation d'une méthode d'analyse statique nécessite d'avoir accès au code source du logiciel à analyser. D'autre part, il est indispensable d'avoir des informations précises sur la structure interne du processeur, ce qui est de plus en plus difficile (sans contact privilégié avec un constructeur) étant donnée la complexité grandissante des processeurs. Ces deux raisons nous amènent actuellement à étudier les outils de caractérisation de temps de réponse de type boîte noire, nécessitant moins d'informations sur le matériel et le logiciel, au détriment de la sûreté des estimations. L'idée est ici d'obtenir les temps d'exécution par exécution du logiciel, en orientant la génération des données d'entrée du logiciel de manière à se placer autant que faire se peut dans le scénario aboutissant au pire temps de réponse du logiciel. La méthode triviale consistant à générer toutes les configurations possibles de paramètres d'entrée posant des problèmes d'explosion combinatoire, il est nécessaire de réduire la taille de l'espace de recherche. Soulignons aussi qu'un des problèmes est de générer des données d'entrée pour le logiciel qui soient valides, afin d'évaluer le pire temps de réponse du logiciel et non pas le temps de réponse en présence de fautes d'origine logicielle.

6.3. Environnement Java embarqué pour ordinateurs de poche

Participants : Michel Banâtre, Gilbert Cabillic, Jean-Philippe Lesot, Salam Majoul, Frédéric Parain, Jean-Paul Routeau, Pushpendra Singh.

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 (*appliance*). 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.

Le contexte de nos recherches est illustré sur la figure 1. La machine d'exécution Java utilise les services du système d'exploitation afin de s'exécuter au-dessus de l'architecture matérielle. Cette machine Java est en charge de l'exécution de plusieurs applications Java différentes. L'architecture matérielle peut posséder un ou plusieurs processeurs pouvant être hétérogènes, différents types de mémoires, des périphériques divers (écran, clavier, son, réseau, carte de stockage disque) ainsi qu'une batterie. Nos travaux actuels se focalisent sur différents axes de recherches complémentaires que nous présentons ci-après.

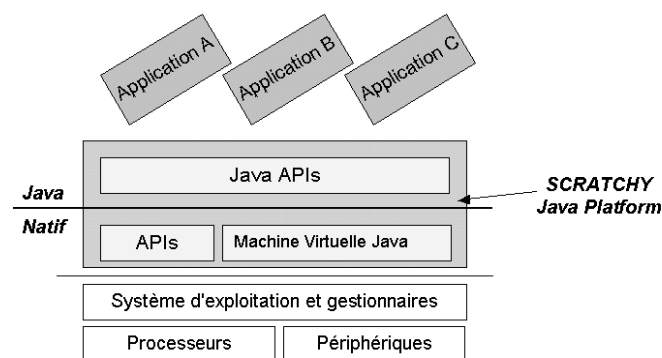


Figure 1. Contexte.

6.3.1. Machine d'exécution Java modulaire (Scratchy)

Les travaux menés autour de la machine d'exécution Scratchy ont pour objectif de permettre de travailler sur le compromis lié aux contraintes d'embarquabilité de l'architecture d'un *appliance*. En effet, en fonction de la fois des possibilités de l'architecture mais également des besoins des applications, le compromis entre volume mémoire, performance temporelle et consommation énergétique d'un code peut s'avérer différent.

Afin d'agir sur ces trois paramètres, nous avons basé notre approche sur une décomposition modulaire de notre machine d'exécution Java nommée Scratchy. Dans un premier temps nous avons conçu un outil de développement qui permet, depuis un ensemble de modules indépendants de générer le code source final de notre environnement Java. Cet outil génère automatiquement les appels entre les services des modules et permet de plus de réaliser des optimisations sur l'allocation mémoire des structures utilisées par les différents modules. Il permet également de choisir parmi plusieurs versions disponibles d'un service. Par exemple, entre une version performante qui génère plus d'expansion mémoire avec une version moins gourmande en énergie, mais moins performante. Dans un second temps, nous avons écrit en partant de rien la totalité de notre environnement d'exécution Java en le décomposant en un ensemble de modules. Ces différents aspects sont détaillés dans Nous avons introduit deux modules spécifiques qui permettent à la fois l'exécution de Scratchy au-dessus d'un multiprocesseur hétérogène et la gestion des interactions avec le système d'exploitation sous-jacent. Comme indiqué dans le paragraphe 4.1, notre environnement d'exécution Java est compatible avec la spécification CLDC² de Sun. Néanmoins, dû au support de la gestion de l'arithmétique flottante et des threads, notre environnement est très proche de la spécification CDC20. Les processeurs Pentium mono-processeur et bi-processeur sont complètement supportés et les processeurs DSP C55X et ARM9 le sont partiellement.

²CLDC, CDC and the K Virtual Machine (KVM). Sun microsystems. <http://java.sun.com/products/cldc/>, 2000.

Enfin, les systèmes d'exploitation cibles sont les systèmes Linux, Windows ainsi que VxWorks qui est un système temps-réel commercial.

Nous étudions actuellement l'évolution des fonctionnalités de Scratchy afin d'atteindre plusieurs objectifs :

- *Performance temporelle.* Notre solution est basée sur un couplage fort matériel/logiciel qui exploite les possibilités de notre approche modulaire afin d'obtenir un environnement d'exécution Java très performant. De plus, nous étudions la façon d'améliorer la performance de la gestion mémoire de la machine Java. Tout d'abord, comme la machine Java suppose la présence d'un allocateur mémoire permettant d'allouer la mémoire des objets ainsi que les structures de données nécessaires à son fonctionnement, nous étudions les besoins de la machine à l'aide de traces d'exécution afin de proposer une stratégie de gestion mémoire qui permette d'améliorer la performance de manière significative. De plus, nous concevons également un ramasse miette adaptée aux applications Java embarquée qui permettra de diminuer le temps d'exécution nécessaire à l'identification des objets devenus inutiles en mémoire.
- *Sécurité.* Même si l'environnement Java permet un fort degré d'isolement entre les différentes applications s'exécutant sur une même machine d'exécution Java, il n'est pas suffisant pour garantir un fonctionnement fiable face à des applications erronées voire malveillantes, en particulier à cause de l'utilisation de bibliothèques natives à travers JNI. Après l'analyse qualitative et quantitative des risques pesant sur cet environnement nous proposerons une stratégie de sécurité adaptée à la machine Java en utilisant les possibilités offertes par notre approche modulaire. Ces différents aspects sont présentés dans [4].

