Numéro d'ordre: 4169

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Jérémie Albert**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Modèle de calcul, primitives, et applications de référence, pour le domaine des réseaux ad hoc fortement mobiles

---

Soutenue le : 13 décembre 2010

Après avis des rapporteurs :

Pascal Bouvry	Professeur
Rachid Guerraoui	Professeur

Devant la commission d'examen composée de :

Olivier Beaumont	Directeur de recherche	Président
Pascal Bouvry	Professeur	Rapporteur
Isabelle Demeure .	Professeur	Examinatrice
Serge Chaumette	Professeur	Directeur de thèse

– 2010 –

Remerciements

Les remerciements ont souvent été la première partie que je lisais lorsque je m'attaquais à la lecture d'une thèse. Seule partie informelle d'un mémoire de thèse, c'est sans aucun doute la partie qui en dit le plus sur l'auteur, permettant notamment au lecteur d'avoir un aperçu du contexte dans lequel s'est déroulé ce travail.

Pour commencer cette longue liste de remerciements, je tiens à exprimer ma reconnaissance à mes rapporteurs, Pascal Bouvry et Rachid Guerraoui d'avoir accepté cette tâche, pour leurs remarques sur le document et leurs disponibilités lors de nos échanges.

Merci aussi à Isabelle Demeure d'avoir accepté de participer à mon jury de thèse et à Olivier Beaumont d'avoir accepté, en plus d'y participer, de le présider.

Je tiens bien sûr aussi à remercier mon directeur de thèse Serge Chaumette de m'avoir proposé, il y a 3 ans, un sujet de thèse suffisament souple et adaptable pour me permettre de l'emmener là où je le souhaitais.

Je tiens aussi à remercier toutes les personnes liées à mon environnement de travail - collègues de CVT et leurs conjoint(e)s - qui ont su rendre mon quotidien si agréable. Je pense notamment à Arnaud, Lucile, Lionel, Amélie, Eve, Armand, Fabien, Alice, Rémi, Joinjoin et Julien pour les "anciens" et à Jo, Rémi, Renaud, Damien, Hugo, Mauricio, Bissyandé et Cyril pour les "moins anciens".

Merci aussi à ma famille (dont l'architecture n'est pas triviale) de m'avoir entouré et encouragé sans m'avoir jamais fait ressentir une quelconque pression. Merci donc aux quatre membres permanents de ma famille compsicoise (33) et aux deux membres par intérim' plaisirois (78) ainsi qu'aux cinq membres de ma famille colombienne (92). Un merci tout particulier à mon oncle toulousain (31) qui a su faire naître en moi une grande curiosité pour ces étranges machines qu'étaient les premiers ordinateurs personnels il y a maintenant plus de 20 ans ! Merci aussi à ma famille d'adoption blayaise et francilienne d'avoir fait le déplacement pour endurer ma soutenance de thèse.

Je tiens aussi à remercier mes amis "gerboisiens", Bobo, Boyan et Damien d'être venu assister à ma soutenance mais aussi à Buendon, Émilie, Hélène, Hélène, Julie, Kiki et Vianney pour les moments de détente que nous avons pu partager et qui me permettaient certainement inconsciemment de trier les bonnes idées que j'avais en tête des mauvaises. Merci aussi aux non gerboisiens Maëlle et Pierre, Jess et Kevin et aux Lulus.

Enfin, un énorme merci à Carole d'avoir réussi à supporter le rythme étrange qu'implique la compagnie d'un thésard, surtout en période de rédaction !

Table des matières

Introduction

Depuis le milieu des années 1970 et l'apparition de l'ordinateur personnel, les terminaux informatiques se miniaturisent de plus en plus. Cette miniaturisation les a rendus dans un premier temps déplaçables, puis, dans un second temps, mobiles. En parallèle, ils sont passés d'objets aptes à interagir uniquement avec l'Homme à des objets capables de communiquer les uns avec les autres. Nous appellerons réseau l'interconnexion de ces objets communicants, objets que nous appelons nœuds dans la suite du document. Initialement reliés par des fils, les dispositifs communicants peuvent aujourd'hui utiliser des technologies sans fil adaptées à leur mobilité grandissante. On distingue deux grandes familles de réseaux : ceux qui possèdent une infrastructure et ceux qui n'en possèdent pas.

Dans les réseaux avec infrastructure, les nœuds sont reliés entre eux par l'intermédiaire de dispositifs (souvent dédiés et) stables. Ces dispositifs peuvent arbitrer entre les nœuds, centraliser par exemple des annuaires et parfois même acheminer les messages. C'est le cas par exemple des réseaux de type IP/Ethernet dans lesquels les nœuds sont reliés à des passerelles. C'est aussi le cas des réseaux sans fil de type IP/Wi-Fi dans lesquels ils sont connectés à des points d'accès. Dans ces réseaux, la stabilité des liens de communication entre les nœuds permet la mise en place de mécanismes garantissant l'acheminement des messages et la sécurité du réseau. Par exemple, la mise en place d'une infrastructure à clef publique rendue possible par un accès permanent à un nœud stable (Autorité de Certification - CA) permet de garantir l'authentification des nœuds, la confidentialité des échanges, l'intégrité des données face à un attaquant et la non-répudiation des messages.

Les réseaux sans infrastructure sont, au contraire, caractérisés par l'absence de dispositifs dédiés et stables. Dans ce type de réseau les nœuds communiquent directement les uns avec les autres, de proche en proche. Les terminaux mobiles capables de former un tel réseau (téléphones portables, assistants personnels, tablettes, etc.) se sont popularisés et diversifiés ces dernières années. Ils intègrent de plus en plus souvent des modules de communication sans fil à portée limitée tels que des cartes Bluetooth, Wi-Fi ou encore ZigBee. Des objets autonomes comme par exemple les robots et les drones (aussi étudiés dans l'équipe Muse du LaBRI), et qui sont eux aussi équipés de modules de communication sans fil, commencent à se populariser. Tous ces équipements peuvent s'interconnecter les uns avec les autres et constituer de façon spontanée des réseaux que l'on nomme réseaux mobiles ad hoc.

Dans la littérature, les réseaux mobiles ad hoc qui se forment de manière non planifiée et imprévisible sont souvent étudiés en faisant l'hypothèse d'une composition et d'une topologie qui évoluent très peu après la mise en place initiale de la configuration. Cette stabilité

des nœuds entraîne une stabilité des liens de communication. Il est alors naturellement possible d'utiliser des mécanismes (tels que le routage) qui permettent aux algorithmes et applications conçus pour un contexte de réseau statique de fonctionner dans ce contexte ad hoc faiblement mobile. De même, la mise œuvre de mécanismes de sécurité identiques à ceux des réseaux infrastructurés est possible.

Les travaux présentés dans cette thèse sont au contraire centrés sur les réseaux ad hoc fortement mobiles. Dans ce contexte, les nœuds sont par définition mobiles et volatiles, ce qui engendre des modifications incessantes de la composition et de la topologie du réseau. En effet, les dispositifs mobiles ou nœuds qui forment un réseau ad hoc fortement mobile peuvent apparaître ou disparaître à tout instant. Les raisons de leur disparition peuvent être leur panne (comme dans tout système informatique), leur mobilité qui les éloigne des autres nœuds, leur énergie qui vient à manquer, etc. Dans un système informatique classique, la panne ou le message qui n'arrive pas à destination, font exception. Par conséquent, il est possible de mettre en place des mécanismes afin de gérer ces évènements. Dans un réseau ad hoc fortement mobile, l'arrêt ou l'isolement géographique d'un dispositif est un évènement récurrent et fréquent qui fait partie intégrante du fonctionnement de l'application. Nous pensons donc que les mécanismes de gestion de panne et de perte de message créés pour les réseaux avec infrastructure ou faiblement mobiles ne sont pas adaptés au contexte de réseau fortement mobile dans lesquels s'inscrivent nos travaux. Autrement dit, nous pensons que l'approche qui consiste à utiliser des algorithmes, des services tels que le routage, ou des applications créées pour un contexte statique dans un contexte dynamique en ajoutant des mécanismes cherchant à cacher la forte dynamique du réseau n'est pas réaliste. De plus, l'absence totale de dispositifs centralisants pouvant faire autorité et de liens de communication stables rend difficile la mise en place de mécanismes de sécurité. Pour palier ce problème, nous proposons d'associer à chaque nœud un TPM (*Trusted Platform Module*) ou *Secure Element* tel qu'une carte à puce ou une carte SIM (*Subscriber Identity Module*), composants que nous supposons inviolables. Nous faisons alors reposer la sécurité sur la fiabilité (admise) d'un de ces dispositifs associé à chaque nœud.

L'objectif de ces travaux de thèse est (i) d'étudier les hypothèses concernant les nœuds et le réseau qui sont fréquemment retenues dans la littérature afin de déterminer si elles sont en adéquation avec le contexte fortement mobile dans lequel nous nous plaçons, (ii) de proposer un modèle de calcul permettant la validation formelle d'algorithmes et d'applications, (iii) de fournir un ensemble de primitives regroupées au sein d'une bibliothèque permettant le développement d'applications pour les réseaux ad hoc fortement mobiles, et (iv) de présenter un ensemble d'applications qui montrent la validité de notre approche.

Structure du document

Le présent document est composé de 4 chapitres. Les paragraphes ci-dessous en résument le contenu.

Chapitre 1 : Définition du contexte

La plupart des études sur les réseaux mobiles ad hoc font un grand nombre d'hypothèses sur les communications entre les nœuds du réseau et sur leur mobilité. Dans cette thèse, nous listons les plus fréquentes de ces hypothèses et, en fonction de la pertinence de chacune d'elles, nous les intégrons ou non à notre contexte d'étude. Nous précisons ensuite quels sont les avantages, en dehors du réalisme de notre approche, de ne pas faire certaines de ces hypothèses. Nous détaillons enfin très finement le contexte dans lequel nous nous plaçons en définissant un domaine de recherche dérivé de l'étude des MANets que nous appelons les iMANets (highly Mobile Ad hoc Networks).

Chapitre 2 : Modèles de calcul et *middlewares* existants

Dans le chapitre 2, nous présentons les modèles de calcul créés pour les réseaux mobiles ad hoc qui ont vu le jour ces dernières années. Nous précisons pour chacun la manière dont il modélise la mobilité et les communications entre nœuds (*unicast*, *broadcast*, synchronisme, asynchronisme, etc.). Nous présentons ensuite les middlewares et les supports d'exécution conçus pour ces réseaux en détaillant le contexte précis pour lequel ils ont été créés.

Chapitre 3 : CiMAN, un modèle formel pour les réseaux ad hoc fortement mobiles

Dans le chapitre 3, nous introduisons CiMAN, une algèbre de processus pour la modélisation des réseaux ad hoc fortement mobiles. Nous présentons les différentes entités de notre modèle (processus, unités de calcul, nœuds et messages), les actions de communication qui leurs sont associées, et les opérateurs sur ces entités (par exemple la composition parallèle) qui permettent la définition d'un système. Nous définissons ensuite les relations de bisimulation qui permettent de comparer ces systèmes entre eux. Nous présentons aussi quelques exemples d'utilisation de CiMAN.

Chapitre 4 : Primitives, support d'exécution et applications de référence formellement validés

Dans le chapitre 4, nous utilisons CiMAN pour modéliser les primitives qui composent le support d'exécution que nous avons développé, sans oublier les aspects sécurité associés. Nous modélisons ensuite (i) les algorithmes de comptage que nous avons créés et (ii) une application de collecte de traces de voisinage que nous avons développée, en montrant pour chacun le respect de certaines propriétés fondamentales. Nous présentons enfin l'implémentation du support d'exécution ainsi que les applications de référence précédemment validés.

Chapitre 1

Définition du contexte

Sommaire

L'étude des réseaux mobiles ad hoc est une tâche complexe, laquelle complexité est en partie due à l'hétérogénéité des capacités de calcul et de communication des dispositifs, et à l'absence de liens de communication stables. Il est donc souvent indispensable de supposer que les nœuds qui composent un réseau mobile ad hoc et le monde dans lequel ils évoluent possèdent certaines caractéristiques afin d'aboutir à un résultat. Nous commençons ici par lister et discuter certaines des hypothèses classiquement retenues dans la littérature en examinant, pour chacune, ses conséquences sur l'étude des réseaux mobiles ad hoc.

1.1 Hypothèses fréquentes dans l'étude des MANets

Dans [KNE03, KNG+04a, KNG+04b], David Kotz et ses collègues de l'Université de Dartmouth et du MIT ont listé une partie des hypothèses fréquemment retenues dans la littérature consacrée au réseaux mobiles ad hoc. Leurs travaux mettent en lumière des suppositions qui, bien que souvent non réalistes, permettent de simplifier l'étude des MANets.

Dans cette section, nous reprenons certaines de leurs suppositions (que nous avons nommées H_1 à H_6) qui nous semblent adaptés à la description du contexte dans lequel nous nous plaçons. Nous présentons ensuite quatre nouvelles hypothèses que nous avons identifiées (H_7 à H_{10}). Nous montrons que certaines d'entre elles ont pour effet de masquer des caractéristiques importantes spécifiques des réseaux mobiles ad hoc. La plupart des suppositions relevées par Kotz et ses collègues sont encore souvent présentes dans la littérature de ces dernières années. Il est cependant important de noter que malgré tout, la levée de certaines de ces hypothèses a conduit à la création de nouveaux domaines de recherche. C'est par exemple le cas de l'absence de connectivité permanente qui a suscité des travaux spécifiques (cf. section 1.1.10 page 13).

1.1.1 H_1 : Le monde est plat

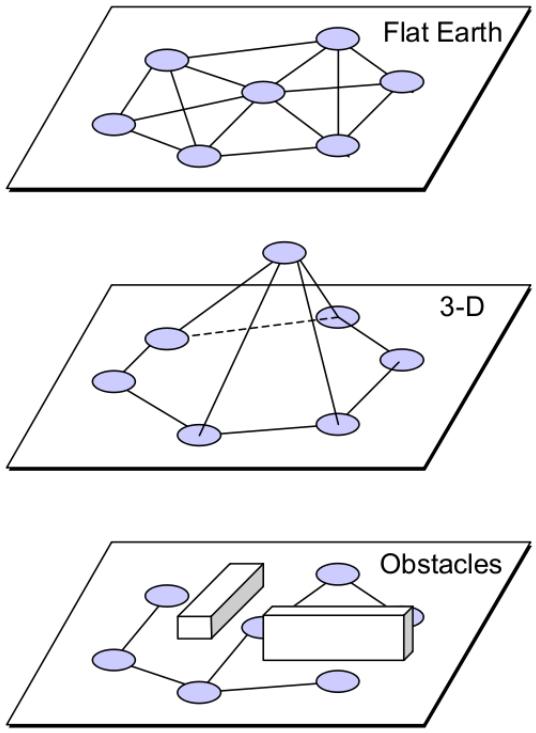


FIG. 1.1: Le monde n'est pas plat [KNE03]

Les études sur les MANets s'appuient rarement sur un espace à trois dimensions. Pourtant, même si deux nœuds se trouvent à la même latitude et longitude, ils peuvent en réalité être séparés de plusieurs dizaines (voire centaines) de mètres (deux personnes dans un bâtiment à des étages différents, deux voitures sur la route mais à un croisement sur deux niveaux ou dans un parking à plusieurs niveaux, etc.). Il en résulte que même si les nœuds du réseau étudié sont des piétons ou des voitures, l'hypothèse qui consiste à les faire évoluer dans un

espace à deux dimensions est difficilement justifiable dès lors que l'objectif est de se rapprocher du monde réel. De plus, dans un MANet, il est souvent supposé que la possibilité pour deux nœuds de communiquer dépend uniquement de la distance les séparant (cf. H_2 section 1.1.2). Or, la distance séparant deux nœuds dans un espace à trois dimensions est supérieure ou égale à leur distance calculée dans un espace à deux dimensions (pour tout $A(x,y,z)$ et $B(x',y',z')$, $AB = \sqrt{(x-x')^2+(y-y')^2+(z-z')^2} \geq \sqrt{(x-x')^2+(y-y')^2}$). Par conséquent, des liens de communication qui n'existent pourtant pas dans un espace à trois dimensions peuvent sembler présents dans la projection en dimension deux de cet espace (cf. figure 1.1.1). Les résultats de simulation ou les preuves qui s'appuient sur ces topologies non représentatives du réseau peuvent donc parfois fournir des résultats incohérents avec la réalité.

