Intégration des fautes dans un modèle de programmation pour réseaux mobiles

Corentin Mehat, Olivier Marin, Fréderic Peschanski

MajecSTIC 2009
Avignon, France, du 16 au 18 novembre 2009

Intégration des fautes dans un modèle de programmation pour réseaux mobiles

Corentin Méhat, Olivier Marin, Fréderic Peschanski

Laboratoire d'Informatique de Paris 6 / CNRS
Université Pierre et Marie Curie
4 Place Jussieu
F-75252 Paris Cedex 05
prenom.nom@lip6.fr

Résumé
N ous proposons un langage de spécification dont le support d'exécution sont des périphériques mobiles utilisant un environnement pervasif à base de réseaux mobiles ad-hoc. Nous décrivons les problématiques de communication de groupe et de détection de fautes pour ce contexte où les canaux de communication ne peuvent pas être supposés fiables. Cette approche permet de traiter la mobilité des périphériques comme une question de tolérance aux fautes.

Abstract
W e propose a specification language for programs running on mobile devices using pervasive communications. We describe the specific group communication and fault detection challenges for this particular context where communication channels cannot be considered as being reliable. This approach allow us to consider device mobility as a fault tolerance problem.

Mots-clés : mobilité, tolérance aux fautes, plateforme, intergiciel

Keywords: mobility, fault tolerance, plateform, middleware

1. Introduction

Il semble juste d'affirmer que la conception de systèmes répartis en environnements mobiles ad-hoc demeure une tâche difficile et semée d'embûches. Les raisons de cette complexité s'expliquent principalement par les caractéristiques systèmes sous-jacentes. Aux difficultés classiques – mais toujours bien réelles– de concurrence (et non-déterminisme associé), et de répartition (asynchronisme, potentialité de pannes), s'ajoutent de nouvelles problématiques plus spécifiques à la mobilité.
Le contexte mobile impose en effet des contraintes très strictes ; on notera en particulier l'infrastructure de communication très volatile avec les connexions et déconnexions intempestives dues aux déplacements incessants des périphériques, une problématique que nous rapprochons des phénomènes de pannes. Ces difficultés font apparaître comme nécessaire la modélisation des aspects les plus complexes des applications. En découlent des possibilités de vérification et de simulation avant le déploiement effectif des applications. A un niveau intermédiaire des abstractions expressives, de haut niveau et à la sémantique claire, doivent être proposées pour permettre au concepteur de se concentrer sur les aspects applicatifs tout en permettant la prise en compte des contraintes inhérentes aux environnements mobiles. Le lien entre ces abstractions de haut-niveau, nécessaires à la phase d'implémentation, et les couches basses du système est le rôle de la couche intergicielle où se retrouvent en pratique une bonne partie des problèmes fondamentaux.
Les approches de modélisation pure existent [17] , de même que les boîtes à outils [16] ou langages de programmation dédiés [8]. Mais si de nombreuses approches intègrent des concepts liés à la mobilité, la plupart adoptent une vue relativement classique de mobilité purement logicielle (migration) [3, 14, 19] qui ne concerne pas notre contexte (arrêt/reprise de processus en cours d'exécution, copies mémoires) . Même s'il y a des problématiques communes, par exemple les

besoins de liaisons dynamiques [16], la prise en compte de la mobilité des périphériques est relativement orthogonale – et selon nous plus fondamentalement liée à des problèmes de connectivité réduite [22] : intermitence des connexions aux réseaux fixes, connexions opportunistes à d'autres périphériques mobiles.
Notre travail se positionne spécifiquement sur la problématique de périphériques mobiles, et innove principalement par la proposition d'un intergiciel offrant un cadre commun aux activités de modélisation, d'implémentation et de déploiement.
Cet article présente les voies de recherche que nous nous proposons d'explorer, ainsi que les esquisses de solutions que nous avons identifiées. Il est construit de la manière suivante. La section 2 présente le modèle d'interactions à un niveau principalement informel ; une présentation beaucoup plus complète et formelle est disponible dans [21].
La section 3 détaille les apports que nous nous proposons d'intégrer pour mieux modéliser le comportement d'un environnement mobile.Pour finir, une analyse évaluative des travaux connexes est présentée en section 4 avant de clore et de souligner quelques perspectives supplémentaires relatives à notre travail.