6.3.2. Ordonnancement multicritère d'applications multimédias

L'utilisation des 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.

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. 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). Nous avons également introduit dans ce modèle la notion de mode d'exécution qui permet de prendre en compte dynamiquement les différents comportements des applications en fonction de leurs données en entrées. Enfin, afin de permettre à un concepteur d'une application d'exprimer ce modèle en Java, nous avons défini et réalisé une Api Java qui permet de représenter ce modèle.

Après avoir étudié les techniques de réduction de la consommation pour les systèmes embarqués, nous définissons actuellement une politique d'ordonnancement multi-critère compatible avec ce modèle. Cette politique est définie afin 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,
- exécution 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.

Afin d'évaluer le comportement de la règle de placement, un simulateur a été réalisé dans le but de simuler une architecture multiprocesseur hétérogène. De plus, afin d'obtenir les informations propres à chaque application nécessaires à la réalisation de la politique d'ordonnement (estimation du temps d'exécution moyen et estimation de la consommation énergétique moyenne des traitements d'une application), nous examinons la conception d'une technique de caractérisation ("profiling") d'applications multimédia qui permettra d'obtenir simplement ces estimations [3].

6.3.3. Disponibilité pour environnement d'exécution sans fil

Cette recherche a pour objectif de fournir des solutions à la disponibilité adaptées aux appareils et à leur contexte d'utilisation. Nous avons réalisé une classification des types de fautes dans ce contexte (fautes dues au réseau, au manque d'autonomie, à des fautes logicielles, ...) ainsi qu'à l'étude des différentes approches de sûreté de fonctionnement existantes adaptées aux appareils. Nous travaillons actuellement à la conception d'un ensemble de nouvelles techniques permettant d'accroître la disponibilité d'un appareil. Ainsi nous étudions les possibilités de point de reprise et de restauration pouvant être offertes à l'aide d'une machine d'exécution Java. Afin de diminuer le coût temporel nécessaire à la capture de l'état d'une application Java, notre approche consiste à coupler la phase de ramasse miette (permettant d'identifier l'ensemble des objets atteignables d'une application) avec la phase de capture. Parmi les extensions de cette solution, nous étudions également la possibilité de réaliser une capture incrémentale afin de diminuer le volume mémoire nécessaire à son stockage.

6.4. Noyaux de systèmes d'exploitation embarqués

Participants : Gilbert Cabillic, Isabelle Puaut.

6.4.1. Mécanismes de base pour les systèmes d'exploitation embarqués

Cette activité concerne l'étude des mécanismes de base à fournir par les noyaux de systèmes embarqués pour prendre en compte les contraintes des applications embarquées (temps-réel notamment) et celles des architectures (ressources mémoire et énergie limitées, présence d'éléments tels que des caches, de la mémoire statique sur-puce, support pour la traduction d'adresses). Dans ce contexte, deux points attirent plus particulièrement notre attention, et concernent l'impact d'éléments architecturaux présents dans les architectures embarquées sur les systèmes d'exploitation.

D'une part, de plus en plus de processeurs embarqués disposent de mémoire à accès rapide sur puce (mémoires caches ou mémoire statique de type scratchpad). La présence de mémoires caches pose un problème de déterminisme dans le cadre d'applications embarquées ayant des contraintes de temps-réel strict, car les caches sont conçus pour améliorer les performances des logiciels dans le cas moyen et pas dans le pire des cas. De plus, l'utilisation de caches impose un délai supplémentaire pour le rechargement du cache après un changement de contexte entre processus. Afin d'exploiter au mieux les caches existants dans l'architecture embarquée, nous explorons actuellement l'utilisation de méthodes de *verrouillage statique* du contenu du cache, de manière à rendre à la fois les temps d'accès aux caches et les temps de changement de contexte entièrement prévisibles, rendant ainsi leur utilisation dans un cadre temps-réel aisée. Toutefois, le processus de sélection du contenu du cache pose des problèmes d'explosion combinatoire (il n'est pas en effet possible d'analyser le comportement du système avec tous les contenus possibles du cache). Des algorithmes de sélection des contenus du cache ont été définis, la métrique que nous avons utilisée pour sélectionner les contenus étant basée sur le comportement temps-réel au pire-cas du système. Il est prévu d'étendre ces algorithmes en optimisant la consommation énergétique, particulièrement importantes dans le cadre des systèmes embarqués, et des les transposer aux mémoires de type scratchpad. Ces travaux sur l'utilisation de mémoires rapides dans le cadre de systèmes temps-réel embarqués fait l'objet d'une coopération bilatérale avec l'Universidad Politécnica de Valencia (Programme d'Action Intégrée Picasso, voir paragraphe 8.1.1).

D'autre part, on voit de plus en plus de supports matériels pour des espaces d'adressage séparés (support matériel pour la traduction d'adresses, MMU pour Memory Management Unit et cache de traduction d'adresses associés) dans les processeurs embarqués. La motivation de l'introduction d'un tel mécanisme est la sécurisation des applications, notamment en présence de fautes logicielles. La présence de MMU

pose des problèmes dans les systèmes embarqués, tant du point de vue de la mémoire consommée pour la traduction d'adresses et des temps d'accès à la mémoire (problème de prévisibilité dû à l'utilisation du cache de traduction d'adresses, particulièrement important pour les applications embarquées temps-réel strict). Des travaux sont nécessaires pour tirer parti au mieux du support matériel (séparer les applications entre elles) tout en respectant leurs contraintes (consommation mémoire et énergie adaptés, temps d'accès mémoire bornés et faibles pour les applications temps-réel strict). Plus généralement, de la même manière qu'il est utile de proposer un support pour l'isolation spatiale entre applications, une voie de recherche est la fourniture par le systèmes d'exploitation de mécanismes d'isolation temporelle, permettant notamment de faire cohabiter des applications ayant des contraintes temporelles différentes (strictes et souples).

6.4.2. Système d'exploitation embarqué construit en Java

Cette activité consiste à relever le défi d'un problème peu abordé actuellement dans la littérature. Convaincus qu'une solution performante à l'exécution d'un code Java pourra être disponible dans les années à venir, nous abordons ici la construction d'un noyau de système d'exploitation embarqué en utilisant Java comme langage de programmation. Ce type de système d'exploitation a la particularité d'être téléchargeable à distance et être indépendant de l'architecture, ce qui offre des avantages tant de développement que de maintenance évidents. De plus, afin de profiter des fonctionnalités de sécurité présentes dans le langage Java ainsi que des outils évolués permettant d'écrire un système d'exploitation plus robuste, nous avons le souci de préserver, à l'opposé des approches existantes, la non représentation ou manipulation de pointeurs en Java. Bien sûr un très petit noyau de système et une machine d'exécution Java écrits en code natif sont toujours nécessaires, mais seules ces deux pièces logicielles doivent être portés sur une nouvelle architecture sans remettre en cause la partie principale du système d'exploitation écrit en Java.