1.1.2 H_2 : Une zone de transmission radio est circulaire

Dans un MANet, les dispositifs mobiles utilisent des technologies de communication radio. La topologie du réseau dépend directement des positions des dispositifs et de la portée de leur radio. On appelle portée (radio) d'un émetteur, la distance jusqu'à laquelle les messages émis sont audibles par des dispositifs récepteurs. Il est très souvent supposé que les dispositifs récepteurs sont similaires relativement à leur capacité de réception. Or, il convient de rappeler que lors d'une communication sans fil, la réception ou non d'un message dépend aussi du dispositif récepteur. De même qu'une oreille humaine peut-être plus ou moins sensible, les récepteurs de dispositifs mobiles ne sont pas tous équivalents.

Il est aussi souvent supposé que la zone dans laquelle un nœud peut émettre des messages est un disque (ou une boule dans un environnement en 3 dimensions) dont le centre est la position de ce nœud et le rayon sa portée. Il n'existe alors pas dans cette surface (ou espace) de point à partir duquel les messages ne peuvent être reçus. La réalité est toute autre pour plusieurs raisons. Tout d'abord, les obstacles ne sont pas pris en compte. Ensuite, les antennes (même omnidirectionnelles) ne fournissent pas une couverture parfaitement régulière en forme de disque (ou de boule) mais une zone de couverture variable qui dépend de plusieurs facteurs tels que la fréquence utilisée, le bruit sur cette fréquence, la puissance d'émission, etc. Enfin, la surface (ou l'espace) dans laquelle les messages peuvent être reçus n'est pas homogène et il en résulte des positions auxquelles les messages ne sont pas audibles.

1.1.3 H_3 : Toutes les radios possèdent la même portée

Les nœuds qui composent un réseau mobile ad hoc possèdent une quantité d'énergie limitée à la capacité de leur batterie. Afin d'augmenter leur durée de fonctionnement, les nœuds doivent donc réduire leur consommation énergétique et/ou adapter leur comportement en fonction de l'énergie dont ils disposent. Pour cela, ils peuvent par exemple modifier le nombre ou la nature des services qu'il proposent, ou les protocoles qu'ils utilisent pour par exemple minimiser le nombre de messages échangés et ainsi l'énergie consommée [DPHU+08, PDR08]. Ils peuvent aussi réduire la puissance de leurs interfaces de communication sans fil [GC04]. Cette réduction provoque une diminution (i) de la consommation énergétique de l'interface de communication et (ii) de la portée de la radio

associée à cette interface. La portée d'une interface de communication radio d'un nœud peut donc évoluer au cours du temps, ce qui invalide l'hypothèse de portée identique entre toutes les radios.

1.1.4 H_4 : La relation de voisinage est symétrique

La symétrie de la relation de voisinage correspond à l'axiome suivant : *Si un message de A vers B arrive alors une réponse immédiate [à ce message] de B vers A arrive* [KNG+04a, KNG+04b]. Cette hypothèse permet de ramener l'étude des réseaux mobiles ad hoc à l'étude d'un graphe non orienté dans lequel les sommets sont mobiles. Dans ce cas, il est supposé que toute arête présente dans le graphe à un instant t, sera toujours présente à l'instant $t + \delta + 2 \times \epsilon$ avec δ égal au temps que met un sommet pour traiter le message qu'il a reçu et fournir une réponse et ϵ égal au temps de propagation du message le long d'une arête. Elle est notamment faite dans les travaux de ré-étiquetage dynamique de graphes par calculs locaux DA-GRS [CC06, Cas07, CCF09].

Cette hypothèse très utilisée est souvent justifiée par les auteurs par le fait que toutes les radios possèdent une portée circulaire de même rayon. Elle repose donc souvent sur les hypothèses H_2 et H_3. Or, comme nous venons de le voir, ces dernières ne sont pas valides dans la réalité. De plus, comme nous le présentons dans l'annexe A, cette hypothèse ne permet pas d'exploiter les liens de communication unidirectionnels qui peuvent exister entre les nœuds puisque seuls les liens de communication bidirectionnels sont considérés. Ceci peut alors nuire à l'efficacité d'une application.

1.1.5 H_5 : La force du signal perçue par un nœud est uniquement fonction de la distance entre les nœuds

Rappaport montre [Rap01] que la puissance d'un signal radio qui se propage dans le vide décroît en suivant une loi exponentielle. La décroissance de la puissance du signal est donnée par la fonction FSPL (*Free-Space Path Loss*) ci-dessous (avec λ la longueur d'onde en mètres et d la distance à l'émetteur en mètres) :

$$FSPL = (\frac{4\pi d}{\lambda})^2$$

Si l'on exprime la fonction $FSPL$ en décibels, on obtient alors (avec f la fréquence du signal en MHz et d la distance à l'émetteur en mètres) :

$$FSPL(dB) = 20\ log_{10}(d) + 20\ log_{10}(f) - 147,55$$

Un exemple d'atténuation du signal pour une fréquence de 2,432 GHz (canal 5 en Wi-Fi) est présenté figure 1.2.

Les mesures effectuées en environnement réel [KNE03] (donc pas dans le vide) montrent une faible corrélation entre la force du signal reçu et la distance séparant les dispositifs. En effet, dans la réalité, les obstacles sont autant d'objets qui empêchent la propagation ou qui réfléchissent ou réfractent les signaux. La force du signal n'est donc pas calculable grâce à la seule distance séparant les dispositifs. L'hypothèse H_5 n'est donc pas réaliste.

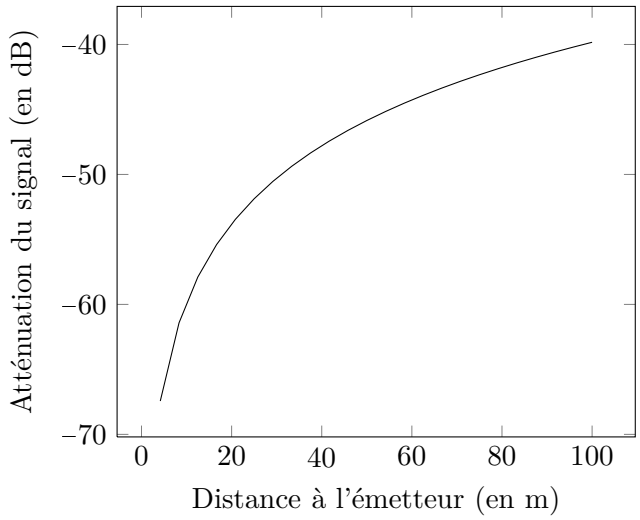


Fig. 1.2: Atténuation d'un signal radio de 2,432 GHz dans le vide

1.1.6 H_6 : Les nœuds sont capables de détecter localement la perte d'un message

On dit qu'un nœud est capable de détecter localement une perte de message lorsqu'il est capable de détecter qu'un message qu'il a émis n'a pas été reçu par son destinataire. L'hypothèse de détection locale de perte de message permet de supposer que les nœuds sont capables de gérer les messages perdus en les ré-émettant jusqu'à ce que ceux-ci soient reçus par le destinataire.

En pratique, cette détection est impossible pour au moins deux raisons. Tout d'abord, contrairement à un réseau filaire classique dans lequel les communications sont *full-duplex*, les communications sans fil sont *half-duplex* ; un nœud ne peut pas émettre et recevoir simultanément. Cette absence de simultanéité empêche les nœuds de détecter qu'une collision vient d'avoir lieu. De plus, même si l'interface de communication sans fil était *full-duplex*, le problème classique du nœud caché [RG06] (cf. figure 1.3) empêcherait une détection de collision fiable. Le problème du nœud caché apparaît lorsque qu'un nœud ne peut recevoir un message provenant d'un nœud émetteur à cause du parasitage de la bande de fréquence par un nœud tierce, et que ce parasitage n'est pas audible (et donc pas détectable) par le nœud émetteur. Il est donc impossible de détecter localement une perte de message.

1.1.7 H_7 : La mobilité des nœuds est très simplement exprimable

Pour comparer les performances des algorithmes écrits pour les réseaux mobiles ad hoc, des modèles de mobilité tels que le *Random Waypoint* [CBD02] (cf. Algorithme 1) et le *Random Walk* [CBD02] (cf. Algorithme 2) se sont imposés comme référence.

Ces deux modèles de mobilité sont parmi les plus utilisés dans la littérature. Pourtant, aucun d'eux ne correspond à une mobilité réaliste. Il est par conséquent très difficile de

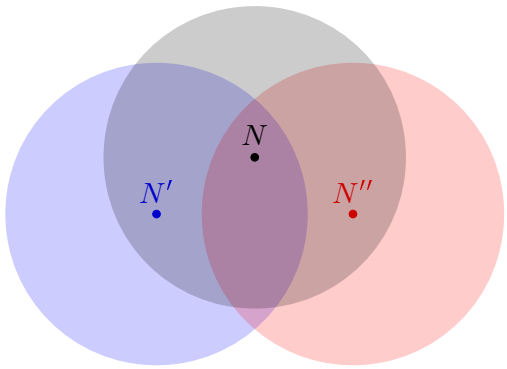


FIG. 1.3: Problème du nœud caché. N et N' peuvent communiquer mais N' ne peut détecter ni la présence de N'', ni les interférences que ce dernier peut générer.

Algorithme 1 *Random Waypoint*

```
while true do
  destination ← randomPoint()
  speed ← randomSpeed()
  moveTo(destination, speed)
end while
```

faire le lien entre les résultats obtenus grâce à ces modèles de mobilité et ceux qui seraient obtenus en environnement réel.

1.1.8 H_8 : Les nœuds possèdent un identifiant unique

Cette supposition permet aux nœuds de se distinguer les uns des autres. Ils peuvent alors, par exemple, adresser les messages qu'ils envoient. Cette supposition est présente dans un grand nombre d'algorithmes comme l'élection de *leader*, le rendez-vous ou encore dans les algorithmes de routage que nous présentons dans la section suivante.

Dans cette thèse, nous ne supposons pas que les nœuds possèdent toujours un identifiant unique. Alors que l'absence de dispositif centralisant pouvant faire autorité rend difficile la mise en place de mécanismes de sécurité dans ce type de réseau, nous montrons que contre-intuitivement, l'absence d'identifiant permet d'apporter certaines garanties originales en terme de sécurité. Nous supposons ici que les *Secure Elements* (cartes à puce, cartes SIM, etc.) associés à chaque nœud sur lesquels nous faisons reposer la sécurité sont parfaitement fiables et inviolables.

Algorithme 2 *Random Walk*

```
while true do
  direction ← randomDirection()
  moveToDirection(direction)
end while
```

Nous accordons une grande importance à un problème de sécurité classique qu'est le respect de la vie privée. En effet, pour un grand nombre d'applications et d'algorithmes pour réseaux mobiles ad hoc, chaque nœud doit avoir connaissance de son voisinage. Comme celui-ci évolue sans cesse à cause de la mobilité des nœuds, ces derniers s'annoncent régulièrement, dévoilant ainsi leur identité. Certaines expérimentations, (par exemple [Blub]) ont montré qu'il était alors possible de suivre les dispositifs mobiles grâce aux messages d'annonce qu'ils fournissent. Par exemple, les nœuds équipés d'une interface Bluetooth dite visible diffusent régulièrement leurs adresses MAC. Comme l'adresse MAC de chaque équipement Bluetooth est unique, elle permet de "suivre à la trace" le dispositif mobile et son porteur. Le raisonnement s'applique de la même façon aux interfaces Wi-Fi car elles diffusent, elles aussi, une adresse MAC unique.

L'approche qui consiste donc à s'annoncer périodiquement en dévoilant son identité ne nous semble pas raisonnable. Nous souhaitons créer des algorithmes et des applications qui fonctionnent sur les technologies telles que Bluetooth et Wi-Fi sans compromettre la sécurité des personnes qui les utilisent. Nous montrons (i) qu'il existe des algorithmes qui, contre-intuitivement, ne nécessitent pas la présence d'identité sur les nœuds (cf. ci-après) et (ii) que l'absence d'identité, au sens défini précédemment, n'empêche pas la mise en place de mécanismes de reconnaissance des nœuds (cf. chapitre 4).

L'algorithme que nous présentons dans le paragraphe suivant est issu de la réflexion que nous avons eue sur l'utilisation de la primitive de *broadcast*. Il permet d'illustrer l'approche que nous souhaitons adopter dans cette thèse même s'il peut difficilement être ramené à un cas d'utilisation concret convaincant.

Exemple d'algorithme sans identité. L'objectif de l'algorithme que nous présentons maintenant est de permettre la conservation d'une donnée de manière confidentielle dans un réseau de terminaux mobiles. Il existe toujours au moins un nœud du réseau qui possède cette donnée mais personne ne sait qui. Aucun nœud ne possède d'identité et les liens de communication ne sont pas supposés symétriques. Les nœuds communiquent par *broadcast*, et les messages qu'ils envoient ne sont pas à destination d'un nœud particulier.

La figure 1.4 illustre l'exécution de cet algorithme. Le nœud en rouge est le nœud qui possède la donnée et qui exécute l'algorithme 4. Tous les autres nœuds exécutent l'algorithme 3. Les nœuds bleus sont ceux qui reçoivent la donnée et les nœuds verts font suivre les acquittements (i.e. la valeur de hachage de la donnée de départ) qu'ils reçoivent. La distinction entre le nœud qui possède la donnée au départ et les autres nœuds est présente dans les algorithmes 3 et 4 afin de simplifier leur écriture et ainsi de faciliter leur compréhension. Dans une vraie implémentation, tous les nœuds exécuteraient un algorithme unique, fusion des deux.

Il est intéressant de remarquer que cette approche sans identité apporte des garanties en terme de sécurité. En effet, les nœuds du réseau ne peuvent pas déterminer quel(s) nœud(s) est (sont) le(s) porteur(s) de l'information initiale et aucune information concernant le (ou les) nœud(s) qui possède(nt) la donnée ne circule dans le réseau. Cet algorithme relativement simple illustre le type de propriété que nous souhaitons étudier et mettre en œuvre. Il garantit notamment la non divulgation d'information concernant les nœuds du réseau.