2. Les espaces d'interaction

Considérant notre problématique de mobilité, le modèle de spécification formelle que constitue les *espaces d'interaction* [21] nous est apparu intéressant. La terminologie d'*agents mobiles* qui y est décrite correspond aux périphériques mobiles qui nous intéressent. Après une justification de ce choix, nous décrirons brièvement les problématiques de connectivité qui nous intéressent particulièrement dans les espaces d'interaction puis décrirons brièvement en quoi consiste l'ensemble de ce modèle.

Ce modèle initialement conçu pour les *agents mobiles* intégre des aspects de connectivité dynamique inspirés du $\pi-$calcul [20]. Il propose également une métaphore géométrique des systèmes répartis asynchrones, et supporte un modèle de communication basé sur des canaux multi-points pouvant être créés ou détruits dynamiquement, et dont l'identité peut être communiquée entre processus. Ces caractéristiques permettent de modéliser et d'implémenter des protocoles de communication dont l'architecture évolue dynamiquement - un premier pas fondamental pour le support des infrastructures pervasives.
Nous proposons d'intégrer aux espaces d'interaction un modèle de faute permettant entre autres de prendre en compte les phénomènes de déconnexions non contrôlées qui sont généralement liés aux déplacements des périphériques mobiles. Le modèle présuppose à l'heure actuelle une vision relativement idéalisée de la répartition, et doit donc être rendu plus réaliste par l'intégration des phénomènes de pannes. En présence de modifications imprévues du contexte, notamment en cas de défaillance, on cherche donc à offrir la possibilité de spécifier des adaptations de l'application afin de garantir la continuité de service. Notre problématique porte tant sur les aspects formels que sur les aspects plus pratiques liés à son implémentation. En intégrant à l'intergiciel des mécanismes de détection de défaillances appropriés, nous voulons offrir un environnement de conception qui permettra de modéliser puis de déployer directement des algorithmes répartis dans un contexte mobile.
Nous proposons en particulier de relâcher une contrainte du modèle actuel concernant la cardinalité des communications. Les canaux sont à l'heure actuelle soit *unicast* (point-à-point) ou *multicast* (multi-point $1 \rightarrow N$). Nous proposons un support des canaux $N \rightarrow M$ que nous nommons *polycast*. Cet aspect est également discuté dans l'article en raison de son impact important sur le modèle de panne.

2.1. Modes de communication

Le modèle des *espaces d'interaction* tel que présenté dans [21] possède deux modèles de canaux, nommés *unipoint* (1 écrivain vers 1 lecteur) et *multipoint* (1 écrivain vers N lecteurs). Le multipoint ne correspond cependant pas au modèle multicast classique puisque d'une part il est $1 \; vers \; N$, et d'autre part il n'y est pas question de routage. Les problématiques qui nous préoccupent portent uniquement sur la gestion des canaux en termes de placement des données (et leur diffusion éventuelle) et de maintenance des informations sur les différents périphériques enregistrés sur un

canal en lecture ou en écriture.
Ceci correspond par exemple au canal C_3 sur les figures 1 et 2, où seul Q est enregistré en tant qu'écrivain sur le canal, et où R et S sont en lecture. Au niveau de l'implémentation, la gestion du canal est alors simplifiée : puisqu'un unique site possède un rôle d'écrivain, la gestion du canal lui revient entièrement.
Dans un contexte réellement pervasif et mobile, il n'est plus raisonnable de centraliser ainsi cette gestion. Ceci permet donc de lever la contrainte d'unicité de l'écrivain. Le nouveau modèle de canal est appellé *polycast* (canal C_1 sur la figure 2), pour le différencier des deux précédents et correspond donc à de la communication de groupe (N écrivains vers M lecteurs). La levée de ces contraintes entraîne de plus la possibilité pour un site d'être à la fois lecteur et écrivain sur un site, ce qui est par exemple représenté par l'enregistrement de Q sur la figure 2.
Notons en particulier que le modèle N écrivains vers M lecteurs ne correspond pas à un système de publication/souscription au sens de [11], puisque l'arrivée d'un message sur un canal ne peut pas être considérée comme un événement pour le lecteur. La nature asynchrone des canaux est en effet telle que le message restera sur le canal tant que l'écrivain n'aura pas effectué de lecture (consommation) sur le canal concerné.
De plus, et c'est également ce qui permet de différencier ce modèle d'un middleware orienté message, notre recherche ne porte pas sur les problématiques de routage de bout en bout puisque nous nous intéressons aux problématiques de voisinages de périphériques. Un autre point important est l'absence dans notre modèle de « base » centralisant les données, comme dans la plupart des MOM mobiles [18]. En effet, la mobilité de nos périphériques contraint fortement la persistance des données, que ce soit pour la gestion des canaux ou pour la transmission des messages.
Enfin, le modèle de canal N *vers* M proposé ici est de plus bas niveau que la notion de topic *topic* en publication/souscription, et ne porte pas sur les questions de sélection en fonction du type ou du contenu. De plus, la problématique du passage à l'échelle ne se pose pas puisque les canaux n'ont de sens que dans un périmètre limité en contexte mobile.