L'objectif de ce travail consiste donc à définir le point d'encrage entre le monde natif et le monde Java, qui permette de construire un système d'exploitation embarqué permettant d'exécuter des applications Java. Nous avons actuellement permis la représentation dans le monde Java des notions d'interruption et de contexte de threads, et nous avons validé la construction d'un ordonnanceur à temps partagé des threads Java construit à l'aide de l'interruption du temps. Nous travaillons à l'identification des autres entités de construction ainsi qu'à la définition du support d'exécution natif le plus petit permettant d'exécuter ce genre de système (voir [4]). Ce petit système d'exploitation permet de 'transférer' dans le monde Java les notions usuelles nécessaires à la construction d'un système d'exploitation plus gros comme la gestion des interruptions ou l'allocation mémoire bas-niveau. Une fois ces notions définies, nous validerons la construction en Java des fonctionnalités d'un système d'exploitation comme les gestionnaires de périphériques, le ramasse miette ou la gestion multi-application. Comme indiqué précédemment, le problème principal de cette approche est de définir un système d'exploitation qui ne possède pas directement la notion de pointeur mémoire et qui soit représentable et compatible avec les entités du langage Java.

6.5. Evaluation de performances des systèmes embarqués

Participants : *Gilbert Cabillic, David Decotigny, Frédéric Parain, Isabelle Puaut.*

Un aspect important lors de la construction de systèmes d'exploitation embarqués est d'évaluer les performances des mécanismes de gestion de ressources qu'ils intègrent. Nous explorons deux voies complémentaires pour cette évaluation de performances.

La première est l'analyse de performances en grandeur réelle via la construction effective des systèmes ; cette première approche est utilisée dans l'environnement Java embarqué décrit dans le paragraphe 6.1 Ce type d'approche présente l'intérêt d'être précise, mais en revanche a un coût de mise en œuvre important.

Nous explorons également l'utilisation de méthodes de simulation pour l'évaluation de performances des systèmes. L'intérêt de ce mode d'évaluation des performances est de ne pas requérir de connaissance statique des caractéristiques temporelles du système et de son environnement. Ainsi, contrairement aux méthodes d'analyse (statique) d'ordonnancement, elles peuvent être utilisées pour valider une large classe de systèmes temps-réel, dont les systèmes temps-réel souples, pour lesquels bien souvent on ne connaît pas statiquement de manière sûre toutes les caractéristiques temporelles. Bien que moins précise qu'une expérimentation

sur machine cible, la simulation d'un système temps-réel est plus légère à mettre en place que le système final, et ainsi permet, à un coût de mise en œuvre raisonnable, d'évaluer les performances d'une politique d'ordonnancement ou de l'architecture d'un système d'exploitation. Nous avons conçu un simulateur à événements discrets modulaire adapté à la simulation de systèmes embarqués temps-réel. L'infrastructure de simulation sur laquelle nous travaillons, nommée Artisst [6][7], pour "*Artisst is a Real-Time System Simulation Tool*", est orientée vers la simulation de systèmes d'exploitation temps-réel, et sa modularité permet d'évaluer de manière précise différentes politiques d'allocation de ressources (ordonnanceur du système d'exploitation notamment).

6.6. Systèmes d'information spontanés (SIS)

Participants : Michel Banâtre, Carole Bonan, Gilbert Cabillic, Paul Couderc, David Touzet, Arnaud Troël, Stéphane Tudoret, Grégory Watts, Frédéric Weis.

L'informatique mobile connaît actuellement un essor considérable dû au développement conjugué des communications sans fil et des technologies embarquées. Ces évolutions récentes peuvent par exemple être illustrées par le succès de la téléphonie mobile. Ces architectures cherchent pour la plupart à déployer un ensemble de services homogènes sur la totalité de leur zone de couverture. Le système WAP, par exemple, propose à ses utilisateurs l'accès au même ensemble de serveurs Web, quelque soit leur position géographique.

Un certain nombre d'études se distinguent aujourd'hui de cette approche classique. Pour ce faire, elles proposent d'exploiter la mobilité des utilisateurs, et plus généralement leur contexte, afin d'adapter les services rendus par un système à la situation des utilisateurs. L'ensemble de ces travaux est aujourd'hui regroupé sous l'étiquette ubiquité numérique. L'exploitation de ce principe permet d'envisager le déploiement de nouveaux types d'applications.

Notre approche à ces travaux 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). Dans ce cadre, le fonctionnement des nœuds mobiles s'affranchit de toute infrastructure, qu'elle soit de communication ou d'acquisition du contexte.

Notre objectif est de définir, dans le cadre d'un environnement mobile et embarqué, une architecture système permettant la mise en œuvre d'un SIS. Dans ce but, nos travaux actuels se focalisent sur trois axes de recherches : (i) la définition d'interactions sans fil de proximité efficaces entre des nœuds mobiles, (ii) la représentation des informations échangées entre les nœuds mobiles, (ii) et l'accès aux informations entre nœuds voisins, au sein d'un SIS. Nous décrivons dans la suite de ce paragraphe les résultats obtenus autour de ces trois axes.

6.6.1. Gestion des interactions de proximité au sein 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, taux d'erreurs variables...). 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. La prise en compte de ces paramètres dans le but d'améliorer les communications sans fil est un problème activement étudié dans de nombreux domaines d'applications, notamment pour améliorer les protocoles de routage dans les réseaux ad hoc.

Dans le cadre des SIS, l'originalité de notre approche est de prendre en compte les paramètres caractérisant la mobilité des nœuds voisins, non pas au niveau des protocoles de communication (comme c'est le cas dans les réseaux ad hoc), mais dans le cadre du système d'exploitation. Notre démarche est la suivante : nous proposons d'estimer le temps de communication "restant" ou *link expiration time* entre deux entités mobiles au sein d'un SIS, ce temps étant considéré comme une ressource par le système d'exploitation du nœud et devant

nous permettre de construire un schéma prédictif de communication. Afin de le calculer, il est nécessaire de s'appuyer sur des paramètres caractérisant la mobilité des nœuds. Ces paramètres peuvent être obtenus directement au niveau de la carte de communication sans fil du nœud : distance, qualité et puissance du signal reçu, taux d'erreurs. Il est également possible de faire appel à des systèmes externes de localisation, comme le GPS par exemple.