Algorithme 3 Algorithme du nœud qui possède initialement la donnée (nœud rouge de la figure 1.4)

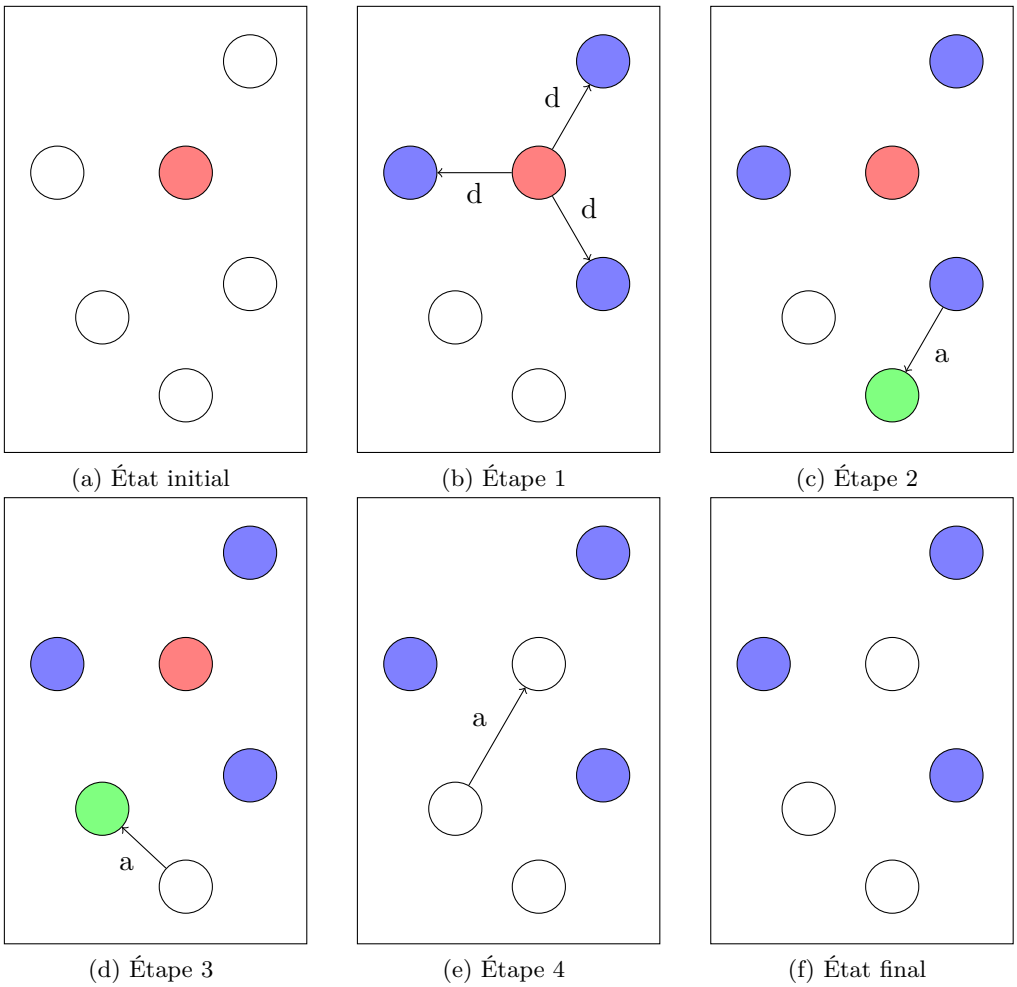
```
continue ← true
while continue do
  send(myData)
  msg ← receive()
  if msg = hash(myData) then
    delete(myData)
    continue ← false
  end if
end while
```

Algorithme 4 Algorithme de tous les autres nœuds

```
while true do
  msg ← receive()
  if msg is a data to save then
    myData ← msg
    send(hash(myData))
  end if
  if msg is a hash then
    send(msg)
  end if
end while
```

1.1.9 H_9 : Il existe des mécanismes de routage efficaces dans les MANets

Un grand nombre d'articles dans le domaine des réseaux mobiles ad hoc reposent sur l'existence d'algorithmes de routage capables d'acheminer les messages dans le réseau. AODV (Ad hoc On Demand Distance Vector) [PR99, PBRD03] et OLSR (Optimized Link State Routing) [CJ03] par exemple, sont des algorithmes dont l'objectif est de construire des routes entre les nœuds, de manière proactive pour AODV et de manière réactive pour OLSR. Ils permettent l'acheminement des messages dans un réseau mobile ad hoc sans connaissance préalable ou globale de la topologie du réseau. Les résultats obtenus en simulation par ces algorithmes montrent qu'ils sont capables de construire des routes permettant aux nœuds d'acheminer une partie des messages [BMJ+98, DCY00]. Cependant, des expérimentations en environnement réel [IHB+10a, IHB+10b] montrent qu'une forte mobilité des nœuds qui jouent le rôle de relais dans l'algorithme OLSR diminue fortement la proportion des messages qui atteignent leur destination. D'autres expérimentations montrent que AODV et OLSR échouent à acheminer une grande proportion de paquets lorsqu'il n'existe pas de chemin stable entre les nœuds ou que les liens de communication entre eux sont asymétriques [BD07].



(a) État initial (b) Étape 1 (c) Étape 2

(d) Étape 3 (e) Étape 4 (f) État final

Fig. 1.4: Illustration de l'exécution des algorithmes 3 et 4.

1.1.10 H_{10} : Le réseau est quasiment toujours connecté

Un grand nombre de travaux du domaine des réseaux mobiles ad hoc reposent sur la présence implicite quasi permanente d'un chemin entre tous les nœuds du réseau. Dans ce contexte, la mise en place d'algorithmes ou d'applications conçus initialement pour un cadre statique s'appuie sur la présence de mécanismes de routage (cf. section 1.1.8). Dans la réalité, la mobilité des nœuds peut les isoler, le réseau peut se partitionner de manière temporaire ou définitive. Il en résulte que cette hypothèse de connexité quasi permanente ne correspond pas à la nature des réseaux ad hoc fortement mobiles. Contrairement aux réseaux statiques ou faiblement mobiles dans lesquels le partitionnement du réseau ou l'isolement d'un nœud est suffisamment rare pour être traité et caché par des mécanismes adaptés, dans un réseau ad hoc fortement mobile ce comportement est le cas standard et doit être pris en compte au niveau applicatif et non plus occulté.

Le domaine de recherche qui a émergé de ce constat est celui des réseaux DTN (*Delay/-Disruptive Tolerant Network*) dits tolérants aux délais ou aux déconnexions [FC06]. Une

partie de la communauté de ce domaine s'est rassemblée au sein du groupe de recherche DTNRG [FF02] qui a émergé de la recherche sur les réseaux inter-planétaires [IPN]. De de tels réseaux sont composés de dispositifs séparés de plusieurs millions de kilomètres, engendrant de nombreuses pertes de messages et des délais de communication importants et théoriquement incompressibles car dûs à la vitesse de propagation des ondes (vitesse de la lumière).

De nombreux travaux théoriques mais aussi appliqués ont été menés dans le domaine des DTN ces dernières années, et ont donné lieu, par exemple, à la création de *middlewares* comme 7DS [SSAMSGHHGS07, AMSSHS08] (cf. section 2.2.2.1 page 26), DoD-WAN [GR04, GM07] (cf. section 2.2.2.6 page 33) et EMMA [MMH05] (cf. section 2.2.2.8 page 36).

1.2 Notre contexte de recherche : les iMANets

Dans la littérature consacrée aux réseaux mobiles ad hoc, les suppositions sur les nœuds et sur les réseaux qu'ils constituent sont à la fois nombreuses et variées. Dans ce chapitre, nous avons présenté celles que nous avons fréquemment rencontrées. Pour chacune, nous avons indiqué les raisons qui font qu'elle est peu conforme à la réalité et qui ont même parfois pour conséquence de masquer certaines caractéristiques importantes et spécifiques des MANets (par exemple l'asymétrie des liens de communication).

Nous définissons donc un iMANet (réseau ad hoc fortement mobile) comme étant un réseau mobile ad hoc dans lequel aucune des suppositions de ce chapitre n'est faite. L'ensemble de ces travaux de thèse concernent l'étude de ce type particulier de réseaux mobiles. Nous minimisons ainsi les hypothèses sous-jacentes sur le réseau et sur les nœuds qui le composent. Sauf indication contraire, nous considérons toutes les suppositions que nous avons détaillées dans ce chapitre comme invalides et elles ne font donc pas partie de nos hypothèses.

Dans la suite, après avoir présenté l'état de l'art des modèles de calcul et des *middlewares* pour les réseaux mobiles ad hoc (chapitre 2), nous introduirons CiMAN (chapitre 3), une algèbre de processus qui permet la modélisation des réseaux ad hoc fortement mobiles et des algorithmes et applications conçus pour ce contexte.

Chapitre 2

Modèles de calcul et *middlewares* existants

Sommaire

Avant de présenter nos travaux portant sur la définition d'un nouveau modèle de calcul pour les réseaux ad hoc fortement mobiles (chapitre 3) et sur le développement d'une bibliothèque pour développer des applications sur ces réseaux (chapitre 4), nous introduisons, afin de nous positionner par rapport à l'existant, le domaine des systèmes de transitions étiquetées puis les *middlewares* pour les MANets. Dans ce chapitre, nous présentons l'état de l'art de ces deux domaines de recherche à l'heure de la rédaction du présent document.

2.1 État de l'art des modèles de calcul

2.1.1 Introduction aux systèmes de transitions étiquetées

Un système de transitions étiquetées (LTS, Labeled Transition System) est un modèle de machine abstraite qui permet de simuler l'exécution d'un algorithme de processus. Plus formellement, un LTS est un triplet $(S, \Lambda, \rightarrow)$ où S est un ensemble d'états, Λ est un ensemble d'étiquettes et $\rightarrow \subset S \times \Lambda \times S$ est l'ensemble des transitions étiquetées possibles du système. Un LTS permet de modéliser les actions que peuvent effectuer les processus de ce système au travers des transitions étiquetées correspondantes. Dans la suite, on appelle évènement l'étiquette associée à une transition. Afin de modéliser des processus communicants, un LTS définit les interactions possibles entre ces processus. Il permet aussi la

comparaison de plusieurs systèmes grâce à des relations d'équivalence appelées bisimulations. Il existe souvent plusieurs niveaux de bisimulation en fonction des évènements observables ou non.

2.1.2 Modélisation des systèmes communicants

Nous nous intéressons aux LTS qui modélisent des systèmes communicants. CCS (Calculus of Communicating Systems) [Mil89] par exemple est un de ces modèles. Créé par Robin Milner dans les années 1980, il permet de modéliser des communications synchrones, toujours deux à deux. Dans un réseau mobile ad hoc, les nœuds sont (i) mobiles et (ii) communiquent en utilisant des technologies sans fil et donc du *broadcast*. Ces deux caractéristiques ne sont pas supportées par CCS. Nous présentons ci-dessous deux autres LTS qui traitent de ces questions. Le premier permet la modélisation des communications de type *broadcast* entre les nœuds et le second permet de définir des processus mobiles.

Broadcast. K.V.S. Prasad a développé en 1993 un modèle nommé CBS (Calculus of Broadcasting Systems) [Pra93, Pra95] qui permet de modéliser des communications de type *broadcast*. En CBS, les processus émettent des messages chacun leur tour en utilisant une opération de *broadcast*. Un message envoyé est instantanément reçu par les autres processus. Les conflits qui peuvent se produire lorsque deux processus doivent envoyer un message sont résolus de manière non déterministe. En revanche, CBS ne permet pas de modéliser la mobilité des processus. Chaque processus peut communiquer avec tous les autres processus : le réseau sous-jacent est un réseau totalement connecté dans lequel il existe un lien de communication bidirectionnel entre chaque paire de processus. CBS modélise ainsi un réseau de type Ethernet dans lequel tous les processus ont accès au même médium de communication (par exemple un bus).

Mobilité. Dans un réseau mobile ad hoc, les liens de communication entre les processus dépendent notamment de la position de ces derniers et peuvent apparaître ou disparaître à tout instant. Afin de modéliser une topologie dynamique résultant de la mobilité des processus, Robin Milner a développé, dans les années 1990, le π-calcul [Mil99], une algèbre de processus très proche de CCS. Cependant, contrairement à CCS, le π-calcul permet la transmission des étiquettes (qui modélisent des ports de communication) entre les processus qui peuvent ainsi, lors de leur évolution, communiquer avec d'autres processus sur des ports qui leurs étaient précédemment inconnus (c'est-à-dire qui n'étaient pas présents dans leur définition syntaxique). Cette caractéristique est un moyen de modéliser la mobilité des processus en créant et en supprimant des liens de communication pendant leur exécution. Cependant, le π-calcul, tout comme CCS, ne permet la modélisation que des communications synchrones deux à deux : le *broadcast* qui est une caractéristique fondamentale de la communication dans les réseaux mobiles ad hoc ne peut pas être modélisé en π-calcul.

Broadcast et mobilité. Une algèbre de processus ne peut convenir à la modélisation des réseaux mobiles ad hoc que si elle fournit à la fois une communication à base de *broadcast*

(comme CBS) et la possibilité d'exprimer la dynamique de la topologie (comme le π-calcul). En 2001, une première algèbre de processus possédant ces deux caractéristiques a été créée, il s'agit du bπ-calcul [EM01]. Le bπ-calcul est un modèle de calcul dans lequel le *broadcast* est l'unique moyen de communication entre les processus. Il s'inspire à la fois du π-calcul [Mil99] et de CBS [Pra93, Pra95]. Il permet aux processus de communiquer grâce à des canaux (comme en π-calcul) et uniquement en utilisant du *broadcast* (comme en CBS). L'émission d'un message est non bloquante et la présence d'un récepteur n'est pas nécessaire pour qu'un message puisse être émis. Les processus ne reçoivent bien sûr pas les messages envoyés sur les canaux sur lesquels ils ne sont pas en attente de réception. Ils ne peuvent en revanche pas ignorer les messages envoyés sur un canal sur lequel ils sont en attente et la perte d'un message est explicite en bπ-calcul. Enfin, les pertes silencieuses de messages ne sont pas modélisées. Par ailleurs, tout comme en π-calcul, la mobilité des processus est la conséquence de la transmission de canaux de communication. Cette approche ne nous paraît pas très naturelle pour modéliser la mobilité des nœuds dans les réseaux ad hoc fortement dynamiques.

2.1.3 Modèles de calcul pour réseaux mobiles ad hoc

Les modèles présentés précédemment n'ont pas été créés pour le contexte des réseaux mobiles ad hoc. En revanche, ces cinq dernières années, plusieurs modèles spécifiques à ce contexte particulier ont vu le jour. Nous les décrivons dans cette section par ordre chronologique des publications associées.

2.1.3.1 CBS$^{\#}$

CBS$^{\#}$ [NH06] a été créé pour permettre la spécification et l'analyse de la sécurité dans les réseaux mobiles sans fil. Ses auteurs introduisent la notion de *broadcast local* [NH06] ; seuls les nœuds voisins du nœud émetteur peuvent recevoir le message émis. La relation de voisinage est définie grâce à un graphe de connectivité qui est un graphe non-orienté dont les sommets sont des positions géographiques et dont les arêtes représentent la possibilité pour deux nœuds qui se trouvent à ces positions de communiquer. Le modèle sépare ainsi les actions des processus de leur connectivité. Dans [NH06] les auteurs argumentent que cette séparation est essentielle pour la modélisation des réseaux mobiles sans fil car, dans ces réseaux, la possibilité pour deux nœuds de communiquer est déterminée par l'environnement (comme par exemple la distance séparant ces nœuds) et non par les actions des processus. On notera que le graphe de connectivité qui représente les connexions entre les nœuds étant un graphe non-orienté, la communication entre les nœuds est donc supposée bidirectionnelle. Tout comme en CBS, la notion de port n'est pas présente : la communication par *broadcast* entre les nœuds est toujours effectuée sur le même canal. De plus, il existe au plus un processus qui peut émettre par étape de calcul, les conflits ainsi engendrés sont résolus de manière non déterministe.

2.1.3.2 CSN

CSN (Calculus for Sensor Networks) [SMLB06] est un modèle de calcul pour réseaux de capteurs. Ces derniers y sont modélisés par la donnée de l'expression d'un processus, associée à une position, une portée radio, et l'état d'une batterie. Ce dernier point est très original et n'est pas pris en compte par les autres modèles de calcul que nous avons pu étudier. Les auteurs supposent (i) que la position d'un capteur peut varier, i.e. qu'il est mobile, et (ii) que l'état de la batterie n'a pas d'influence sur la portée radio d'un capteur, sa portée étant constante au cours du temps. Concernant les interactions en CSN, les capteurs communiquent les uns avec les autres en utilisant du *broadcast*. La comparaison entre la distance séparant un capteur émetteur d'un message d'un second capteur et la portée de cet émetteur, permet de déterminer si ce second capteur reçoit ou non le message. La portée des nœuds n'est pas supposée identique : la topologie du graphe repose donc sur des liens de communication asymétriques. Enfin, les auteurs de CSN supposent que les collisions et les pertes de message peuvent être gérées à un niveau plus bas de la pile protocolaire. Les communications sont donc supposées toujours réussies, c'est à dire que les messages envoyés sont toujours reçus par les nœuds voisins du nœud émetteur.

2.1.3.3 CWS

CWS (Calculus for Wireless System) [MS06] a été créé pour modéliser les réseaux sans fil à très bas niveau. L'objectif de CWS est de mettre en avant les interférences qui peuvent se produire lors des communications entre les dispositifs. Pour cela, les opérations d'émission et de réception ne sont pas supposées atomiques (contrairement à tous les autres modèles présentés ici), mais elles sont modélisées au plus proche de la réalité, c'est à dire avec une transmission progressive des messages. Si la transmission d'une trame ne se termine pas à cause d'une interférence par exemple, l'ensemble de la trame est ignoré par le récepteur. Par ailleurs, les nœuds possèdent une portée radio parfaitement circulaire et ne sont pas supposés mobiles pendant une transmission. Les auteurs argumentent que le temps nécessaire à une transmission est suffisamment faible pour qu'il soit raisonnable de considérer que les nœuds sont immobiles pendant une telle opération. Ils donnent comme exemple un nœud qui se déplace à 5 m/s (18 km/h) en envoyant un message de 2000 octets à une vitesse de 2 Mbits par seconde. La transmission prendrait alors 0.008 seconde, ne permettant au nœud de se déplacer que de seulement 4 cm.