2.2. Principes de base

Le modèle des *espaces d'interaction* [21] a été développé pour servir de cadre à la conception de logiciels pervasifs basés sur une notion d'agent mobile. Il est fondé sur les contraintes liées à la répartition (concurrence, asynchronisme, décentralisation) dans les systèmes ouverts. En particulier, il prend en compte les spécificités fondamentales des infrastructures pervasives : sensibilité à la localisation (*location awareness*), volatilité des canaux de communication et échanges de messages asynchrones en mode point-à-point ou multi-point. Les canaux (logiques) de communication sont en particulier manipulables explicitement dans le langage créé, qui s'inspire des mécanismes du $\pi-$calcul de Milner [20].
Le modèle propose de plus une représentation graphique sous la forme d'un espace à trois dimensions (exemple figure 1 et 2) :
– dimension des agents (ou périphériques mobiles),
– dimension des identités de canaux
– dimension dite des actes représentant les états des canaux
L'intersection d'un agent et d'un canal encode la capacité pour l'agent de lire et/ou d'écrire sur le canal, et sur ce damier se placent les messages (ou *actes*) reçus, sous la forme de cubes. La figure 1 montre l'abstraction sous la forme d'un espace d'interaction. P, Q, R, S sont quatre périphériques mobiles pouvant interagir par l'intermédiaire des canaux $C_{1..4}$. Nous pouvons constater que l'agent R peut lire sur C_1 et C_3 (capacités de lecture en couleur claire), et l'agent P peut uniquement écrire sur C_1 (capacité d'écriture en noir).
La prise en compte des modifications dynamiques des capacités de lecture et/ou d'écriture sur les canaux permet de modéliser les phénomènes de connexions/déconnexions. La dynamique associée peut être précisément perçue en comparant l'évolution de la structure de la communication entre les figures 1 et 2. Dans le second état un nouveau périphérique T entre dans l'environnement et se connecte en lecture sur C_1.
En plus de leur potentiel concernant la modélisation et les vérifications formelles (cf. sémantique opérationnelle dans [21]), les *espaces d'interaction* proposent un langage réduit pouvant être implémenté efficacement et de façon légère. Le but est de réduire la distance entre les problématiques

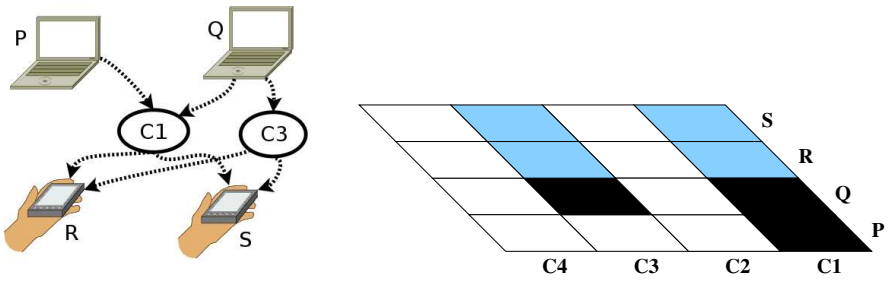

FIG. 1 – État initial : rôles des localisations sont connus.