Afin de déterminer le temps restant de communication, deux solutions, reposant sur des estimations successives de la distance séparant les deux nœuds communicants, ont été proposées. La première s'appuie sur la méthode des moindres carrés, et la seconde sur une méthode d'estimation non linéaire. Notons que ces approches s'appuient essentiellement sur la distance euclidienne séparant les deux entités protagonistes. Cette mesure est difficile à obtenir au niveau d'une carte de communication sans fil. Toutefois, dans le cadre de nos simulations, la distance est équivalente à la puissance du signal reçu. Cette puissance est une mesure plus facile à obtenir dans la réalité. Dans le cadre de notre étude, dont un des objectifs est de montrer l'impact de la prédiction sur les échanges d'informations, cela s'avère efficace, et ces méthodes peuvent s'appliquer sans problème à partir d'une mesure reposant sur la puissance.

Le schéma prédictif permet 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 amené à étudier différentes politiques d'ordonnement visant à sélectionner le ou les blocs d'informations pouvant être transmis, en fonction des tailles et des priorités des blocs proposés par l'application d'une part, et de la "viabilité" de ces blocs d'autre part (estimée par rapport au taux d'erreurs du canal de communication et au temps de communication restant).

L'ensemble des schémas prédictifs ont été 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, et de pouvoir embarquer au sein de chaque entité simulée l'architecture de communication, propre aux systèmes d'informations spontanés. Cette étude a permis de démontrer que l'approche prédictive dans les communications au sein d'un SIS permet d'éviter d'entamer des émissions qui ne peuvent pas mener à leur terme, et donc d'améliorer l'efficacité dans l'utilisation de la bande passante.

6.6.2. Représentation des informations au sein d'un SIS

Toujours dans le cadre des communications de proximité, nous étudions le problème de la représentation des données. En effet, compte tenu de l'instabilité d'un SIS et des temps de communication limités, il n'est pas possible d'assurer le transfert atomique d'informations si elles sont trop volumineuses. De plus, le schéma prédictif peut être mis en défaut, à cause par exemple de l'imprécision relative des paramètres retenus pour caractériser la mobilité des nœuds. Il faut donc se doter d'un mode de représentation de l'information propre aux SIS. Notre idée initiale était de découper l'information en blocs élémentaires, chacun des blocs étant caractérisé par une taille et une priorité. Le transfert de l'information, assuré par le protocole de communication sous-jacent, porte sur chacun de ces blocs. Une des pistes suivies est d'appliquer des politiques de d'adaptation (i.e. de dégradation) pour générer ces blocs. Par exemple, une image (l'information à transmettre) peut donner lieu à différents types de représentations, avec une précision plus ou moins grande. Ces différents niveaux de précision constituent les blocs à transmettre, le choix s'effectuant en fonction du temps de communication estimé entre deux nœuds.

Nous étudions également une seconde approche, complémentaire de la précédente. Il s'agit de proposer un format des données échangées entre les nœuds d'un SIS qui permet d'exploiter une information même si elle n'a pas été totalement reçue par le nœud distant. Pour cela, un mode de représentation s'appuyant sur le langage HTML, appelé progressive HTML a été défini. Ce dernier permet de structurer un document sous une forme arborescente, l'importance de chaque partie de ce document étant fonction de sa profondeur dans l'arbre (plus l'information est loin de la racine, moins elle est importante). Nous avons ensuite associé ce mode de

représentation des données à une fonction de transmission entre les nœuds mobiles d'un SIS, afin que la ou les parties les plus importantes d'un document soient systématiquement transmises au début de la communication.

6.6.2.1. Techniques d'accès à l'information au sein d'un SIS

Un SIS est constitué d'un ensemble de nœuds dont le nombre varie perpétuellement. Chacun de ces nœuds est porteur d'un ensemble d'informations, qu'il est susceptible de rendre accessible dans le cadre du SIS auquel il participe. L'ensemble de ces informations "publics", distribuées sur l'ensemble des nœuds participant à un instant t à un SIS donné, est appelé l'espace visible du SIS. Cette espace doit refléter dynamiquement l'ensemble des informations potentiellement accessibles dans le SIS. Nous étudions actuellement les mécanismes systèmes nécessaires à la construction de l'espace visible. La difficulté réside dans la nature totalement décentralisée d'un SIS. En effet, chaque nœud est autonome. Il est donc impossible de faire jouer à l'un d'entre eux le rôle d'un annuaire centralisé (à la façon par exemple dont le lookup service centralise les services au sein du système JINI). Dans un premier temps, un protocole "minimal" (ou bootstrap) a été proposé, afin de permettre à deux nœuds qui viennent de se détecter mutuellement (dans un même voisinage physique), de s'identifier et de démarrer une connexion. Ce protocole nous permet d'"exporter" vers chaque nœud participant au SIS une liste d'identifiants représentant les informations publics disponibles au sein de l'espace visible. Nous cherchons maintenant à étendre cette solution en prenant en compte l'aspect dynamique du SIS, notamment en collaborant étroitement avec le niveau prédictif.

Le deuxième axe que nous étudions concerne les modes d'interactions au sein de l'espace visible d'un SIS. Il s'agit de définir de quelle manière les nœuds d'un SIS peuvent accéder aux informations de l'espace visible associé. Ce problème nous renvoie aux techniques d'accès à l'information utilisées par l'ubiquité numérique. Le principal objectif de l'ubiquité numérique est de réaliser une intégration transparente des environnements physique et numérique. Et cette intégration doit permettre aux utilisateurs d'interagir le plus naturellement possible avec les calculateurs qui les environnent. Dans le cadre spécifique des SIS, si on pousse ce raisonnement à terme, les interactions doivent avoir lieu sans que l'utilisateur n'ait conscience d'accéder aux informations des nœuds se trouvant dans le même voisinage physique. Nous avons proposé une solution appelé le Web de proximité, qui permet à des utilisateurs d'échanger automatiquement des informations en fonction de leur centres d'intérêts respectifs. Trois problèmes ont été considérés dans le cadre de cette étude :

- la définition de profils utilisateur : afin d'émettre automatiquement des requêtes "pertinentes" vers un nœud distant, les centres d'intérêt d'un utilisateur doivent être découverts. Dans ce but, le système doit construire localement et automatiquement un profil de l'utilisateur, c'est à dire les sujets sur lesquels ce dernier souhaite récupérer de l'information au sein d'un SIS.
- l'indexation des informations publics : chaque nœud doit automatiquement organiser ces documents publics, afin de permettre des échanges efficaces durant les périodes (souvent brèves) de communication au sein d'un SIS. Une solution d'indexation thématique a été retenue. Ainsi, à chaque objet, il est proposé d'adjoindre 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 nœuds distants au sein d'un SIS.
- la découverte spontanée des informations pertinentes : pendant la rencontre de deux nœuds au sein d'un SIS, il est nécessaire de sélectionner les documents en fonction des profils utilisateurs.