2.1.3.4 CMN

En CMN (Calculus of Mobile Ad Hoc Networks) [Mer07], un réseau mobile ad hoc est vu comme un ensemble de nœuds qui évoluent en parallèle et qui utilisent des canaux et une primitive de *broadcast local* (même sens que pour CBS$^{\#}$) synchrone pour communiquer. Chaque nœud possède une identité unique, une position, une portée radio (pas forcément la même pour tous) et un *tag* indiquant s'il est mobile ou statique. Comme en CSN, la comparaison entre la distance séparant un émetteur et un nœud tierce avec la portée de l'émetteur permet de déterminer si ce nœud tierce reçoit ou non un message provenant de l'émetteur. Les transmissions sont supposées atomiques et infaillibles i.e. sans perte

possible de messages. Massimo Merro, l'auteur de CMN, suppose l'existence de protocoles sous-jacents qui permettent d'éviter les collisions lors des communications.

2.1.3.5 CMAN

CMAN (Calculus for Mobile Ad Hoc Networks) [God07] est un modèle de calcul pour les MANets dont l'objectif est la modélisation des protocoles de routage dits cryptographiques comme par exemple ARAN [SDL+02]. L'objectif "cryptographique" d'ARAN est de s'assurer que les requêtes de routage qu'il utilise sont signées et vérifiées lors de la création des tables de routage du réseau. Le protocole est dit sûr si aucune information incorrecte de routage ne peut être injectée ou imposée par un nœud malveillant.

Dans CMAN, les nœuds ont une position géographique et peuvent communiquer les uns avec les autres en utilisant une opération de *broadcast local* (même sens que pour CBS$^{\#}$) qui modélise une opération d'émission radio non adressée. Les transmissions entre les nœuds sont atomiques, i.e. l'émission et la réception d'un message se produisent lors d'une unique transition. Seuls les nœuds qui se trouvent dans le voisinage immédiat d'un nœud émetteur peuvent recevoir le message émis. La position des nœuds est une position logique et la topologie du réseau fait partie intégrante de la définition syntaxique du système. Les changements de topologie sont alors la conséquence des étapes de calcul représentant l'évolution du système. Jens Chr. Godskesen, l'auteur de CMAN, argumente que les liens de communication entre les entités qui composent un MANet ne sont pas toujours bidirectionnels. Cependant, la supposition qu'ils le sont est tout de même faite dans CMAN : *[...] we adopt that communication between nodes in a network is carried out on bidirectionnal links, [...]* [God07]. Tout comme dans CMN, les transmissions sont atomiques et les messages peuvent ne pas être reçus par certaines entités en écoute. Cependant, contrairement à CMN, il n'est pas possible de restreindre des canaux de communication à certains nœuds ; autrement dit, les messages sont envoyés sur un médium de communication qui n'est pas "restreignable".

2.1.3.6 ω-calculus

Le ω-calcul [SRS08] est, comme les modèles précédemment décrits, un modèle de calcul pour les réseaux mobiles ad hoc. Les nœuds que le ω-calcul permet de définir communiquent les uns avec les autres grâce à une opération de *broadcast local* (même sens que pour CBS$^{\#}$) et utilisent tous le même canal pour communiquer. Ce modèle est décliné en plusieurs versions dont le ω_1-calculus [SRS08] qui, par l'ajout de restrictions sur les étiquettes qui représentent les opérations d'émission et de réception par *broadcast*, permet de simuler l'utilisation de canaux. Par ailleurs, en ω-calcul, les nœuds possèdent une portée radio circulaire constante et identique pour chacun. Il en résulte que les liens de communication sont donc implicitement supposés bidirectionnels.

2.1.4 Bilan

Nous résumons dans le tableau 2.1 les caractéristiques de chacun des modèles de calcul étudiés. Il est intéressant de remarquer qu'aucun des modèles présentés ne permet d'exprimer à la fois (i) la mobilité des nœuds (ou processus), (ii) une communication entre les nœuds de type *broadcast* asymétrique asynchrone non atomique sur des canaux, et (iii) des pertes de messages possibles et non détectables lors de communications inter-nœuds. Nous pensons pourtant que ces points sont importants dans le contexte qui nous préoccupe.

	mobilité des processus	type de communication	utilisation de canaux	communication asymétrique	communication asynchrone	transmission atomique	perte de message
CCS		deux-à-deux	✓			✓	
π-calcul	✓	deux-à-deux	✓			✓	
CBS		*broadcast*				✓	
bπ-calcul	✓	*broadcast*	✓			✓	
CBS$^{\#}$	✓	*broadcast*				✓	
CSN	✓	*broadcast*		✓	✓	✓	
CWS		*broadcast*	✓	✓	✓		✓
CMN	✓	*broadcast*	✓	✓		✓	
CMAN	✓	*broadcast*				✓	✓
ω-calcul	✓	*broadcast*				✓	✓

Tab. 2.1: Résumé des caractéristiques des modèles de calcul pour MANets

2.2 État de l'art des *middlewares*

Ces dernières années, une partie de la communauté scientifique s'est intéressée aux réseaux mobiles ad hoc. La diversité des problèmes rencontrés dans ces réseaux a eu pour conséquence la création d'un grand nombre de solutions logicielles. Nous en présentons un sous-ensemble, celles qui nous semblent les plus pertinentes. Dans un premier temps, nous détaillons les différents paradigmes sur lesquels s'appuient ces *middlewares*. Ensuite, nous présentons succinctement l'objectif et les caractéristiques de chacun d'eux ainsi que le contexte dans lequel ils ont été conçus.

2.2.1 Paradigmes de communication

Nous avons choisi de classer les *middlewares* selon trois grandes familles de paradigmes de communication.

2.2.1.1 Communication par *Tuple Space*

Genèse. Le paradigme de coordination et de communication par *Tuple Space* a été introduit par Gelertner en 1985 avec Linda [Gel85]. Il a été créé pour les systèmes distribués dans lesquels les dispositifs sont fixes et les liens de communication entre ces dispositifs sont stables. Un *Tuple Space* est une zone mémoire partagée par les différents nœuds qui composent le système ; elle est distribuée sur les nœuds où sont placés des *Tuples*. Un *Tuple* est un ensemble de données, c'est la traduction anglaise de n-uplet.

Primitives sous-jacentes. Dans Linda la communication entre les nœuds se fait de manière asynchrone grâce à trois primitives. La primitive **out(t)** injecte le *Tuple t* dans le *Tuple Space*. La primitive **in(t)** lit un *Tuple* qui correspond au modèle (*pattern matching*) du *Tuple t* en le retirant du *Tuple Space*. La primitive **read(t)** lit un *Tuple* qui correspond au modèle du *Tuple t* mais sans le retirer du *Tuple Space*. Les opérations d'ajout (`out`) et de suppression (`in`) de *Tuples* dans le *Tuple Space* sont des opérations en exclusion mutuelle ; il est donc impossible que deux nœuds les effectuent simultanément.

Adaptation à un environnement dynamique. Il existe aujourd'hui un grand nombre de *middlewares* pour réseaux mobiles ad hoc qui reposent sur le paradigme de *Tuple Space* (voir plus bas). Or, les réseaux mobiles ad hoc ont une topologie qui ne cesse d'évoluer contrairement aux réseaux statiques pour lesquels ce paradigme a initialement été pensé. Cette évolution permanente engendre la création et la disparition de liens de communication entre les nœuds, modifiant ainsi la composition même du *Tuple Space*. En effet, comme dans un cadre statique, un *Tuple Space* dans un contexte dynamique est l'agrégation des *Tuples Space* locaux des nœuds qui composent le réseau. Plus précisément, il est l'union des *Tuple Space* locaux des nœuds qui appartiennent à une même composante connexe. Ces nœuds, mobiles, sont par définition susceptibles d'apparaître et de disparaître à tout instant, modifiant donc de fait le *Tuple Space*. La présence ou non d'un *Tuple* dans un *Tuple Space* n'est alors plus seulement due aux actions des nœuds qui injectent ou

retirent des *Tuples* dans le *Tuple Space* mais est aussi la conséquence de la mobilité des nœuds.

Hypothèses sous-jacentes. Il est explicitement défini dans [Gel85] qu'une implémentation de *Tuple Space* est utilisable lorsque les problèmes de synchronisation et de perte de *Tuple* sont suffisamment rares pour ne pas devoir être pris en compte lors du développement d'une application. "*In a usable implementation of TS, failures will be rare enough not to have to be taken into account in the course of normal program development - in the same sense in which system failures, file crashes, and so on take place but do not impede normal program development on a useful conventional installation*" [Gel85]. De même que les problèmes système ou matériel critiques n'ont pas d'influences sur le développement d'un programme classique, les défaillances du *Tuple Space* doivent être suffisamment rares pour pouvoir être totalement ignorées lors du développement d'une application l'utilisant.

Comme expliqué précédemment, dans un cadre dynamique, un *Tuple Space* est l'union partagée des *Tuple Spaces* distribués sur les nœuds qui appartiennent à une même composante connexe. Il en résulte que : (i) la relation d'appartenance à une composante connexe étant transitive, la relation d'interaction entre les nœuds est une relation transitive, et (ii) le mécanisme d'exclusion mutuelle pour l'utilisation des opérations in et out sur le *Tuple Space* nécessite une synchronisation entre les nœuds. Cette synchronisation n'est possible que s'ils sont mutuellement à portée les uns des autres. La relation d'interaction entre les nœuds est donc non seulement transitive mais aussi symétrique. En conséquence, ce paradigme de communication n'exploite pas les liens de communication asymétriques qui peuvent exister dans un MANet, et nécessite de part par la transitivité requise une relation de voisinage multi-sauts. Il semble donc peu adapté au contexte que nous visons (cf. section 1.2 page 14) dans lequel l'existence de canaux symétriques ne peut pas être garantie. Dans nos travaux nous pousserons le raisonnement jusqu'au bout, en n'utilisant que des liens de communications asymétriques entre les nœuds, seul type de liaison dont on puisse garantir l'existence.

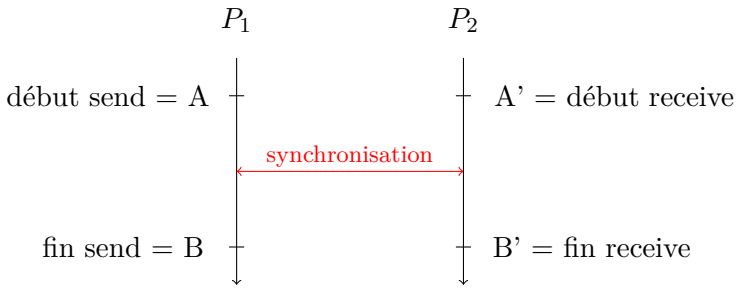
***Middlewares* orientés *Tuple Space*.** Bayou (section 2.2.2.4 page 30), EgoSpaces (section 2.2.2.7 page 35), LIME (section 2.2.2.14 page 42), Limone (section 2.2.2.15 page 43), MESHMdl (section 2.2.2.16 page 45), PeerWare (section 2.2.2.20 page 48) et TOTA (section 2.2.2.25 page 53) sont des *middlewares* orientés *Tuple Space* pour les réseaux mobiles ad hoc. Nous détaillons leurs caractéristiques dans la section 2.2.2 page 25.

2.2.1.2 Communication par messages

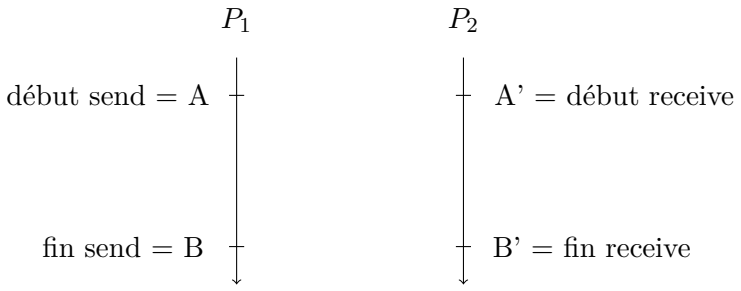
Le paradigme de communication par messages est le plus naturel des paradigmes de communication. Il repose sur deux primitives élémentaires que sont *send* et *receive*. La primitive *send* permet l'envoi d'un message, et la primitive *receive* permet la réception d'un message. On distingue deux types de communication par message, la communication synchrone et la communication asynchrone.

Communication synchrone. Lors d'une communication synchrone, l'émetteur et le récepteur sont bloqués jusqu'à ce que le transfert du message prenne place (cf. figure 2.5a). Les primitives *send* et *receive* sont toutes deux bloquantes, et seul le transfert du message permet aux deux entités alors en attente d'évoluer. Dès lors que la communication est de type *broadcast* ou *multicast*, i.e. faisant intervenir plus de trois nœuds, elle nécessite l'utilisation préalable d'une procédure de rendez-vous. Or cette procédure de rendez-vous repose sur la possibilité pour l'ensemble des nœuds de communiquer les uns avec les autres. La communication synchrone repose donc sur la présence de liens de communication bidirectionnels. Or, comme nous l'avons vu dans le chapitre 1, les liens de communication dans un MANet ne sont pas tous bidirectionnels. La communication synchrone ne semble donc pas être la communication la plus adaptée à ce contexte.

Communication asynchrone. Lors d'une communication asynchrone, la primitive d'émission *send* n'est pas bloquante (cf. figure 2.5b). La réception quant à elle peut être bloquante ou non bloquante.



(a) Communication synchrone - B ne peut pas se produire avant A' et B' ne peut pas se produire avant A



(b) Communication asynchrone - B peut se produire avant A'

FIG. 2.5: Communication synchrone vs. communication asynchrone

***Middlewares* orientés messages (MOM).** 7DS (section 2.2.2.1 page 26), Ad-Hoc InfoWare (section 2.2.2.2 page 27), AGAPE (section 2.2.2.3 page 28), CARISMA (section 2.2.2.5 page 31), DoDWAN (section 2.2.2.6 page 33), EMMA (section 2.2.2.8 page 36), Expeerience (section 2.2.2.9 page 37), JASON (section 2.2.2.11 page 39), JMobiPeer (section 2.2.2.12 page 40), JXME (section 2.2.2.13 page 41), Mobile Chedar (section 2.2.2.17 page 45), Mobile Gaia (section 2.2.2.18 page 46), MoCoA (section 2.2.2.19 page 47), SCOMET

(section 2.2.2.22 page 49), STEAM (section 2.2.2.24 page 52), XMIDDLE (section 2.2.2.26 page 55) sont des *middlewares* orientés messages pour les réseaux mobiles ad hoc. Nous détaillons leurs caractéristiques dans la section 2.2.2 page 25.

2.2.1.3 Communication par agents mobiles

Un agent mobile est une entité autonome en terme de déplacement et capable d'effectuer des opérations. Il possède une connaissance partielle de son environnement et peut se déplacer de nœud en nœud du réseau. Le paradigme des agents mobiles permet de se rapprocher d'un modèle de communication client/serveur mais dans lequel il est en particulier possible d'optimiser la taille des données qui circulent sur le réseau. Pour cela, on rapproche physiquement les agents des données dont ils ont besoin pour s'exécuter, favorisant ainsi les interactions locales.

On pourra par ailleurs noter que dans un *middleware* orienté *Tuple Space* tel que TOTA (cf. section 2.2.2.25 page 53), les *Tuples* sont des données qui contiennent parfois leurs propres règles de propagation. Le paradigme de communication par agents mobiles se rapproche donc, en cela, du paradigme de communication par *Tuple Space*.

Hypothèses sous-jacentes. Dans les systèmes à base d'agents mobiles, les agents se déplacent de nœud en nœud mais ne se multiplient pas. Une condition nécessaire à leur déplacement est une synchronisation entre les nœuds. Lors du déplacement d'un agent mobile, une absence de synchronisation entre les nœuds ne permet pas de garantir (i) sa non duplication, et (ii) sa non disparition. Cette synchronisation nécessite (cf. section 2.2.1.1 page 22) la présence de liens de communication symétriques. Ce paradigme de communication n'exploite donc pas les liens de communication asymétriques qui peuvent pourtant exister de manière significative dans un MANet (cf. annexe A page 163). Ce paradigme de communication semble donc peu adapté au contexte que nous visons.

***Middlewares* orientés agents mobiles.** GecGo (section 2.2.2.10 page 38) et SELMA (section 2.2.2.23 page 51) sont des *middlewares* orientés agents mobiles pour les réseaux mobiles ad hoc. Nous détaillons leurs caractéristiques dans la section 2.2.2 ci-après.