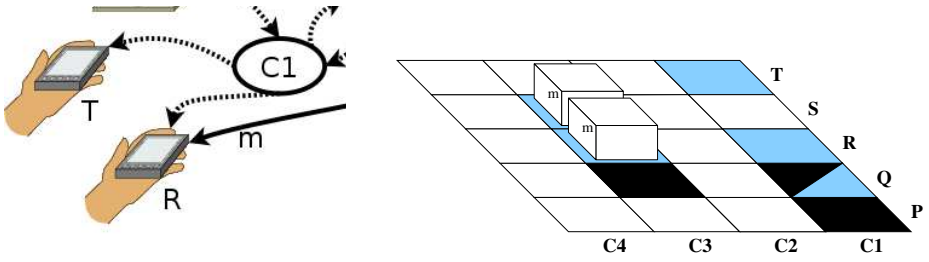

FIG. 2 – Évolution : Q émet sur C_3, T arrive et écoute C_1, Q écoute C_1, S n'écoute plus sur C_1.

de modélisation/vérification et d'implémentation/exécution. Ainsi, dans une démarche descendante l'idée est de pouvoir enrichir les spécifications abstraites pour obtenir des programmes réellement exécutables. De façon complémentaire, dans une démarche ascendante, on peut extraire (par exemple par équivalence faible [20]) des propriétés vérifiables sur des programmes existants.

3. Intégration de la tolérance aux fautes dans le modèle

Nous présenterons un modèle de fautes associé à notre modèle pour les réseaux mobiles avant de décrire un mécanisme permettant d'assurer cette détection dans les *espaces d'interaction* et enfin conclure par les ajouts qui sont alors nécessaire afin de spécifier cette détection.

3.1. Modèle de pannes

Dans les *espaces d'interaction* la notion de canal est centrale puisque c'est en fonction de cette dernière que peuvent être définies toutes les interactions d'un périphérique donné avec les autres périphériques du système. Contrairement à de nombreuses modélisations des fautes ([5,7] par exemple) nous ne supposerons donc pas les communications comme étant fiables et la détection s'effectuera sur les erreurs produites par la défaillances des canaux. Ceci semble d'autant plus logique lorsqu'on se place dans le cadre de la mobilité des périphériques puisque ce sont alors les liens entre entités logicielles qui sont les plus susceptibles d'être brisés (connexions et déconnexions intempestives).
Notre modèle de fautes se focalise sur les canaux. Nous définissons dans ce sens quatre classes de fautes que nous définissons ainsi :

Faute de lecture : Faute affectant la lecture sur un canal donné.

Faute d'écriture : Faute affectant l'écriture sur un canal donné.

Faute du canal : Faute affectant toute les possibilités d'interactions (lecture, écriture, changement de rôle) avec un canal donné.

Faute byzantine : Toute autre faute n'entrant pas dans les classes précédentes.

Notons que les fautes de canal peuvent-être considérées comme des pannes permanentes puisqu'aucune interaction avec le canal n'aura de sens par la suite. De même, les pannes de lecture

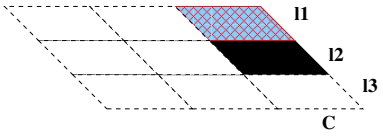

FIG. 3 – Détection 1/2 : Marquage par un *Flag* de la présence d'une faute.

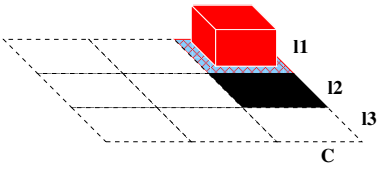

FIG. 4 – Détection 2/2 : Émission d'un acte de faute.

et d'écriture sont un raffinement de panne transitoire puisqu'elles sont susceptibles d'affecter un site particulier, sans mettre en péril la viabilité du canal sur lequel elles sont détectées.
Il n'y a pas dans les *espaces d'interaction* de mécanisme pour la détection de fautes. Nous allons donc maintenant présenter le mécanisme proposé pour permettre la spécification du d'un système comportement lorsque des fautes sont détectées.