C'est sur ce plan que la politique d'indexation de l'information prend tout son sens. 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. Le protocole de découverte proposé permet à un nœud de découvrir, parmi les autres entités appartenant au SIS, toutes celles qui correspondent "le mieux" à son profil.

Afin de proposer une implémentation complète de ce protocole, une architecture composée de quatre modules a été définie :

- **Storage** : ce module se charge d'une part du stockage des documents publics (accessible via l'espace visible) au niveau de chaque nœud, ainsi que de la sauvegarde des informations obtenues sur une entité distante du SIS. Dans la mesure où les ressources d'une entité participant à un SIS (par exemple un PDA) sont limitées, ce module a été conçu comme un cache. Ainsi, seules les informations les plus récemment accédées au sein d'un SIS sont conservées localement.
- **Database manager** : ce module implante le mécanisme de data mining qui réalise le mécanisme d'indexation des informations au niveau de chaque nœud du SIS.
- **Information Discovery** : ce module réalise la découverte des éléments communs entre les profils utilisateurs du nœud local et du nœud distant.
- **Communication** : ce dernier module réalise l'interface entre les mécanismes de découverte automatique, et l'interface de communication sans fil. Dans nos travaux futurs, il devra collaborer avec les mécanismes prédictifs et d'adaptation des données décrits dans la partie précédente.

6.7. Systèmes d'informations spatiaux

Participants : *Michel Banâtre, Paul Couderc, Julien Pauty.*

L'informatique mobile connaît depuis le début des années 1990 un essor important. Deux aspects principaux contribuent à cet essor. D'une part le développement des calculateurs mobiles : ordinateurs portables, assistants numériques, téléphones mobiles, outils de navigation GPS sont autant de formes des calculateurs qui peuvent être embarqués par un utilisateur aujourd'hui. D'autre part, les infrastructures de communication pour terminaux mobiles sont également en plein essor, en particulier les réseaux téléphoniques cellulaires.

Parallèlement à ces développements, le Web s'est imposé comme le fédérateur des systèmes d'informations grande échelle, en offrant une interface universelle. Aujourd'hui, de plus en plus d'entités réelles (personnes, objets, administrations, entreprises, produits, lieux...) ont une existence sur le Web, c'est-à-dire une représentation sous la forme d'une page ou d'un site.

Technologiquement, l'informatique mobile offre aujourd'hui le potentiel pour permettre un accès omniprésent à des systèmes d'informations grande échelle comme le Web. Cependant, offrir cet accès n'est pas suffisant.

À cause de sa structure anarchique et dynamique, la recherche d'informations sur le Web est complexe, et peut être longue et fastidieuse. Des moteurs de recherche indexant une partie du Web facilitent ces recherches, mais dans le cadre d'une utilisation mobile, ils sont inadaptés : l'utilisateur doit pouvoir accéder très rapidement aux informations du Web en rapport avec la situation où il se trouve.

Ces travaux visent à proposer une intégration du système d'information, par exemple le Web, à l'environnement physique, permettant ainsi à l'utilisateur d'accéder très rapidement aux informations du Web en rapport avec sa situation (réelle). Notre système repose sur la notion de contexte spatial, qui représente un ensemble de nœuds du système d'information proches d'un point de référence selon une dimension donnée. En particulier, la dimension spatiale permet de déterminer les nœuds du système d'information qui sont connexes dans l'espace physique à un point donné. Dans notre modèle, un système Web classique ne permet à l'utilisateur que de naviguer selon la dimension « thématique » (correspondant aux hyperliens de la page courante), et selon la dimension « historique » (correspondant à la suite des pages visitées). Grâce à ce système, l'utilisateur peut naviguer dans le système d'information implicitement, en se déplaçant physiquement.

Mais un tel concept n'a d'intérêt que s'il est possible de construire simplement des systèmes d'informations reflétant cette dimension spatiale. En effet, l'espace physique réel étant extrêmement dynamique (à cause des nombreuses entités mobiles qu'il comporte), le problème de la représentation du contexte spatial des nœuds du système d'information est difficile. En particulier, les solutions centralisées répercutant dans une base de données géographiques les positions des entités mobiles du système à chaque déplacement, trouvent leurs limites.

Notre approche à ce problème utilise la notion de contexte spatial : il s'agit d'un ensemble d'informations accessibles dans une zone limitée de l'espace, diffusées directement par les entités physiques (ou un représentant « électronique » embarqué sur l'entité) auxquelles ces informations correspondent. Pour faire un parallèle avec les architectures classiques, nous pouvons dire que l'espace physique (volumique) constitue la mémoire globale de notre architecture. À un instant donné un processus embarqué sur un ordinateur mobile ne peut adresser qu'un ou sous ensemble de ces informations (ou contexte), celles pour lesquelles il a un lien d'accès. Ce contexte varie en fonction du déplacement physique du processus ou d'une réorganisation de la mémoire globale par un déplacement explicite des « contenants » de l'information.

Nous avons défini une architecture spatiale qui repose sur des technologies de communication sans fil courte portée et de calculateurs mobile (PDA) qui permettent d'implanter simplement la notion de contexte spatial. Dans la suite de cette section nous revenons brièvement sur ces différents points.

6.7.1. Structure d'un système d'information spatiale

6.7.1.1. Mécanismes d'adressage à base de tuple

Un système d'information spatiale est composé d'un ensemble d'entités, chacune d'elles remplit un volume de l'espace physique (sphère, ...). À chacune de ces entités est associée un ensemble d'informations. Chaque point de l'espace physique est alors inclus dans un ensemble (éventuellement vide) de volumes associés à différentes entités.