2.2.2 Middleware existants

Dans cette section, nous présentons en détails les principaux *middlewares* pour réseaux mobiles ad hoc en suivant pour chacun le même schéma rédactionnel. Nous commençons par préciser le contexte de création du *middleware*, c'est à dire ses auteurs ainsi que le laboratoire de recherche dans lequel les travaux associés ont pris place et l'année ou les années pendant lesquelles il a été l'objet d'un travail actif. Nous présentons ensuite l'objectif principal poursuivi par ses auteurs puis ses caractéristiques techniques. A partir de celles-ci nous identifions les hypothèses sur lesquelles s'appuient ces *middlewares*, qu'elles soient explicitement décrites par les auteurs ou implicites, dictées par les caractéristiques techniques du *middleware* considéré. Enfin, nous donnons l'état courant des travaux associés.

2.2.2.1 7DS

Contexte de réalisation. 7DS[1] (*Seven Degrees of Separation*) [SSAMSGHHGS07, AMSSHS08] a été développé par Suman Srinivasan, Arezu Moghadam, Se Gi Hong et Henning Schulzrinne de 2007 à 2008 à l'*Internet Real Time Laboratory, Columbia University, New York* aux États-Unis. 7DS tient son nom de la théorie *Six degrés de séparation* (aussi appelée *Human Web*) qui a été établie par un écrivain hongrois nommé Frigyes Karinthy dans une de ses nouvelles *Chaînes* extraite de son recueil de nouvelles *Everything is different* [Fri29] publié en 1929. Il y écrit que si l'on considère qu'une personne est à distance 1 des personnes qu'il connaît de manière directe, à distance 2 des personnes que connaissent les personnes qu'il connaît directement, etc., alors il est au plus à distance 6 de toutes les personnes sur le globe.

Objectif. 7DS se place dans un contexte dans lequel les utilisateurs souhaitent naviguer sur le Web. Son objectif est de maximiser la propagation d'informations provenant du réseau DTN vers Internet et réciproquement. Chaque nœud stocke localement tous les documents récupérés depuis Internet par son propriétaire. Il utilise ensuite ces informations afin de simuler un accès Internet depuis le réseau mobile ad hoc : lorsque ses nœuds voisins tentent de récupérer sur Internet un document qu'il possède localement dans son cache, il agit alors comme passerelle et simule une connexion Internet en proposant les documents qu'il possède dans son cache.

Caractéristiques techniques (et suppositions implicites). 7DS est un *middleware* orienté messages (MOM) pour les réseaux mobiles ad hoc qui peuvent être déconnectés (DTN). Le système repose sur deux suppositions : tout d'abord, les dispositifs mobiles sont sporadiquement en contact avec d'autres terminaux et finissent toujours par se connecter à Internet ; ensuite, l'intérêt pour les sujets d'information populaires est partagé par un grand nombre d'utilisateurs. Il existe donc une forte probabilité pour que l'information recherchée par un dispositif mobile soit déjà stockée dans le cache d'un dispositif voisin.

Un nœud est parfois non connecté à Internet, parfois connecté, mais ce toujours de manière sporadique. Lors d'une connexion d'un nœud à Internet, celui-ci peut être amené à jouer le rôle de passerelle, créant alors un point de centralisation temporaire dans le système.

7DS fonctionne au dessus de la couche IP, et donc au dessus de toutes les technologies de communication qui sont capables de proposer une telle couche. Les dispositifs obtiennent une adresse IP sans passer par un serveur DHCP mais grâce à Zeroconf [Zer] (*Zero Configuration Networking Specification*). 7DS utilise le protocole mDNS [Mul] (multicast DNS) dont l'implémentation la plus populaire, qui est développée par Apple, se nomme Bonjour [App]. 7DS utilise la bibliothèque BonAHA [SSAMHS09] disponible en ligne à l'adresse `http://bonaha.sourceforge.net/`. BonAHA a aussi été développée par Suman Srinivasan, Arezu Moghadam et Henning Schulzrinne. Son objectif est de faciliter le développement d'applications pour les réseaux mobiles ad hoc. Pour cela, elle fournit au développeur une abstraction du réseau, que celui-ci soit un réseau fixe, un réseau ad

[1] `http://www.cs.columbia.edu/irt/`

hoc sans fil ou un réseau déconnecté. Elle masque ainsi à l'application et au développeur les caractéristiques réelles du réseau utilisé.

7DS a été écrit en C et a été testé sur Linux, MacOS X et les systèmes Windows. Il n'existe pour l'instant pas de version pour les plateformes Symbian, Windows Mobile, Android ou iPhone OS.

Aucun mécanisme de sécurité n'est mis en avant dans les articles qui décrivent 7DS.

État actuel. 7DS a fait l'objet de publications récentes à la date de rédaction du présent document, et le projet semble donc toujours actif. Le code de 7DS est accessible en ligne à l'adresse `http://bonaha.cvs.sourceforge.net/bonaha`.

2.2.2.2 Ad-Hoc InfoWare

Contexte de réalisation. Ad-Hoc InfoWare[2] [PGGH03, PAD+05] a été développé par Thomas Plagemann, Vera Goebel, Ellen Munthe-Kaas, Norun C. Sanderson, Matija Puzar, Ovidiu Valentin Drugan, Katrine Stemland Skjelsvik, Anna Lekova-Kovacheva et Jon Egil Andersson (Thalès Communication) de 2003 à 2007 au sein du groupe de recherche *Distributed Multimedia Systems (DMMS)* du département Informatique de l'Université d'Oslo en Norvège.

Objectif. Ad-Hoc InfoWare a pour objectif de proposer un ensemble de services permettant l'accès à l'information et la diffusion de celle-ci dans un réseau ad hoc. Ils sont fournis au niveau d'une couche *middleware*. Plus spécifiquement, il propose des services pour le partage d'information dans le cadre d'opérations de secours et de sauvetage [PAD+05]. Son but est de permettre le partage d'informations entre toutes les organisations concernées : il met à leur disposition des fonctionnalités voisines de celles fournies par les systèmes de base de données distribuées de haut niveau mais dans un environnement mobile ad hoc.

Caractéristiques techniques (et suppositions implicites). Ad-Hoc InfoWare est un *middleware* orienté message (MOM). Il a été conçu pour supporter la mobilité engendrée par des équipes de sauvetage sur le terrain.

Certains nœuds peuvent posséder un accès à Internet et sont alors utilisés comme passerelle. Lors de la mise en place d'une opération de sauvetage (c'est à dire avant le déploiement des dispositifs sur le terrain) il est nécessaire que tous les nœuds possèdent un accès Internet pour la distribution des certificats. De plus, ce *middleware* suppose qu'il existe des mécanismes de routage entre les nœuds. Le partitionnement pourtant probable du réseau lors d'une opération de sauvetage n'est pas mentionné dans les articles qui décrivent Ad-Hoc InfoWare.

Les technologies de communication supportées pour la communication entre les nœuds sont le Wi-Fi, le Bluetooth et l'Infrarouge.

[2] `http://heim.ifi.uio.no/~infoware/`

Le matériel sur lequel il peut s'exécuter va du PDA à l'ordinateur portable. Les tests de 7DS effectués par ses auteurs ont été réalisés sur un PDA Compaq iPAQ (processeur SA-1110 206 MHz avec 64 Mo de RAM) et sur deux PC (processeur Pentium II à 350 MHz avec 512 Mo de RAM pour l'un et processeur Pentium III à 500 MHz avec 384 Mo de RAM pour l'autre) [PGGH03]. Il n'est cependant pas précisé sur quels systèmes d'exploitation fonctionne Ad-Hoc InfoWare.

Sécurité. Contrairement à la plupart des *middlewares* pour MANet, Ad-Hoc InfoWare apporte de la sécurité dans le réseau. Il utilise un système à base de PKI (*Public Key Infrastructure*) [CSF+08]. La distribution de certificats est faite dans une première étape, de manière centralisée, avant le déploiement des nœuds. L'utilisation de la PKI par les terminaux mobiles est somme toute classique. Ces derniers utilisent la clef publique de la même autorité de certification dont la signature peut ainsi être vérifiée par chaque nœud. De cette manière, il n'est pas nécessaire pour un nœud de contacter l'autorité de certification afin de s'assurer de la validité d'un certificat. L'absence de CA empêche cependant la mise en place d'un mécanisme de révocation. La manière dont sont stockées les clefs n'est pas précisée.

Dans [PAD+05], il est indiqué que la sécurité du système est primordiale (les journalistes ne doivent pas pouvoir accéder aux enregistrements médicaux, des terroristes ne doivent pas pouvoir saboter l'opération de sauvetage en supprimant certaines données, etc.). Deux types d'attaques sont distinguées : les attaques internes et les attaques externes. Les attaques externes peuvent être dues à des manipulations de messages, des injections de fautes dans les tables de routage, de la congestion de trafic, etc. Ad-Hoc InfoWare utilise du chiffrement[PAD+05] pour empêcher certaines de ces attaques. Les auteurs argumentent que les attaques de type déni de service ne peuvent pas être gérées au niveau du *middleware* et elles ne sont donc pas traitées par celui-ci. Les attaques internes, c'est à dire provenant d'un nœud qui a déjà été authentifié sont plus difficiles à détecter et à gérer. Une solution proposée dans [PAD+05] est d'intégrer l'adresse IPv6 d'un nœud dans son certificat [Cas02]. Cela permet d'empêcher les nœuds d'utiliser une autre adresse IP que la leur et ainsi de pouvoir *blacklister* les nœuds malveillants.

État actuel. Il n'y a pas de code disponible sur le site web du projet qui s'est terminé en mars 2007. La dernière publication dans le cadre de celui-ci date aussi de mars 2007.

2.2.2.3 AGAPE

Contexte de réalisation. AGAPE[3] (*Allocation and Group Aware Pervasive Environment*) [BCM03, BCM04, BCM05] a été développé par Antonio Corradi, Rebecca Montanari, Dario Bottazzi, Andrea Zuccherelli, Marco Pivi, Stefano Landi, Gaetano Cocci Grifoni, Michele Baldassarro, Federico Bonazza, Fabio Fioretti, Gianpiero Napoli, Marco Marchini, Antonio Gaetani, Marco Montali et Luca Campeti de 2003 à 2005 à l'Université de Bologne en Italie.

[3] `http://lia.deis.unibo.it/research/AGAPE/LIAIndex.html`

Objectif. L'objectif d'AGAPE [BCM03, BCM04, BCM05] est de permettre aux applications d'utiliser la notion de groupe dans les réseaux mobiles ad hoc. Il est en charge des opérations classiques de gestion de groupe. Ce dernier est composé d'un *Cluster Head* (CH) et de *Managed Entities* (ME). Le CH est désigné dynamiquement et est en charge de la gestion du groupe. Son rôle est notamment d'accepter ou de refuser une nouvelle ME et de donner une vision du groupe aux ME qu'il gère.

Caractéristiques techniques (et suppositions implicites). AGAPE est un *middleware* orienté message (MOM), écrit en Java et qui fournit plusieurs services liés à la gestion de groupes. Le *Proximity Service* (PS) est en charge de la diffusion des *beacons* d'annonce. Il permet ainsi aux CH et ME de s'annoncer auprès de leurs voisins. Chaque nœud fait suivre les *beacons* qu'il reçoit en décrémentant leur TTL (Time To Live) s'il n'est pas égal à 0. En fonction de la densité du réseau, certains nœuds peuvent aussi ne pas faire suivre les *beacons* afin d'éviter une tempête de diffusion (*broadcast storm*). Le *Proximity Enabled Naming System* (PENS) est pour sa part en charge de la gestion des GID (*Group Identifier*) qui sont supposés uniques dans le système et des PID (*Personal Identifier*) qui sont supposés uniques pour un même groupe. Ce service maintient une table de correspondance PID/GID pour chaque *beacon* reçu. Quand un nouveau *beacon* est reçu, une entrée est créée, et réciproquement, lorsqu'un *beacon* n'est plus reçu durant une certaine période de temps, l'entrée correspondante est supprimée. Le *View Manager* (VM) est en charge de créer et de disséminer la vision du groupe aux membres de celui-ci. Le *Join/Leave Manager* (J/LM) permet aux entités non membres d'un groupe de le rejoindre et aux entités membres de le quitter. Enfin, le *Cluster Head Designation Service* (CHDS) est en charge d'élire un CH. Un CH est réélu lorsque le CH courant ne peut plus remplir son rôle (batterie faible ou mémoire insuffisante par exemple).

Les auteurs supposent explicitement que le réseau sous-jacent peut être déconnecté. On notera en particulier que les CH ne sont pas des points de centralisation car, même s'ils ont un rôle spécifique, ils sont élus de manière non définitive.

Dans [BCM04, BCM05], les auteurs supposent la présence d'un protocole de routage comme AODV [PBRD03]. Ils précisent cependant qu'AGAPE a été créé sans supposition sur la disponibilité d'un algorithme de routage particulier.

AGAPE fonctionne sur IP et donc avec toutes les technologies qui peuvent fournir une couche IP.

Les tests effectués par les auteurs d'AGAPE ont utilisé des PC portables et des PDA Compaq iPAQ.

Gestion d'énergie. Contrairement à la plupart des *middlewares* pour MANets, la gestion d'énergie est prise en considération dans AGAPE [BCM04]. Par exemple, un CH peut décider de ne plus être CH si sa batterie est trop faible.

Résultats des simulations et des tests. Les tests dont les résultats sont présentés dans [BCM05] montrent la charge réseau nécessaire à la gestion d'un groupe en fonction de la densité de celui-ci et en fonction de la fréquence d'envoi des *beacons*. Le pire scénario

est celui où l'ensemble des dispositifs qui forment un groupe sont à portée les uns des autres. En effet, dans ce cas, la surcharge due à la réémission (inutile) des *beacons* reçus est d'autant très importante.

État actuel. La dernière publication traitant d'AGAPE date de 2004. Le projet semble donc inactif. Le code n'est pas disponible en ligne.

2.2.2.4 Bayou

Contexte de réalisation. Bayou [DPS+94, TTP+95] a été développé par Alan Demers, Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin Theimer, Brent Welch er Carl H. Hauser de 1994 à 1995 au centre de recherche de Xerox à Palo Alto en Californie aux États-Unis.

Objectif. Bayou [DPS+94, TTP+95] a pour objectif principal de permettre le partage de données sur terminaux mobiles et plus spécifiquement sur des PC portables, en proposant une base de données distribuée. Il est étudié ici car il est très présent dans la littérature associée aux *middlewares* pour les réseaux mobiles ad hoc bien que ce domaine n'existait pas lors de sa création en 1994.

Caractéristiques techniques (et suppositions implicites). Bayou est un *middleware* orienté *Tuple Space*.

La mobilité sous-jacente supposée n'est pas précisément définie et il n'y a pas non plus d'exemple concret d'utilisation permettant d'identifier le type de mobilité que ce *middleware* est capable de gérer. Cependant, Bayou se place dans un cadre où le graphe de communication entre nœuds n'est pas forcément connexe. Il ne repose pas sur un modèle P2P pur mais sur un modèle client/serveur et n'a pas besoin d'accéder à une autorité de certification pour permettre l'établissement de canaux de communication sécurisés entre un client et un serveur ou entre deux serveurs Bayou. Aucune opération de routage entre les nœuds n'est requise pour l'utilisation de Bayou.

Dans [DPS+94, TTP+95], les auteurs ne spécifient pas les interfaces de communication gérées, mais ils affirment que Bayou peut utiliser toutes les technologies existantes, quelles qu'elles soient, y compris par copie sur disquette et déplacement de celle-ci.

Bayou peut être exécuté sur des PC portables avec de faibles ressources (au sens de 1995) et sur des PDA. Les systèmes d'exploitation sur lesquels Bayou fonctionne ne sont pas précisés.

Sécurité. Bayou utilise un système à base de *Public Key Infrastructure* (PKI) pour l'authentification des utilisateurs et le chiffrement des échanges. L'utilisation de la PKI par les nœuds une utilisation classique. En effet, chaque nœud possède localement la clef publique de la même autorité de certification dont la signature peut ainsi être vérifiée par tous les nœuds sans qu'un accès vers l'autorité de certification ne soit nécessaire. Bayou ne fournit donc pas de mécanisme de révocation.