3.2. Mécanisme de détection

Le modèle des *espaces d'interaction* considère pour l'instant un environnement « parfait ». Notre contexte de mobilité des périphériques nous a ainsi amené à définir un modèle de fautes. Afin de pouvoir utiliser ces fautes dans la phase de spécification des agents, il est nécessaire d'ajouter au langage un mécanisme permettant de signaler ces fautes. L'expressivité (au sens de [7]) des détecteurs de fautes dépendra des heuristiques de détection qui seront implémentées en utilisant ces mécanismes. Nous munissons ainsi chaque agent d'un détecteur de défaillances pour chacun des canaux qu'il considère. Cela autorise en particulier l'utilisation de politiques de détection appropriées sur chaque canal.
En présence d'une faute à porter à la connaissance d'un périphérique mobile, deux modifications de l'état du périphérique sont effectuées par le détecteur qui lui est adjoint :
– Notification immédiate de la présence d'une faute par levée d'un drapeau ou *Flag* pour chaque type de faute (figure 3).
– Ajout aux actes reçus sur ce canal d'un acte correspondant à la détection de la faute (figure 4), appelé *acte de faute*.

Les deux étapes 3 et 4 sont donc atomiques sur le périphérique auquel le détecteur de faute a été adjoint. Lorsque plusieurs fautes sont détectées sur un même canal, le *Flag* n'est pas relâché tant que tous les « actes de fautes » n'ont pas été traîtés. En particulier, le traîtement de l'*acte de faute* annule le flag qui lui était associé. La réciproque n'est pas vraie puisque traîter le *Flag* levé pour une faute ne supprime pas *acte de faute* placé dans la file des actes reçus sur le périphérique mobile.
Au niveau de la représentation graphique, la levée du *Flag* est représentée par l'application d'un *pattern* sur le damier coloré en fonction du rôle de chacune des localisations vis à vis de chacun des canaux.

Avec ce double mécanisme, il est ainsi possible de spécifier un agent pour qu'il réagisse à la détection d'une faute de manière immédiate ou bien de manière retardée. Ceci est lié à l'implémentation des *espaces d'interaction* : en fonction des politiques de répartition choisies un tampon local peut exister pour des messages, ou *actes*, déjà transmis. On peut alors spécifier un comportement dès la détection de la faute (par l'observation du *flag*) pour, par exemple, ne plus écrire de message sur un canal défaillant.
En revanche, l'observation des *actes de faute* permet de retarder la détection de la faute à un stade d'exécution ultérieur. On masque ainsi la faute de manière temporaire, sans en perdre la trace. Il est par exemple ainsi possible de conserver un comportement normal de l'agent avant la réception de l'*acte de faute*, et de cesser de lire sur le canal défaillant à partir de sa réception. Le tampon local pourra ainsi être vidé avant de cesser de lire sur le canal, permettant éventuellement de traîter des messages réceptionnés physiquement avant l'occurence de la faute détectée.

Ce double mécanisme nous permet donc de spécifier le comportement d'agents lorsque des fautes

sont détectées. Nous traitons dans le cadre de cet article de comment assurer la détection de fautes au sein des *espaces d'interaction*. Notons en particulier que le respect des propriétés de tolérance aux faute incombe ainsi à la spécification des mécanismes de recouvrement que mettent en place les agents lors de la détection d'une faute. En fonction de l'expressivité des détecteurs mis en place, on peut envisager de permettre le consensus [6]. Mais selon [15], la détection permet uniquement de garantir les propriétés de sureté pour le système. La vivacité du système dépendra quand à elle des vérifications que l'on pourra effectuer sur la spécification des agents.

3.3. Modification du langage

Au niveau de la spécification des agents, l'ajout de la détection de fautes offre la possibilité de concevoir des agents ayant un traitement particulier qui sera déclenché lors de la détection d'une faute. En terme de langage cela correspond par exemple à une notion d'exception comme par exemple en Java, et le traitement déclenché par la faute sera alors équivalent à celui d'un `catch(exception){/* traitement */}` en Java. La levée de l'exception correspondra dans notre cas à la détection d'une faute, que ce soit la levée d'un `flag` ou la consommation d'un acte de faute.

Ainsi, pour ajouter à chaque agent la possiblité de réagir à la présence d'une faute le langage est enrichi d'une primitive permettant de spécifier le comportement en cas d'erreur.

|C.catch(type.failure).P : lorsque l'erreur de type `failure` est détectée sur le canal C, exécuter P. `type` correspondant soit à `Flag`, soit à `Act`.