Si nous considérons qu'à un instant donné, un processus, (ou calcul), est associé à un point de l'espace physique, le contexte (i.e. l'ensemble des entités), accessible par ce processus est caractérisé par l'ensemble des entités pour lesquelles ce point appartient à l'ensemble des volumes associés. Ce contexte évolue en fonction de la mobilité du calcul ou des entités considérées. Comme plusieurs entités peuvent faire partie d'un processus, il est nécessaire de disposer d'un mécanisme de sélection des informations d'une entité suivant leur type. La solution retenue à ce problème est dérivée des notions de *tuple* et d'espace de *tuple* introduits dans LINDA.

Toute information associée à une entité est caractérisée par un *tuple* qui est une séquence d'information typée. Ainsi <10, « Paul », 3.14> définit un *tuple* dont le premier élément est un entier, le second est la chaîne « Paul » et le troisième, la valeur réelle 3.14. L'adressage des informations se fait alors de façon anonyme via du « *pattern matching* » entre un modèle de *tuple* et les éléments de l'espace des *tuples*.

6.7.1.2. Accès à l'information spatiale

Les espaces d'adressage étant définis, notre objectif est de caractériser les opérations d'accès aux informations contenues (tuples) dans ces espaces. Pour ce faire, nous proposons quatre opérations de base, qui bien qu'inspirées de LINDA, en diffèrent. D'une part, elles prennent en compte la mobilité physique des entités et des processus, et d'autre part le fait que les entités sont relatives à des objets physiques.

Les opérations que nous avons définies sont au nombre de quatre :

- `out(tuple)` publie un nouveau tuple dans l'espace des tuples de l'entité considérée. La portée de ce tuple dépend du volume physique occupé par l'entité.
- `rd(pattern)` retourne un tuple correspondant au *pattern* passé en paramètre, sans le détruire. Si aucun tuple ne correspond au *pattern*, le processus se bloque.
- `capture(pattern)` retourne l'ensemble des tuples correspondant au *pattern*, sans les détruire. Si aucun tuple ne correspond au *pattern*, le processus se bloque.
- `drop(tuple)` permet à l'entité qui a publié un tuple de le retirer de l'espace des tuples correspondant.

6.7.2. Architecture

L'architecture de SPREAD s'articule autour de trois composants principaux :

- Le gestionnaire de l'espace de tuples qui gère le stockage des tuples et l'accès à ces tuples au travers des requêtes émises par les applications.

- Un serveur de tuples qui traite des accès aux espaces de tuples distants au moyen d'opérations *get/put*.
- Le système de découverte qui traite de l'évolution spontanée des espaces de tuples en fonction de la portée des systèmes de communication sous-jacents et de la mobilité des utilisateurs.

6.7.2.1. Gestion de l'espace des tuples.

Le principal problème relatif à l'implantation de l'espace des tuples est lié au choix de la stratégie de propagation des tuples aux processus qui sont bloqués sur des requêtes *rd(...)* ou *capture(pattern)*. Une première stratégie (*write-driven*) serait de demander à chaque entité de diffuser en permanence son ensemble de tuple.

À cette solution coûteuse en bande passante, nous avons retenu la stratégie (*read-driven*) pour laquelle la création d'un tuple correspond à une opération locale alors qu'une opération de lecture peut engendrer une opération distante, qui est consommatrice de bande passante. Le gestionnaire de l'espace des tuples gère une liste de requêtes de lecture « en attente ». Cette liste est utilisée par le module de « découverte » qui informe les entités voisines des lectures « en attente ».

6.7.2.2. Adressage spatial.

De façon générale, dans notre modèle un tuple est associé à un certain volume de l'espace physique. La solution que nous avons retenue pour la construction de ces volumes repose sur l'utilisation de système de communication sans fil courte portée (radio ou infra-rouge). La difficulté majeure de ces différentes techniques réside dans la difficulté de faire correspondre le volume « occupé » par le système de communication et celui de la portée de la variable. Nous étudions actuellement à la recherche de solutions logicielles qui permettent de faire coïncider au mieux ces deux volumes.

6.7.3. Applications

L'intérêt de ce modèle de programmation est démontré via les différentes applications relevant du domaine de l'ubiquité que nous avons implantées. La plupart sont relatives à la construction de systèmes de navigation spatiales comme nous l'avons décrit dans les sections précédentes. Mais depuis nous avons expérimenté nos solutions dans d'autres domaines en particulier, celui des transports, domaine où le modèle montre à la fois toute sa puissance et toute sa simplicité d'utilisation.

6.7.4. Problèmes ouverts

Nous sommes convaincus d'avoir proposé une solution originale pour la conception d'applications du domaine de l'ubiquité numérique. Cette solution s'appuie principalement sur la notion de système d'information spatiale et le modèle de programmation associé. Dans ce contexte le monde physique caractérise la mémoire à laquelle appartiennent les informations manipulées. Notre objectif serait d'approfondir une telle architecture de machine dans laquelle la mémoire est le monde physique et les processeurs, ceux embarqués via des PDA mobiles. La mémoire accessible par un tel processeur à un instant donné dépend à la fois du volume occupé par une mémoire et de la position physique des processeurs.

Bien entendu, outre ces problèmes d'adressage les problèmes classiques liés aux systèmes d'information (sécurité, disponibilité,...) sont à prendre en compte.

Un dernier problème, et peut être l'un des plus difficiles est de concevoir les applications les plus réalistes qui permettent de valider une telle architecture.

7. Contrats industriels

7.1. Contrats nationaux

7.1.1. Texas Instruments

- 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.
- Avenant Date de début : 01/10/2001, date de fin : 30/09/2003.

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.

7.1.2. Alcatel

- Numéro de la convention : 101C078,
- Intitulé : logiciels et applications pour ordinateurs de poche sans fil utilisant des communications par voisinage physique.
- Activité de recherche concernée : § 6.6.
- 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. Programme d'action intégré Picasso

- Intitulé : Utilisation de mémoires caches dans les systèmes temps-réel strict
- Période : [Janvier 2003- décembre 2003]
- Partenaires : Irida - l'Universidad Politécnic de Valencia
- Note : Projet soumis, en cours d'évaluation

L'objectif de cette collaboration est d'étudier et d'évaluer les performances de méthodes de verrouillage statique de caches pour rendre prévisibles les accès mémoires dans les systèmes temps-réel strict.