Résultats des simulations et des tests. Les performances du *middleware* sont décrites dans [TTP+95]. Les auteurs donnent le temps nécessaire aux lectures, écritures (avec et sans conflit) des éléments (sous forme de t-uplet) depuis et vers une base de données, ainsi que l'évolution de la taille de cette dernière en fonction du nombre d'opérations de lecture et d'écriture effectuées. Les résultats obtenus montrent l'efficacité de leur middleware.

État actuel. La version dénommée "version initiale" est décrite dans [TTP+95]. Il n'existe pas d'autre article qui traite de ce *middleware* et nous pouvons donc supposer que le projet de recherche dont il découle n'est plus actif.

2.2.2.5 CARISMA

Contexte de réalisation. CARISMA[4] (*Context-Aware Reflective mIddleware System for Mobile Applications*) [CEM03] a été développé par Licia Capra[5], Wolfgang Emmerich[5] et Cecilia Mascolo[5,6] de 2001 à 2003 à l'*University College of London* au Royaume-Uni.

Objectif. CARISMA [CEM03] est un *middleware context-aware*. Un *middleware context-aware* est un *middleware* qui fournit à l'application des informations relatives au contexte. Ces informations peuvent par exemple concerner l'état de charge de la batterie, la position GPS du nœud, etc. Dans [CEM03], les auteurs argumentent que dans un contexte de réseau mobile ad hoc, un *middleware* qui masque les changements de contexte à l'application ne permet pas à celle-ci d'exploiter ce type de réseau[7]. CARISMA est réflexif, c'est à dire qu'il autorise l'application à définir un certain nombre de politiques applicables au niveau du middleware et qui dépendent du contexte. Ceci permet d'exploiter au niveau applicatif la diversité des contextes possibles. Par exemple, une application peut choisir d'envoyer des messages chiffrés lorsque la batterie est chargée à plus de 40% et en clair si elle est chargée à moins de 40% (figure 2.6). Nous pouvons remarquer que le cas où la batterie est chargée à 40% n'est pas traitée.

```
messagingService
   plainMsg
      battery < 40%
   encryptedMsg
      battery > 40%
```

Fig. 2.6: Exemple de politique extraite de [CEM03]

Ce *middleware* autorise aussi la combinaison de plusieurs caractéristiques contextuelles pour la création de politiques. Ceci pouvant engendrer des conflits entre les différentes

[4] `http://www.cs.ucl.ac.uk/staff/L.Capra/research/carisma.html`

[5] aussi présent(e) dans XMIDDLE (cf. section 2.2.2.26 page 55)

[6] aussi présent(e) dans EMMA (cf. section 2.2.2.8 page 36)

[7] Nous partageons cette idée, à savoir que le middleware ne doit pas chercher à cacher le contexte (par exemple la mobilité) à l'application pour lui permettre d'en tirer parti.

politiques, CARISMA fournit un mécanisme permettant de les gérer automatiquement en utilisant une approche micro-économique [Var92].

Les auteurs de [CEM03] définissent qu'il existe un conflit entre deux politiques lorsque leurs conditions d'applications ne sont pas disjointes, c'est à dire que le *middleware* ne peut pas déterminer la politique à utiliser.

Caractéristiques techniques (et suppositions implicites). CARISMA est un *middleware* orienté message (MOM). Le type de mobilité des nœuds n'est pas spécifié dans l'article qui traite de CARISMA que nous avons étudié.

Les nœuds sont connectés les uns aux autres en mode ad hoc et il n'y a pas de point de centralisation.

Il n'est pas précisé sur quels systèmes d'exploitation CARISMA peut fonctionner.

Gestion d'énergie. Le *middleware* est *context-aware* et réflexif. Comme nous l'avons vu l'application peut ainsi modifier le comportement du middleware en fonction de l'état de charge de la batterie par exemple.

Détails supplémentaires. Dans [CEM03], les auteurs illustrent l'utilisation du *Reflective Model* par des exemples de définitions de profils et montrent qu'elles peuvent engendrer des conflits. Ils distinguent les conflits intra-profil (figure 2.7) des conflits inter-profil (figure 2.8) et proposent une manière de les gérer automatiquement grâce à un formalisme approprié défini dans [CEM03]. Dans l'exemple de la figure 2.7, si la batterie est chargée à plus de 40% et que le nœud est à l'extérieur alors le middleware ne peut déterminer le comportement à adopter. C'est un conflit intra-profil. Dans l'exemple de la figure 2.8, si la charge de la batterie d'Alice est inférieure à 40% mais que la bande passante entre Alice et Claire est inférieure à 50% alors Alice et Claire n'utiliseront pas le même format pour leurs messages. C'est un exemple de conflit inter-profil.

```
talkReminder
  soundAlert
    location = outdoor
  silentAlert
    battery > 40%
```

FIG. 2.7: Exemple de conflit intra-profil extrait de [CEM03]

```
% Alice
messagingService
  plainMsg
    battery < 40%
  encryptedMsg
    battery > 40%
```

```
%Claire
messagingService
  plainMsg
    bandwidth > 50%
  compressedMsg
    bandwidth < 50%
```

FIG. 2.8: Exemple de conflit inter-profil extrait de [CEM03]

Résultats des simulations et des tests. Des tests ont été effectués par les auteurs de CARISMA sur des PC portables Dell Latitude équipés de processeurs Pentium II à 300 MHz, de 128 Mo de RAM et de cartes sans fil CISCO Aironet 340 à 10 Mbps. Les auteurs expliquent avoir utilisé ce matériel car sa configuration se rapprochait des configurations

des équipements portables tels que le téléphone mobile Sony Ericsson P800 (processeur ARM9 200 MHz), le PDA iPAQ Pocket PC h5450 (processeur Intel 400 MHz et 64 Mo de RAM) ou encore la tablette PC Compaq TC1000 (processeur 1 GHz et 256 Mo de RAM). Ces tests montrent l'impact sur les performances du *middleware* (i) du mécanisme de réflexion, (ii) de l'extraction du contexte (dont la complexité dépend du nombre de capteurs qui entrent en compte dans sa définition), et (iii) du mécanisme de résolution des conflits.

État actuel. Il semble que le dernier article publié qui traite de CARISMA est [CEM03] et date de novembre 2003. Le code de ce middleware n'est pas disponible en ligne.

2.2.2.6 DoDWAN

Contexte de réalisation. DoDWAN[8] (*Document Dissemination in mobile Wireless Ad hoc Networks*) [GR04, GM07] est développé par Frédéric Guidec, Julien Haillot et Yves Mahéo depuis 2005 dans le groupe de recherche CASA du laboratoire Valoria à l'Université de Bretagne Sud (France).

Objectif. DoDWAN [GR04, GM07] a pour objectif de fournir un support pour le partage de documents dans les réseaux mobiles ad hoc qui peuvent être déconnectés (DTN).

Caractéristiques techniques (et suppositions implicites). DoDWAN est un *middleware* orienté message (MOM) intégralement développé en Java qui a été conçu pour les réseaux mobiles ad hoc possiblement déconnectés. Afin de permettre le partage de documents dans un réseau DTN, DoDWAN utilise sur un algorithme de propagation sélective. Cet algorithme se déroule en trois étapes. Tout d'abord, les nœuds annoncent le catalogue des documents qu'ils possèdent localement ainsi qu'une description de leurs domaines d'intérêt (les documents possédés par les nœuds sont *taggués* par des mots-clefs qui correspondent à des domaines d'intérêts). Ensuite, les nœuds traitent les requêtes qu'ils ont reçues et, en fonction du nombre de nœuds voisins qui présentent un intérêt pour les documents qu'ils possèdent localement, les nœuds les *broadcast* ou non. De cette manière, les nœuds peuvent recevoir les documents correspondant à leurs domaines d'intérêt tout en minimisant l'utilisation de la bande passante.

L'utilisation de DoDWAN ne repose sur aucune supposition particulière concernant la mobilité des nœuds ou sur la connexité du graphe sous-jacent et aucun routage n'existe entre les nœuds. La communication se fait par des interactions pair-à-pair.

DoDWAN utilise des cartes Wi-Fi pour établir des communications entre les différents dispositifs. Ces cartes sont configurées en mode ad hoc, partageant avec les autres dispositifs du réseau un même SSID (Service Set Identifier) et une même clef de chiffrement. DoDWAN fonctionne au dessus d'IP, sur les réseaux sans fil compatible 802.11.

Il peut s'exécuter sur les terminaux qui disposent d'une machine virtuelle Java (PC, PC portable, PDA) version 1.5 ou supérieure. Cependant, la supposition qui est faite lors

[8] `http://www-valoria.univ-ubs.fr/SARAH/presentation.shtml`

des simulations est qu'un dispositif mobile est éteint lorsqu'il est en mouvement. Cela correspond bien à la réalité concernant les PC portables mais beaucoup moins en ce qui concerne les PDAs et les téléphones mobiles.

Sécurité. Le chiffrement des communications entre les nœuds du réseau est effectué en utilisant à une clef symétrique connue de tous les nœuds. Ceci ne permet de garantir ni la confidentialité deux à deux, ni l'authentification, ni la non-répudiation.

Gestion d'énergie. DoDWAN fait parti des rares *middlewares* qui essaient de minimiser la consommation énergétique des nœuds. Les résultats de simulation montrent en cela l'efficacité du protocole de dissémination de documents, meilleure que celle de l'algorithme glouton (voir paragraphe suivant).

Résultats des simulations et des tests. Nous présentons ici les résultats des simulations mis en avant dans [GM07].

Paramètres de simulation. Afin de tester les algorithmes de propagation d'un document, des tests de DoDWAN ont été effectués avec le simulateur MADHOC [Hog05]. La simulation fait intervenir 120 dispositifs sur une aire de 1 km^2. Chaque dispositif possède une interface Wi-Fi ayant une portée de 100 mètres et peut soit rester immobile pendant une période allant de 30 secondes à 3 minutes, soit se déplacer à une vitesse comprise entre 2 et 3 m/s. Les dispositifs qui sont en mouvement sont supposés éteints tandis que les dispositifs immobiles sont quant à eux supposés allumés.

Scénario. Dans le scénario mis en place, chaque dispositif en fonction envoie un document de 50 ko toutes les 10 secondes. Les documents sont classés par thèmes. Il existe 16 thèmes dans le réseau, chaque dispositif est intéressé par exactement 2 de ces thèmes et aucun dispositif n'est intéressé par les 2 mêmes. Un nouveau document est introduit dans le réseau toutes les 2,5 secondes et chaque document a une durée de vie de 60 secondes. Les nœuds utilisent l'algorithme de propagation sélectif de DoDWAN : chaque dispositif diffuse son catalogue et les 2 thèmes qui l'intéressent toutes les 15 secondes. Chaque nœud est capable de stocker au plus 200 documents en cache.

Résultats. Les tests effectués permettent de comparer les performances de l'algorithme de propagation sélectif de DoDWAN à ceux d'un algorithme de propagation glouton.

Le nombre de documents différents reçus par chaque nœud avec l'algorithme sélectif est de l'ordre du nombre de documents envoyés sur le réseau, tandis qu'avec l'algorithme glouton, seul 1 document sur 2 environ est reçu par les dispositifs concernés. En effet, les problèmes d'interférence lors de l'utilisation de l'algorithme glouton (dues à des tempête de diffusion) diminuent quantitativement la qualité de la diffusion.

En ce qui concerne le nombre de documents diffusés par seconde dans l'ensemble du réseau, il est de 5,1 avec l'algorithme sélectif et de 179,5 (soit 35,2 fois plus) avec l'algorithme

glouton tandis que le nombre de documents reçus est de 5,6 pour l'algorithme sélectif et de 2,5 (soit environ 2 fois moins) pour l'algorithme glouton. Ces résultats sont la conséquence de plusieurs facteurs. Tout d'abord, l'utilisation de l'algorithme glouton entraîne des émissions de documents par un nœud sans qu'il existe forcément de voisin (i) intéressé par ce document et (ii) qui ne possède pas déjà ce document. Par ailleurs, les tempêtes de diffusion ou *broadcast storms* sont plus fréquentes avec l'algorithme glouton qu'avec l'algorithme de propagation sélectif.

Pour conclure, l'algorithme de propagation sélectif a un pourcentage de distribution des documents (c'est à dire un pourcentage de documents qui sont reçus par les nœuds intéressés) de 98.4% tandis que l'algorithme glouton n'atteint que 45%. De plus, la diffusion par *broadcast* utilisée par le protocole de propagation sélectif de DoDWAN permet dans certains cas à plusieurs nœuds de recevoir en même temps le même document. En effet, 9% des paquets diffusés bénéficient à plus d'un receveur ; l'utilisation du *broadcast* permet donc au protocole de propagation sélectif d'optimiser l'utilisation de la bande passante et l'énergie des nœuds.

État actuel. Le code source de DoDWAN est disponible en ligne et le projet est toujours actif à la date de rédaction du présent document.

2.2.2.7 EgoSpaces

Contexte de réalisation. EgoSpaces[9] [JR02, JR04, JR06] a été développé par Christine Julien, Guia-Catalin Roman[10] de 2002 à 2006 aux *Department of Electrical and Computer Engineering, University of Texas at Austin*, aux États-Unis et *Department of Computer Science, Washington University, St. Louis, Missouri*, aux États-Unis.

Objectif. EgoSpaces [JR02, JR04, JR06] a pour objectif de simplifier le développement d'applications *context-aware* dans les réseaux mobiles ad hoc. Une application *context-aware* est une application qui adapte son comportement en fonction de paramètres environnementaux (comme par exemple la charge de la batterie du terminal mobile, la lumière ambiante détectée, l'inclinaison du dispositif, etc.). Le scénario d'utilisation mis en avant est le suivant. On considère un ensemble de conducteurs automobile dans lequel les individus souhaitent garder une trace de toutes les voitures avec lesquelles ils ont failli entrer en collision. De cette manière, si une de ces voitures "dangereuses" est à nouveau à proximité, un avertissement sonore ou lumineux avertie le conducteur et le mettra ainsi sur ses gardes. Il est important de noter que la confidentialité n'est pas traitée dans cet exemple, chaque automobiliste pouvant ainsi récupérer des informations sur les véhicules croisés.

Caractéristiques techniques (et suppositions implicites). EgoSpaces est un *middleware* orienté *Tuple Space* écrit en Java. La mobilité sous-jacente n'est pas explicitement définie. Cependant le paradigme de communication par *Tuple Space* est compatible avec

[9] `http://users.ece.utexas.edu/~julien/egospaces.html`

[10] aussi présent dans LIME (cf. section 2.2.2.14 page 42) et Limone (cf. section 2.2.2.15 page 43)

les réseaux DTN. Chaque *Tuple* est de la forme $(name, type, value)$. Le champ *name* correspond à son nom (chaque *Tuple* doit posséder un nom unique) et le champ *type* est le type de donnée stockée dans *value*. Pour reprendre l'exemple des véhicules, une voiture est définie comme suit :

$$((type, enumeration, car), (direction, string, NORTH), (speed, integer, 65))$$

EgoSpaces innove par rapport à LIME (cf. 2.2.2.14 page 42) en introduisant la notion de vue. Une vue est une projection de toutes les données disponibles pour un agent, un agent étant une entité autonome qui peut accéder au *Tuple Space*. Voici un exemple de vue qu'un agent dont le rôle est d'avertir le conducteur en cas de rapprochement d'un véhicule "dangereux" peut déclarer : *All location data owned by collision warning agents on cars within 100 meters of my current location.* [JR06].

Comme expliqué dans la section 2.2.1.1 page 22, l'utilisation d'un paradigme de communication par *Tuple Space* implique que la relation de voisinage considérée entre les nœuds est une relation symétrique. Par ailleurs, bien qu'il semble fonctionner au-dessus d'une couche IP, les technologies de communication supportées et la configuration minimale requise par les dispositifs mobiles ne sont pas spécifiées dans [JR02, JR04, JR06].

Résultats des simulations et des tests. Une simulation [JR06] a été réalisée grâce à OMNet++ [Var01]. Elle a fait intervenir 50 nœuds sur une surface rectangulaire de 3000m × 600m. Le modèle de mobilité utilisé était la marche aléatoire. Les résultats portent notamment sur la latence moyenne d'exécution des primitives sur le *Tuple Space* et sur le lien entre la taille de la vue définie par l'agent et la latence d'exécution des primitives.

État actuel. La dernière publication qui traite de EgoSpaces date de 2006 [JR06]. La dernière version du code disponible en ligne date quant à elle précisément du 5 janvier 2006.