En particulier, l'agent spécifié par P sera exécuté lorsque l'agent tentera d'effectuer un pas d'exécution qui n'est plus possible : *dock* en présence d'une erreur d'écriture, *link* en présence d'une erreur de lecture, *read* en l'absence d'acte sur le canal et en présence d'une erreur de lecture (il n'y aura donc pas d'attente sur le read).

4. Travaux connexes

Le modèle que nous proposons étend les primitives de mobilité du π-calcul [20]. De ce point de vue, les espaces d'interactions peuvent être vus comme une implémentation répartie et asynchrone de la mobilité des canaux, en rendant explicite les capacités de lecture et d'écriture. Une différence importante avec la plupart des variantes asynchrones du $\pi-$calcul [2] concerne le caractère explicite de l'état des canaux de communication. Ceci autorise, par exemple, la possibilité de raisonner sur la causalité locale ou encore les propriétés d'ordre (FIFO on non). Dans [21] nous introduisons un opérateur permettant d'élaborer une sémantique compositionnelle. Ceci distingue donc notre approche du modèle des *acteurs* [1]. Les configurations d'acteurs sont proches des *espaces d'interaction* mais la manipulation explicite des identités d'acteur est remplacée ici par une communication plus implicite à travers les identités de canaux. L'intérêt de ce changement de perspective est double. D'un point de vue pratique, la gestion des ressources liées à la communication s'en trouve simplifiée (exemple : récupération automatique) et d'un point de plus théorique, c'est une des clés pour l'analyse compositionelle des systèmes.

Les interactions autres que point-à-point on été largement étudiées, à un niveau formel avec les algèbres de processus basée sur le *broadcast* [10]. La différence principal concerne le caractère dynamique et constructif de la cardinalité dans les *espaces d'interaction* : cela s'apparente à de la communication de groupe (dynamique) mais liée à la notion de canal. A notre connaissance un modèle à N écrivains et M lecteurs (dénommé *polycast* dans notre modèle) reste nouveau dans le domaine, et nous pensons que l'expressivité du langage s'en trouve augmentée. Il est important de distinguer le *polycast* des intergiciels orientés messages (MOM) [9, 18] et les modèles à base de publication/souscription (*publish/subsribe*) [11]. La première différence concerne l'échelle du modèle : N et M sont relativement petits dans notre cas, bornés par le nombre de périphériques pouvant se trouver simultanément dans une zone géographique limitée. Ainsi il s'agit d'une approche de plus bas niveau et nous n'introduisons pas de problématique de routage de bout en bout. La persistance des messages n'est pas non plus une priorité, même si le modèle reste ouvert à ce niveau. Finalement, le modèle ne propose pas de sélection des messages en fonction du topique ou du contenu. Bien sûr, les primitives proposées peuvent être utiles à l'implémentation de telles infrastructures.

Le cœur de ce travail concerne le modèle de panne proposé comme extension aux *espaces d'interaction*. Si l'on compare à Dpi et son modèle de panne [13], notre approche est clairement plus pragmatique et proche de la « réalité » des infrastructures réparties. Dans Dpi, en l'occurence, les agents ne peuvent communiquer que localement et doivent donc se déplacer dans ce but. Nous nous intéressons principalement aux périphériques mobiles, pour lesquels le modèle des *espaces d'interaction* semble plus adapté. En comparaison de [12], nous ne proposons pas de hiérarchisation des localisations. L'intérêt d'une telle hiérarchisation concerne le modèle hiérarchique de panne qui en découle. Mais les pannes de localisation ne sont *pas* au cœur de notre problématique « orientée canal ». L'introduction d'espaces d'interaction hiérarchiques nous semble cependant une piste intéressante que nous souhaitons étudier dans le futur.

5. Conclusion et perspectives

Dans cet article, nous proposons une nouvelle approche pour concevoir des systèmes répartis destinés à être déployés sur des périphériques mobiles. Notre approche s'appuie sur les *espaces d'interaction*, un puissant outil de spécification des échanges de données entre processus. Notre contribution consiste à :

– étendre le modèle de communication des *espaces d'interaction* avec un mécanisme N *vers* M pour mieux répondre aux besoins des algorithmes coopératifs,
– définir un modèle de fautes permettant de spécifier les réactions d'un système réparti vis-à-vis de comportements erratiques propres aux environnements mobiles,
– explorer des politiques de détection de fautes qu'il s'agira de tester pour en vérifier la viabilité ; nous envisageons d'approfondir les résultats de [22], qui propose une implémentation d'algorithmes de détection asynchrone avec une configuration dynamique.
– implémenter les mécanismes proposés et les intégrer à l'intergiciel léger réparti associé aux *espaces d'interaction* afin de permettre le déploiement des systèmes directement à partir de leur spécification.