8.2. Réseaux et groupes de travail internationaux

8.2.1. CaberNet

- Numéro de la convention : 1 01 D0236
- Intitulé : IST NoE CABERNET (Network of Excellence in Distributed and Dependable Systems)
- Période : [Janvier 2001 - Décembre 2003]
- 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 (Grèce),
 - Trinity College Dublin (Irlande),
 - CNUCE-CNR, Instituto di Elaborazione dell'informazione - NCR, Politecnico di Milano, Scuola superiore S. Anna, Tecnopoli 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 des résultats

9.1. Animation de la communauté scientifique

9.1.1. Comités de programme

Les membres du projet participent de manière régulière à des comités de programme de conférences internationales ou française, ainsi qu'à des comités de lecture :

- International conference on Object-Oriented Real-Time Distributed Computing (ISORC) en 2002 et 2003 (I. Puaut),
- Ecole d'été sur les systèmes d'exploitation temps-réel en septembre 2003 (I. Puaut),
- Comité de lecture du numéro spécial "temps-réel" de la revue francophone "Technique et Science Informatiques" (I. Puaut),
- 15th Euromicro Conference on Real-Time Systems 2003 (ECRTS 2003) (I. Puaut),
- Chairman de la session "High level analysis" on 2nd workshop on worst case execution time analysis, juin 2002 (I. Puaut).

9.1.2. Autres responsabilités sur un plan national

- I. Puaut est secrétaire de l'Association ACM-SIGOPS de France (ASF), depuis octobre 2002.

9.2. Enseignement universitaire

Les chercheurs et enseignants/chercheurs du projet ACES 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*.

- Ifsic :

- Responsabilité du module SYCR (SYstèmes d'exploitation et de Calculs Répartis) en DEA d'informatique (M. Banâtre, G. Cabillic, F. Weis),
- Responsabilité du module systèmes répartis Diic 3 ARC (M. Banâtre et F. Weis),
- Intervention en DESS CCI sur le thème des systèmes d'exploitation et Java embarqué (G. Cabillic).
- Ecole des Mines de Nantes, filière FI4 :
 - Responsabilité du module de systèmes répartis à l'École des Mines de Nantes (M. Banâtre),
 - Formation en Java embarqué (G. Cabillic).
- INSA de Rennes :
 - Responsabilité du cours de systèmes d'exploitation répartis, 5ème année, option informatique (M. Banâtre),
 - Responsabilité de la 4ème année option informatique (I. Puaut),
 - Responsabilité du module optionnel "systèmes temps-réel embarqués", 5ème année, option informatique (I. Puaut), et option EEI.
- Université de Rennes I, UFR Structure et Propriétés de la Matière, DESS Domotique et Réseaux Intérieurs : formation sur les réseaux (F. Weis).
- ENSERB (Bordeaux), Conférence communication mobile et informatique ambiante, Décembre 2002 (M. Banâtre).

Nous participons également aux commissions de spécialistes section 27 de divers établissements : INSA de Rennes, Université de Bretagne Sud, Vannes (I. Puaut).

9.3. Accueil de stagiaires

Pendant l'année 2002, les membres du projet ont accueilli et encadré des stagiaires venant de différents établissements :

- Julien Pauty, Ecole Nationale d'Ingénieur de Brest (stage d'ingénieur option DEA),
- Hervé Roussain et Mathieu Minard IFSIC (stage de DEA),
- David Garnier, Ecole des Mines de Nantes (stage d'ingénieur option DEA),
- Mathieu Avila et David Hénot (mémoire d'ingénieur de l'Institut d'Informatique d'Entreprise - IIE),
- Mathieu Bécus, IFSIC, DIIC ARC (stage de fin d'études),
- Kevin Perros, INSA de Rennes (stage de fin d'études).

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. De nombreuses présentations des travaux de recherche des membres du groupe ont également été effectuées dans le cadre de différents séminaires et manifestations scientifiques (e.g. journées RNRT, journées de veille technologique IriSaTech et Iliatech, séminaire Labri-Bordeaux, LAAS-Toulouse, Ircyn-Nantes, Action spécifique CNRS SoC RTOS et multiprocesseurs, Action spécifique CNRS Méthodes et outils logiciels pour le développement de systèmes d'exploitation, Action spécifique CNRS Compilation pour systèmes embarqués, groupe de travail RIS Toulouse logiciel libre et sûreté de fonctionnement, groupe de travail Spécification Temporelle et Stochastique du pôle ARP du GDR, etc).

Michel Banâtre est expert auprès de la C.E.E. pour l'évaluation de projet dans le cadre du 5ème PCRD et pour sa participation au forum « Futur mobile telecommunications and China-EC on beyond 3G », Pékin novembre 2002.

9.5. Dépôts de brevets

- G. Cabillic, F. Parain, J.P. Lesot, M. Banâtre, J.P. Routeau, S. Majoul, G. Chauvel, D. Dinverno, S. Lasserre. « *A Real-time Energy-aware Dynamic Scheduling Algorithm for Heterogeneous Multiprocessor Architecture* ». En cours de dépôt.
- G. Cabillic, J.P. Lesot, M. Banatre, F. Parain, J.P. Routeau, G. Chauvel, D. Dinverno, S. Lasserre. « *Java Profiling System for Energy and Execution Time* ». Europe Number #02290195.3, 29th January 2002.
- M. Banâtre, P. Couderc, G. Cabillic. « *Dispositif et procédé de gestion de données entre équipements de communication en vue de l'obtention d'un service* ». Demande de dépôt du brevet Français, Octobre 2002.

10. Bibliographie

Thèses et habilitations à diriger des recherches

- [1] F. PARAIN. *Ordonnancement sous contraintes énergétiques d'applications multimédia sur une plate-forme multiprocesseur hétérogène*. thèse de doctorat, Université de Rennes I, Mai, 2002.

Articles et chapitres de livre

- [2] T. HIGUERA, V. ISSARNY, G. CABILLIC, M. BANÂTRE, F. PARAIN, J. LESOT. *Memory Management for Real-time Java :an Efficient Solution using Hardware Support*. in « Real-Time Systems Journal », Mai, 2002.

Communications à des congrès, colloques, etc.