2.2.2.8 EMMA

Contexte de réalisation. EMMA (*Epidemic Messaging Middleware for Ad hoc networks*) [MMH05] a été développé par Mirco Musolesi[11], Cecilia Mascolo[11,12] et Stephen Hailes en 2006 à l'Université *College of London* au Royaume-Uni.

Objectif. EMMA [MMH05] est une adaptation de JMS (*Java Message Service*) [HBSF] pour les réseaux mobiles ad hoc. Son objectif est de proposer un système de propagation d'information pour gérer le cas où une communication synchrone n'est pas possible entre deux nœuds du réseau.

[11] aussi présent(e) dans XMIDDLE (cf. section 2.2.2.26 page 55)
[12] aussi présente dans CARISMA (cf. section 2.2.2.5 page 31)

Caractéristiques techniques (et suppositions implicites). EMMA est un *middleware* orienté message (MOM) qui a été créé pour les MANets qui peuvent être DTN. EMMA est une adaptation de JMS (*Java Message Service*) défini par la JSR 914. JMS est un ensemble d'interfaces permettant de réaliser des communications asynchrones entre des composants ou des applications distribuées. Deux types de modèles sont supportés, le point-à-point et le *publish-subscribe*. Les auteurs d'EMMA supposent dans [MMH05] qu'un réseau mobile ad hoc ne peut être efficace sans technique de routage entre les nœuds. EMMA n'en propose pas, il repose simplement sur la présence d'un de ces protocoles.

Des tests ont été effectués sur la technologie WaveLAN qui est compatible avec IEEE 802.11. Nous n'avons pas pu déterminer si EMMA peut aussi utiliser d'autres technologies de communication sans fil comme par exemple Bluetooth. EMMA peut être exécuté sur des PC, PDA, téléphones portables et tout dispositif permettant l'exécution/l'interprétation de code Java Embarqué et a été déployé sur des PDA HP iPAQ sous Linux et sur des PC portables. Il fonctionne grâce à une machine virtuelle J2ME et est donc portable sur tous les systèmes d'exploitation sur lesquels il existe une machine virtuelle de ce type.

Résultats des simulations et des tests.

Paramètres de simulation. Afin de tester le protocole épidémique mis en place dans EMMA, des simulations ont été effectuées dans le simulateur OMNET++ [Var01]. Les auteurs étudient trois scénarii dans lesquels le réseau est composé tour à tour de 16, 24 puis 32 nœuds mobiles répartis aléatoirement sur une surface de 1 km^2. Les nœuds possèdent une portée circulaire de 200 mètres. L'objectif de ces tests est d'évaluer la performance du système en terme de taux de délivrance des messages. Lors de la simulation (qui dure au plus 4 minutes) chaque nœud envoie 200 messages.

Résultats. Parmi les résultats obtenus, les auteurs montrent la corrélation entre le taux de distribution des messages et la taille du cache de chaque dispositif.

État actuel. L'unique publication que nous avons pu étudier qui traite d'EMMA est [MMH05]. Aucun code ne semble disponible en ligne.

2.2.2.9 Expeerience

Contexte de réalisation. Expeerience [TBCM03, BCMT04, BMT04] a été développé par Mario Bisignano[13], Andrea Calvagna, Giuseppe Di Modica[13] et Orazio Tomarchio[13] de 2003 à 2004 au *Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, Catania* en Italie.

Objectif. Expeerience [TBCM03, BCMT04, BMT04] est un *middleware* développé en Java au-dessus d'une version de JXTA [JXT01] adaptée au contexte des réseaux mobiles

[13] aussi présent dans JMobiPeer (cf. section 2.2.2.12)

ad hoc. Son objectif est de fournir un ensemble d'outils de haut niveau pour faciliter le développement d'applications compatibles avec la version standard de JXTA. Il masque à l'application l'hétérogénéité des dispositifs et des technologies de communication sous-jacentes (qu'elle soient avec ou sans fil). Le *middleware* permet à ces applications d'être mobiles, c'est à dire de se déplacer de dispositif en dispositif de manière dynamique selon les besoins. Les auteurs les appellent des *mobile code services*.

Caractéristiques techniques (et suppositions implicites). Expeerience est un *middleware* orienté message (MOM) conçu pour les réseaux non déconnectés dans lesquels chaque dispositif est mobile. Les cas où un nœud devient injoignable et où le réseau se partitionne ne sont pas considérés.

Expeerience est basé sur JXTA qui utilise le protocole ERP (*Endpoint Routing Protocol*) [OG02] aussi appelé PEP (*Peer Endpoint Protocol*) [OG02] pour établir des routes entre les nœuds.

Il n'est pas précisé quelles sont les technologies de communication considérées. En effet, Expeerience laisse la gestion de celles-ci à JXTA [JXT01].

Par ailleurs, Expeerience a été créé pour fonctionner sur des dispositifs allant des PC aux téléphones portables en passant par les PDA. Il fonctionne sur tout terminal qui possède une machine virtuelle Java ou J2ME.

Sécurité. Il n'est pas question de sécurité dans les articles traitant d'Expeerience. En revanche, JXTA fournit une API de sécurité dans sa version 2.0. Étant donné que les auteurs ont adapté le noyau de JXTA, la version utilisée (qui est donc une nouvelle branche de développement de JXTA) n'a pas suivi le même chemin que la version de Sun/Oracle. Il semble [BCMT04] que les mécanismes de sécurité devaient faire parti des futures investigations.

État actuel. Le dernier article traitant d'Expeerience est [BCMT04] publié en 2004. Il n'existe pas de code disponible en ligne. Il semble que ce *middleware* soit devenu JMobiPeer (cf. section 2.2.2.12 page 40). En effet, ils possèdent les mêmes auteurs et [BMT04] est quasiment similaire à [BMT05] qui définit JMobiPeer.

2.2.2.10 GecGo

Contexte de réalisation. GecGo[14] (*Geographic Gizmos*) [SFGL04, SFFS04] a été développé par Peter Sturm[15], Hannes Frey[15], Daniel Görgen[15], Johannes K. Lehnert[15], Volker Fusenig et Thomas Scherer en 2004 à l'Université de Trier en Allemagne.

Objectif. GecGo [SFGL04, SFFS04] a pour objectif de fournir les services requis par les applications *self-organized* [Ash62] qui fonctionnent sur les réseaux ad hoc. Son paradigme

[14] `http://www.syssoft.uni-trier.de/soul`
[15] aussi présent dans SELMA (cf. section 2.2.2.23 page 51)

de communication est celui que l'on pourrait nommer *"en passant"*, c'est à dire dans lequel les terminaux mobiles peuvent interagir pendant un court instant. GecGo est le successeur de SELMA (cf. section 2.2.2.23).

Caractéristiques techniques (et suppositions implicites). GecGo est un *middleware* orienté agents mobiles qui a été créé pour les réseaux fortement mobiles dans lesquels les nœuds ne peuvent communiquer que pendant un court instant : aucun lien de communication n'est supposé stable.

Les auteurs supposent que chaque nœud possède une identité unique.

GecGo utilise du *broadcast* sur le protocole UDP/IP pour la transmission des *beacons* qui permettent aux nœuds de se découvrir mutuellement, et utilise une connexion TCP/IP pour transférer un agent mobile (aussi appelé *gizmo*). Il est écrit en C#.

Résultats des simulations et des tests. Les tests concernant GecGo ont été réalisés sur des PDA iPAQ (processeur 400 Mhz) et sur des PC équipés de processeur Intel Pentium 4 cadencé à 1900MHz. Ils sont présentés dans [SFFS04] et leur objectif est de mesurer le coût des changements de contexte sur un système Microsoft Windows et le nombre de *gizmos* (agents mobiles) transférés par seconde en fonction de la taille de ceux-ci. Les résultats présentés sont ceux d'une version non optimisée de GecGo. Ils montrent notamment le temps nécessaire aux transferts des *gizmos* en fonction de leur taille afin de déterminer celle qu'il ne doivent pas dépasser pour pouvoir être échangés lors de communications "en passant". Par exemple, si deux terminaux mobiles peuvent communiquer pendant 50 secondes consécutives (ils sont séparés de 50 mètres, possèdent une portée de 50 mètres et se croisent à la vitesse constante de 1 mètre par seconde), alors ils peuvent échanger en moyenne 100 *gizmos* de 512 octets.

État actuel. GecGo fait parti du projet SOUL[15]. Les deux publications traitant de ce *middleware* sont [SFGL04, SFFS04] et datent de 2004. Ce projet ne semble plus actif et aucun code n'est disponible en ligne.

2.2.2.11 JASON

Contexte de réalisation. JASON (*Java Ad hoc Services on ad hOc Networks*) [SR04] a été développé par Nicolas Le Sommer et Hervé Roussain en 2004 dans le groupe de recherche CASA du laboratoire Valoria à l'Université de Bretagne Sud (France).

Objectif. JASON [SR04] a pour objectif de permettre l'échange de document de manière efficace. Pour cela, les terminaux mobiles annoncent à leurs voisins quels sont les programmes et services qu'ils fournissent et réciproquement ils peuvent découvrir quels sont les programmes/services que leurs voisins proposent. JASON permet aussi de récupérer les applications que possèdent les voisins d'un nœud, afin de les exécuter localement sur la plateforme JAMUS (*Java Accomodation of Mobile Untrusted Software*) [LSG02].

Caractéristiques techniques (et suppositions implicites). JASON est un *middleware* orienté message écrit en Java. Aucun routage n'est effectué entre les nœuds et seules les relations de voisinage à un saut sont considérées. Les interfaces de communication gérées ne sont pas spécifiées dans le seul article traitant de JASON [SR04].

Sécurité. JASON permet (grâce à la plateforme JAMUS) d'exécuter des applications dans un environnement sécurisé [LSG02]. Cependant, JASON ne fournit aucune sécurité en ce qui concerne les communications entre les nœuds.

État actuel. L'unique article traitant de JASON est [SR04], publié en 2004. Le code source de JASON n'est pas disponible en ligne.

2.2.2.12 JMobiPeer

Contexte de réalisation. JMobiPeer [BMT05] a été développé par Mario Bisignano[16], Giuseppe Di Modica[16] et Orazio Tomarchio[16] en 2005 au *Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, Catania* en Italie.

Objectif. JMobiPeer [BMT05] a avant tout été créé pour la résolution d'un problème technique, à savoir rendre possible des interactions entre les nœuds qui utilisent JXTA et des terminaux mobiles. Il est très proche d'Expeerience (cf. section 2.2.2.9 page 37) en ce qui concerne ses objectifs. En effet, il a été créé pour (i) être compatible avec les protocoles de JXTA, (ii) être utilisable dans un environnement MANet même lorsqu'il n'y a pas de dispositif utilisant JXTA à portée, (iii) pouvoir fonctionner sur les dispositifs à ressources limitées tels que ceux qui possèdent uniquement une machine virtuelle Java embarquée (J2ME), et (iv) repousser les limites et les contraintes induites par l'utilisation de JXME (cf. section 2.2.2.13 page 41).

Caractéristiques techniques (et suppositions implicites). JMobiPeer est un *middleware* orienté message (MOM). Il est utilisable est les réseaux dans lesquels aucun nœud n'est isolé, c'est à dire dans un contexte dans lequel le réseau sous-jacent ne peut pas être déconnecté (DTN).

Les interfaces de communication sans fil gérées ne sont pas spécifiées. Il n'est pas non plus précisé dans l'unique article qui traite de JMobiPeer [BMT05] si des dispositifs mobiles à ressources limitées peuvent être utilisés par celui-ci, il doit cependant pouvoir fonctionner sur tout dispositif qui permet d'exécuter une application Java embarquée (J2ME).

Résultats des simulations et des tests.

Paramètres de simulation. Une simulation a été effectuée par les auteur de JMobiPeer sur deux PC Pentium 4 à 2.4 GHz avec 512 Mo de RAM reliés en Ethernet 100 Mbps. Afin

[16] aussi présent(e) dans Expeerience (cf. section 2.2.2.9 page 37)

de pouvoir exécuter JMobiPeer, le JWTK (*SUN J2ME Wireless Toolkit 2.1*) incluant le simulateur CLDC (*Connected Limited Device Configuration*) et le paquetage SUN J2ME (*Java 2 Micro Edition*) pour les CDC (*Connected Device Configuration*) ont été installés en plus du JDK (*Java Development Toolkit*) en version 1.4.2.

Scénario. Le test présenté dans [BMT05] permet de mesurer le temps d'une découverte de services. Le scénario est le suivant. Supposons qu'un nœud nommé C souhaite découvrir une annonce R localement publiée par un autre nœud nommé S. La phase de découverte comprend les trois étapes suivantes. Tout d'abord, C diffuse une requête de découverte. Ensuite, lorsque S reçoit cette requête, il l'interprète, et envoie à C une réponse contenant R. Enfin, C reçoit la réponse de S, la stocke, et la fournit éventuellement aux services des couches supérieures.

Résultats. Les résultat obtenus par JMobiPeer sont comparés à ceux de JXME (cf. section 2.2.2.13 page 41). Le temps de découverte des dispositifs est en moyenne de 2339 ms pour JMobiPeer-CLDC, de 209 ms pour JXME-proxiless et 163 ms pour JMobiPeer-CDC. La taille de la requête de découverte de l'annonce R est d'environ 1450 octets avec JMobiPeer contre environ 730 pour JXME-proxiless. La taille de la réponse à cette découverte est d'environ 1580 octets pour JMobiPeer contre environ 640 octets pour JXME-proxiless. Les auteurs de JMobiPeer concluent que le surcoût induit par l'utilisation de structures XML pour les messages est relativement faible en comparaison de l'apport de l'interopérabilité entre JMobiPeer et les passerelles JXTA.

État actuel. A notre connaissance, il n'existe qu'un article [BMT05] qui définit JMobiPeer. Le code de ce dernier ne semble pas disponible en ligne.

2.2.2.13 JXME

Contexte de réalisation. JXME[17] (JXTA for J2ME) [AHP02] a été développé par Akhil Arora, Carl Haywood et Kuldip Singh Pablade en 2002 dans le laboratoire de *Sun Microsystems, Inc. Palo Alto, CA* aux États-Unis.

Objectif. JXME [AHP02] est une adaptation de JXTA[JXT01] pour les dispositifs à faibles ressources. Tout comme JMobiPeer (cf. section 2.2.2.12 page 40), JXME a avant tout été créé pour rendre possible des interactions entre les nœuds qui utilisent JXTA et des terminaux mobiles. Il est basé sur CLDC (*Connected Limited Device Configuration*) ou CDC (*Connected Device Configuration*) et sur le profil MIDP (*Mobile Information Device Profile*). Les objectifs principaux de JXME sont (i) d'être interopérable avec les PC équipés de JXTA, (ii) de fournir une infrastructure P2P pour les dispositifs à faibles ressources, (iii) d'être facile à utiliser pour les développeurs, (iv) d'être suffisamment léger pour être utilisable sur des téléphones portables et des PDA, et (v) d'être compatible avec MIDP-2.0 (Mobile Information Device Profile).

[17] `https://jxta-jxme.dev.java.net/`

Il existe deux versions de JXME, une version avec *proxy* et une sans *proxy*. La version avec *proxy* permet à deux dispositifs qui disposent de JXME et qui sont à portée d'un *proxy* JXTA, de communiquer avec les différents nœuds du réseau, qu'ils utilisent JXME ou JXTA. Dans cette version, il n'est pas possible pour deux nœuds JXME du réseau de communiquer directement. La version sans *proxy* autorise une communication directe entre deux dispositifs s'ils sont tous les deux équipés de cette version. Cependant, un *proxy* reste nécessaire pour communiquer avec les nœuds JXTA du réseau.

Caractéristiques techniques (et suppositions implicites). JXME est un *middleware* orienté message (MOM). La mobilité sous-jacente est implicitement supposée faible car les nœuds JXME doivent être à portée d'une passerelle JXTA (*JXTA Relay*) pour pouvoir communiquer les uns avec les autres. Le rôle de cette passerelle est (i) de faire le lien avec les réseaux infrastructurés qui utilisent JXTA, et (ii) de permettre les communications entre deux dispositifs mobiles qui utilisent JXME. Il n'y a donc pas de communication pair-à-pair entre tous les dispositifs du réseau. JXME n'a pas été créé pour les réseaux mobiles ad hoc purs mais pour permettre aux dispositifs mobiles d'interagir avec les réseaux infrastructurés.