Parallèlement à cet axe de recherche, un autre axe complémentaire est en train de se développer : celui de la vérification. A partir de la spécification d'un système en contexte mobile, il est envisageable de définir les comportements que les agents/processus doivent mettre en oeuvre pour permettre la vérification de propriétés de tolérances aux fautes, telles que celles définies dans [4]. Les mécanismes proposés devraient permettre une validation de ces spécifications vis-à-vis des propriétés que l'on souhaite assurer.

Bibliographie

1. Gul Agha. *Actors : a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, USA, 1986.
2. Roberto Amadio, Ilaria Castellani, et Davide Sangiorgi. On bisimulations for the asynchronous pi-calculus. In *CONCUR'96*, volume 1119 sur *LNCS*. Springer Verlag, 2001.
3. O. Angin, AT Campbell, ME Kounavis, et R.R.F. Liao. The mobiware toolkit : programmable support for adaptive mobilenetworking. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 5(4) :32–43, 1998.
4. A. Avizienis, J.C. Laprie, B. Randell, et C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1) :11–33, 2004.
5. M. Bertier, O. Marin, et P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proc. of the International Conference on Dependable Systems and Networks*, Washington, DC, USA, 2002.
6. T.D. Chandra, V. Hadzilacos, et Sam Toueg. The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4) :685–722, 1996.
7. Tushar Deepak Chandra et Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2) :225–267, 1996.
8. R. De Nicola, G.L. Ferrari, et R. Pugliese. KLAIM : a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5) :315–330, 1998.
9. Wolfgang Emmerich. Software engineering and middleware : a roadmap. In *ICSE '00 : Pro-*

ceedings of the Conference on The Future of Software Engineering, pages 117–129, New York, NY, USA, 2000. ACM Press.
10. Cristian Ene et Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Symposium : Fundamentals of Computation Theory*, volume LNCS 1684. Springer Verlag, 1999.
11. P.T.H. Eugster, P.A. Felber, R. Guerraoui, et A.M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2) :114–131, 2003.
12. Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, et Didier Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 sur *LNCS*, pages 406–421, Pisa, Italy, août 26-29 1996. Springer.
13. Adrian Francalanza et Matthew Hennessy. A theory of system behaviour in the presence of node and link failures. In *CONCUR*, pages 368–382, 2005.
14. Alfonso Fuggetta, Gian Pietro Picco, et Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5), May 1998.
15. Felix C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1) :1–26, 1999.
16. Yoad Gidron, Israel Ben-Shaul, Ophir Holder, et Yariv Aridor. Dynamic configuration of access control for mobile components in fargo. *Concurrency and Computation : Practice and Experience*, 13(1) :5–22, 2001.
17. GJ Holzmann. The model checker SPIN. *Software Engineering, IEEE Transactions on*, 23(5) :279–295, 1997.
18. Do-Guen Jung. Design of mobile mom : Message oriented middleware service for mobile computing. In *ICPP '99 : Proceedings of the 1999 International Workshops on Parallel Processing*, page 434, Washington, DC, USA, 1999. IEEE Computer Society.
19. F. Le Mouël et F. André. AeDEn : un cadre général pour une distribution adaptative et dynamique des applications en environnements mobiles. *Revue Électronique sur les Réseaux et l'Informatique Répartie (RERIR*, 11 :169–181, 2001.
20. R. Milner. *Communicating and Mobile Systems : The Pi Calculus*. Cambridge University Press, 1999.
21. Frederic Peschanski, Alexis Darrasse, Nataliya Guts, et Jeremy Bobbio. Coordinating mobile agents in interaction spaces. *Science of Computer Programming*, 66(3) :246–265, 2007.
22. P. Sens, L. Arantes, et M. Bouillaguet. Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants. *Arxiv preprint cs.DC/0701015*, 2007.