- [3] G. CABILLIC, S. MAJOU, J. LESOT, M. BANÂTRE. *A profiling methodology for embedded Java applications*. in « 2nd Usenix Java Virtual Machine Conference », 2002.
- [4] G. CABILLIC, S. MAJOU, J. LESOT, M. BANÂTRE. *An Approach for a Dependable Java Embedded Environment*. in « 10th ACM SIGOPS European Workshop - Dependability », Septembre, 2002.
- [5] P. COUDERC, M. BANÂTRE. *Ambient computing applications : an experience with the SPREAD approach*. in « 36th Hawaii International Conference on System Sciences », 2002.
- [6] D. DECOTIGNY, I. PUAUT. *Artisst : An extensible and modular simulation tool for real-time systems*. in « 5th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2002) », Avril, 2002.
- [7] D. DECOTIGNY, I. PUAUT. *Artisst : An extensible framework for the simulation of real-time systems*. in « 10th International Conference on Real-Time Systems (RTS'02) », Mars, 2002.
- [8] I. PUAUT, D. DECOTIGNY. *Low-Complexity Algorithms for Static Cache Locking in Multitasking Hard Real-Time Systems*. in « 23rd IEEE International Real-Time Systems Symposium », Décembre, 2002.
- [9] I. PUAUT. *Cache modeling vs static cache locking for schedulability analysis in multitasking real-time systems*. in « 2nd Intl Workshop on worst-case execution time analysis, in conjunction with the 14th Euromicro Conference on Real-Time Systems », Juin, 2002.

- [10] I. PUAUT. *Real-Time Performance of Dynamic Memory Allocation Algorithms*. in « 14th Euromicro Conference on Real-Time Systems », Vienne, Autriche, Juin, 2002.
- [11] A. TROËL, M. BANÂTRE, F. WEIS. *Progressive HTML for Proximate and Automatic Interactions*. in « International Workshop on Smart Appliances and Wearable Computing (IWSAWC'02) », Vienne, Juillet, 2002.
- [12] F. WEIS. *Mise en oeuvre des SIS*. in « Assises Nationales du GDR I3, Groupe "Ubiquité et Mobilité », Nancy, Décembre, 2002.

Rapports de recherche et publications internes

- [13] A. COLIN, I. PUAUT, C. ROCHANGE, P. SAINRAT. *Calcul de majorants de pire temps d'exécution : état de l'art*. rapport technique, IRISA numéro 1461 et IRIT numéro 2002-15-R, Mai, 2002.
- [14] D. TOUZET, F. WEIS, M. BANÂTRE. *Accès à l'information en Ubiquité Numérique*. rapport technique, IRISA Numero 1460, 2002.

Bibliographie générale

- [15] 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.
- [16] R. ARNOLD, F. MUELLER, D. WHALLEY, M. HARMON. *Bounding worst-case instruction cache performance*. in « Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS94) », 1994, pages 172-181.
- [17] M. D. BENNETT, N. C. AUDSLEY. *Predictable and efficient virtual addressing for safety-critical real-time systems*. in « Proceedings of the 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands », 2001, pages 183-190.
- [18] éditeurs G. C. BUTTAZZO., *Hard real-time computing systems predictable scheduling algorithms and applications*. Kluwer Academic Publishers,, 1997.
- [19] L. R. CLAUSEN, L. P. SCHULTZ, C. CONSEL, G. MULLER. *Java bytecode compression for low-end embedded systems*. in « ACM Transactions on Programming Languages and Systems », volume 22(3) :471-489, 2000.
- [20] A. COLIN, I. PUAUT. *Worst case execution time analysis for a processor with branch prediction*. in « The Journal of Real-Time Systems », volume 18(2-3) :249-274, 2000.
- [21] M. FLINN, M. SATYANARAYANAN. *Energy-Aware Adaptation for Mobile Application*. in « Proceedings of ACM SOSP », 1999, pages 48-63.
- [22] D. GELERNTER. *Generative communication in Linda*. in « ACM Transactions on Programming Languages and Systems », volume (7)1, 1985.

- [23] M. S. JOHNSTONE, P. R. WILSON. *The memory fragmentation problem : Solved ?*. in « Proceedings of the 1st International Symposium on Memory Management », volume 34, 1998, pages 26-36.
- [24] S.-S. LIM, Y. H. BAE, G. T. JANG, B.-D. RHEE, S. L. MIN, C. Y. PARK, H. SHIN, K. PARK, S.-M. MOON, C.-S. KIM. *An accurate worst case timing analysis for RISC processors*. in « Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS94) », 1994, pages 97-108.
- [25] J. LORCH. *A complete picture of the energy consumption of a portable computer*. Masters thesis, Computer Science, University of California at Berkeley,, 1995.
- [26] K. NILSEN, H. GAO. *The real-time behavior of dynamic memory management in C++*. in « Proceedings of the 1995 Real-Time technology a Applications Symposium, pages 142-153, Chicago, Illinois », 1995.
- [27] B. NOBLE, M. SATYANARAYANAN, J. TILTON, J. FLINN, K. WALKER. *Agile application-aware adaptation for mobility*. in « Proceedings of the 16th Symposium on Operating Systems Principles », 1997.
- [28] P. PILLAI, K. G. SHIN. *Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems*. in « Proceedings of the 18th ACM Symp. on Operating Systems Principles », 2001.
- [29] P. PUSCHNER, A. BURNS. *A review of worst-case execution-time analysis*. in « The Journal of Real-Time Systems », volume 18(2-3) :115-128, Guest Editorial, 2000.
- [30] P. PUSCHNER, A. V. SCHEDL. *Computing maximum task execution times - a graph based approach*. in « Proc. of IEEE Real-Time Systems Symposium », volume 13, 1997, pages 67-91.
- [31] J. STANKOVIC, M. SPURI, M. D. NATALE, G. BUTTAZZO. *Implications of classical scheduling results for real-time systems*. in « IEEE Computer », volume 28(6) :16-25, 1995.
- [32] J. STANKOVIC. *Strategic directions in real-time and embedded systems*. in « ACM Computing Surveys », volume 28(4) :751-763, 1996.
- [33] V. TIWARI, S. MALIK, A. WOLFE. *Power Analysis of Embedded Software : A First Step Towards Software Power Minimization*. in « IEEE Trans. on Very Large Scale Integration Systems », 1994, pages 437-445,.
- [34] M. WEISER, A. DEMERS, B. WELCH, S. SHENKAR. *Scheduling for reduced CPU energy*. in « Proceedings of Operating System Design and Implementation (OSDI), Monterey, CA », 1994.
- [35] M. WEISER. *Some Computer Science Issues in Ubiquitous Computing*. in « Communication of the ACM », volume (7)36, 1993, pages 75-83.
- [36] N. ZHANG, A. BURNS, M. NICHOLSON. *Pipelined processors and worst case execution times*. in « The Journal of Real-Time Systems », volume 5(4) :319-343, 1993.