Il existe cependant une autre version dite *proxyless* qui permet à deux dispositifs équipés de JXME de communiquer directement. Cette version ne permet alors plus au nœuds de communiquer avec les passerelles JXTA. C'est pour palier ce problème que JMobiPeer a été conçu (cf. section 2.2.2.12 page 40.)

JXME gère les communications sur IP (quelle que soit la technologie de communication sous-jacente) et Bluetooth. Il fonctionne sur tous les dispositifs qui disposent d'une machine virtuelle J2ME et qui sont compatibles avec CLDC (*Connected Limited Device Configuration*), MIDP (*Mobile Information Device Profile*) ou CDC (*Connected Device Configuration*).

État actuel. La dernière version est la version 2.1.3 qui date du 20 juin 2006. Il semble que ce projet ne soit plus actif.

2.2.2.14 LIME

Contexte de réalisation. LIME[18] (*Linda In Mobile Environment*) [PMR99, Mur00, PMR00, PMR01, PB02, HR02, MPR03, HPJR03, MP04, MPR06] a été développé par Amy L. Murphy, Gian Pietro Picco[19], Gruia-Catalin Roman[20] de 1999 à 2002 (publications de 1999 à 2007) au *Department of Computer Science, Washington University, St. Louis, Missouri* aux États-Unis.

Objectif. LIME [PMR99, Mur00, PMR00, PMR01, PB02, HR02, MPR03, HPJR03, MP04, MPR06] a pour objectif le déploiement rapide d'applications pour les réseaux mobiles ad hoc tout en proposant une validation formelle de celles-ci grâce à un formalisme

[18] `http://lime.sourceforge.net/`
[19] aussi présent(e) dans PeerWare (cf. section 2.2.2.20 page 48)
[20] aussi présent(e) dans EgoSpaces (cf. section 2.2.2.7 page 35) et Limone (cf. section 2.2.2.15 page 43)

nommé Mobile Unity [MR98].

Caractéristiques techniques (et suppositions implicites). LIME est un *middleware* orienté *Tuple Space* et agents mobiles, écrit en Java et qui reprend le paradigme de Linda en l'adaptant aux réseaux mobiles ad hoc. Ce *middleware* supporte la mobilité physique des dispositifs et la mobilité logique des agents mobiles. Chaque nœud possède un *Tuple Space* local. Lorsque deux agents mobiles sont connectés, leurs *Tuple Spaces* fusionnent pour former un *Transiently Shared Tuple Space*. Initialement, l'utilisation de LIME supposait que les déconnexions entre les nœuds pouvaient être anticipées. Cette hypothèse n'est pas réaliste dans un contexte de réseau mobile, c'est pourquoi cette hypothèse a par la suite été supprimée. Lors d'une déconnexion non anticipée, afin de conserver le *Tuple Space* local dans un état cohérent, les *Tuples* qui étaient en court de transfert d'un agent mobile vers un autre agent mobile sont *taggués* sur chaque agent comme étant dans un état incohérent (*inconsistent state*). Lorsque les deux agents mobiles qui possèdent ces *Tuple Spaces* sont de nouveau connectés, un protocole permettant de rétablir la cohérence de leurs *Tuples* est exécuté.

Les agents mobiles se déplacent de nœud en nœud en utilisant une communication de type flux (TCP). Les liens de communication entre les nœuds sont donc supposés symétriques, une connexion TCP ne pouvant exister dans un cadre asymétrique. Ce middleware ne permet donc pas de tirer parti des liens de communication asymétriques qui peuvent exister entre les nœuds qui composent un MANet (cf. annexe A page 163). De plus, le *Tuple Space* global est la fusion de l'ensemble des *Tuple Spaces* locaux des nœuds qui appartiennent à la même composante connexe. La relation d'appartenance à une composante connexe est une relation transitive. La relation de voisinage est donc elle aussi transitive.

État actuel. LIME est un projet très actif qui a fait l'objet de nombreuses publications de la part des auteurs de ce middleware ces dix dernières années. Le code qui est disponible en ligne sur `lime.sourceforge.net` date cependant de 2002.

2.2.2.15 Limone

Contexte de réalisation. Limone[21] [lFcRH04] a été développé par Chien-Liang Fok, Guia-Catalin Roman[22] et Gregory Hackmann de 2003 à 2004 au *Department of Computer Science and Engineering Washington University in Saint Louis, Missouri* aux États-Unis.

Objectif. Limone [lFcRH04] est un *middleware* dont l'objectif est la gestion de la mobilité logique d'agents mobiles et de la mobilité physique de dispositifs. L'objectif de Limone est de proposer un modèle de coordination orienté *Tuple Space* en faisant le moins de supposition possible sur le réseau sous-jacent.

[21] `http://www.cs.wustl.edu/mobilab/projects/limone/`

[22] aussi présent(e) dans EgoSpaces (cf. section 2.2.2.7 page 35) et LIME (cf. section 2.2.2.14 page 42)

Caractéristiques techniques (et suppositions implicites). Limone est un *middleware* orienté *Tuple Space* et agents mobiles. Lors de la création de Limone, ses auteurs ont cherché à minimiser le nombre de suppositions faites sur l'environnement. Il se place dans un contexte où la mobilité des nœuds est plus importante que celle gérée par LIME (cf. section 2.2.2.14) car il propose des primitives qui reprennent le paradigme de Linda mais en supprimant les opérations complexes de groupe qui nécessitent une synchronisation de l'ensemble des agents mobiles. Une différence majeure avec les autres systèmes à base de *Tuple Spaces* est qu'il permet des interactions asymétriques entre les agents. De plus, l'utilisation de Limone garantit que toutes les opérations distribuées contiennent un mécanisme permettant d'éviter des *deadlocks* dus à des paquets perdus ou à une déconnexion inattendue. Pour ces raisons, Limone gère la perte de paquets et une mobilité des nœuds plus importante que celle-ci prise en charge par LIME (cf. section précédente). Dans [lFcRH04], les auteurs mettent en évidence les suppositions fortes sur le réseau sous-jacent faites dans LIME.

La relation de voisinage entre les nœuds est supposée symétrique car un agent mobile ne se duplique pas, il se déplace. En effet, comme expliqué dans la section 2.2.1.3 page 25, une condition nécessaire au déplacement d'un agent mobile est une synchronisation préalable entre le nœud hôte de l'agent et le nœud cible. Lors du déplacement d'un agent mobile, une absence de synchronisation entre les nœuds ne permet pas de garantir (i) sa non duplication, et (ii) sa non disparition. L'hôte de l'agent mobile doit donc nécessairement savoir qu'un autre nœud est à sa portée. La mobilité des agents mobiles repose donc uniquement sur les relations de voisinage symétriques.

Par ailleurs, Limone fonctionne au dessus d'une couche IP, il supporte donc toutes les technologies de communication capables de fournir une couche IP. Des tests ont été réalisés sur des PC portables à 750 MHz équipés de cartes Wi-Fi mais il n'est pas spécifié si d'autres matériels sont supportés.

Gestion d'énergie. La gestion d'énergie est abordée dans [lFcRH04]. Les auteurs indiquent simplement qu'un *middleware* pour réseaux mobiles ad hoc doit prendre en compte ce paramètre. Cependant, ils ne donnent pas de résultat concernant ce point.

Résultats des simulations et des tests. Un test de performance est mis en avant dans [lFcRH04]. Il compare le temps nécessaire à un *Tuple* pour effectuer un aller-retour entre deux agents qui se trouvent sur deux hôtes différents et la taille du code nécessaire pour écrire ce test en utilisant LIME, Limone ou simplement des sockets. Les résultats obtenus sont présentés dans le tableau 2.2.

Modèle	Lignes de code	Temps (en ms)
Limone	250	50,3
LIME	170	73,6
Sockets	695	44,6

TAB. 2.2: Comparaison des résultats obtenus par Limone, LIME et des sockets extraits de [lFcRH04]

État actuel. La dernière version de Limone disponible en ligne date du 15 novembre 2004. Ce projet ne semble donc plus actif.

2.2.2.16 MESHMdl

Contexte de réalisation. MESHMdl (*MESH Enables Self-Organizing Hosts*) [Her03] a été développé par Klaus Herrmann en 2003 à l'Université de Berlin en Allemagne.

Objectif. MESHMdl [Her03], prononcé *mesh middle*, est un *middleware* dont l'objectif est fournir des mécanismes d'auto-organisation pour les réseaux mobiles ad hoc. Afin de définir l'auto-organisation, les auteurs de MESHMdl utilise la définition suivante (de Scott Camazine et al. extraite de [CFS+01]). L'auto-organisation est un processus dans lequel une structure globale peut émerger grâce à des interactions locales entre les composants d'un système. De plus, les règles spécifiant les interactions entre les composants du système sont exécutées en utilisant seulement l'information locale, sans référence à la structure globale. Il repose sur des agents mobiles et un paradigme de *Tuple Space*. Dans ce système, les agents mobiles ont la capacité de se déplacer de nœud en nœud et de lire et écrire dans le *Tuple Space* de chaque nœud.

Caractéristiques techniques (et suppositions implicites). MESHMdl est un *middleware* orienté *Tuple Space* et agents mobiles. MESHMdl a été conçu pour les réseaux mobiles ad hoc qui peuvent se partitionner (DTN).

MESHMdl offre une couche nommée *Generic Connection Layer* qui fournit des primitives permettant aux couches supérieures de faire abstraction de la technologie radio utilisée. Il utilise les technologies Bluetooth et WLAN et a été conçu pour supporter les dispositifs mobiles à faible capacité tels que les PDA et les téléphones mobiles. Il peut aussi fonctionner sur des dispositifs plus performants tels que des PC portables ou des PC de bureau. Il utilise une machine virtuelle J2ME (*Java 2 Micro Edition*) qui dispose de CLDC (*Connected Limited Device Configuration*) de Sun Microsystems.

État actuel. Le dernier article traitant de MESHMdl est [Her03], publié en 2003. Il semble donc que MESHMdl ne soit plus un projet actif.

2.2.2.17 Mobile Chedar

Contexte de réalisation. Mobile Chedar [KWVV05] a été développé par N. Kotilainen, M. Weber, M. Vapa, J. Vuori en 2005 à l'Université de Jyvaskyla en Finlande.

Objectif. Mobile Chedar [KWVV05] est un *middleware* pour dispositifs mobiles. C'est une extension du *middleware* nommé Chedar (*CHEap Distributed ARchitecture*) [AVW+06] pour les terminaux mobiles. L'objectif de Chedar est de faciliter les interactions pair-à-pair dans un réseau statique. Mobile Chedar permet aux dispositifs mobiles de communiquer

avec les pairs déjà existants du réseau Chedar. Ces derniers jouent alors le rôle de passerelle entre les terminaux mobiles et les autres pairs du réseau Chedar.

Caractéristiques techniques (et suppositions implicites). Mobile Chedar est un *middleware* orienté message (MOM). La mobilité sous-jacente des nœuds supportée n'est pas spécifiée dans [KWVV05]. Les nœuds mobiles utilisent des passerelles pour accéder au réseau infrastructuré pair-à-pair Chedar.

Mobile Chedar a été écrit en J2ME (*Java 2 Micro Edition*) et repose uniquement sur la technologie Bluetooth pour les communications entre les nœuds mobiles et les passerelles.

État actuel. La dernière publication qui concerne ce *middleware* date de 2005. Le projet ne semble plus actif.

2.2.2.18 Mobile Gaia

Contexte de réalisation. Mobile Gaia[23] [CAMCM04, CAMCM05] a été développé par Shevan Chetan, Jalal Al-Muhtadi et Roy Campbell en 2005 à l'université de l'Illinois, Urbana-Champaign aux États-Unis.

Objectif. Mobile Gaia [CAMCM04, CAMCM05] est un *middleware* pour les réseaux ad hoc omniprésents c'est-à-dire, selon la définition des auteurs, un *cluster* de dispositifs personnels qui peuvent communiquer et partager des ressources les uns avec les autres (la mobilité ne fait donc pas partie de la définition). Ce cluster est nommé espace actif (*Active Space*). Chaque espace actif est composé d'un coordinateur et de clients. L'objectif principal de Mobile Gaia est de fournir une architecture orientée service permettant la distribution de ceux-ci aux différents dispositifs définissant l'espace actif. Ce *middleware* est aussi *location-aware*, c'est à dire qu'il fournit un service permettant la localisation des dispositifs. Il utilise pour cela une puce GPS pour obtenir sa position géographique en extérieur. Si le dispositif ne parvient à capter les signaux GPS (parce qu'il se trouve en intérieur par exemple), le middleware tentent d'obtenir la position du dispositif sur lequel il s'exécute grâce aux technologies sans fil, par exemple en s'appuyant sur les points d'accès Wi-Fi disponibles (ces derniers indiquant leur position).

Caractéristiques techniques (et suppositions implicites). Mobile Gaia est un *middleware* orienté message (MOM). Par ailleurs, chaque *Active Space* a besoin d'un coordinateur pour exister. Il ne suit donc pas le paradigme des réseaux ad hoc dans lesquels tous les nœuds sont des pairs (le coordinateur de l'espace est un point de centralisation).

Mobile Gaia utilise les technologies de communication Bluetooth, IrDA, Wi-Fi et Ethernet. Tous les dispositifs fixes et mobiles, du PC de bureau aux téléphones portables en passant par les PDA sont supportés. Les systèmes d'exploitation sur lesquels il fonctionne ne sont cependant pas précisés dans les articles étudiés qui traitent de Mobile Gaia.

[23] `http://choices.cs.uiuc.edu/~chetan/proj.htm`

Sécurité. Mobile Gaia utilise un système à base de clefs asymétriques pour l'authentification des dispositifs et le chiffrement des communications.

État actuel. Mobile Gaia a fait l'objet de deux publications, [CAMCM04] en 2004 et [CAMCM05] en 2005. Le projet ne semble plus actif et aucun code n'est disponible en ligne.

2.2.2.19 MoCoA

Contexte de réalisation. MoCoA[24] [SCB+06] a été développé par Aline Senart, Raymond Cunningham, Mélanie Bouroche, Neil O'Connor, Vinny Reynolds, Vinny Cahill[25] en 2006 au *Department of Computer Science, Trinity College Dublin* en Irlande.

Objectif. MoCoA [SCB+06] est un *middleware context-aware* dont l'objectif est de permettre le développement rapide d'applications elles-mêmes sensibles au contexte.

Caractéristiques techniques (et suppositions implicites). MoCoA est un *middleware* orienté message (MOM). Il a été créé pour les réseaux mobiles ad hoc dont le graphe sous-jacent peut être déconnecté. Tous les nœuds ne sont pas forcément des pairs mais il n'existe pas de point de centralisation. Les technologies de communication gérées, le matériel et les systèmes d'exploitation supportés ne sont pas explicitement définies dans l'article qui traite de MoCoA que nous avons étudié.

Les applications construites pour MoCoA sont structurées grâce aux *Sentient Objects*. Les *Sentient Objects* sont des entités mobiles intelligentes qui extraient, interprètent et utilisent les informations obtenues grâce aux *Sensors*. Dans ce modèle, un *Sensor* est vu comme une entité qui fournit des évènements à l'application en fonction des stimuli du monde réel. Bien que le concept de communication évènementielle soit simple, certaines difficultés peuvent survenir lors du passage à l'échelle. Afin de palier ces difficultés, MoCoA associe aux évènements (i) la localisation de la source de celui-ci et (ii) la définition de garanties temporelles sur leurs livraisons. Lors d'un changement sur le réseau sous-jacent dû à une déconnexion d'un nœud par exemple, il permet la notification au producteur d'évènements d'un changement dans le délai de livraison de ses évènements afin que celui-ci adapte son comportement et le comportement de l'application, et cela, même dans un réseau partitionné. Par exemple, même si la communication entre un ou plusieurs véhicules autonomes et les feux de signalisation n'est pas possible, il est toujours possible de garantir qu'aucun véhicule ne passera au feu rouge, les feux de signalisation adaptant leur comportement lorsqu'ils constatent que certains messages ne sont pas délivrés. Les évènements peuvent être combinés afin d'en déduire des faits contextuels (qui sont des abstractions de haut niveau relatives au contexte). Un contexte est défini comme l'ensemble des informations disponibles sur l'environnement qui peuvent être utilisées pour caractériser la situation dans laquelle se trouve une entité. Il est associé à un ensemble de règles qui permettent de définir le comportement de l'application.

[24] `http://www.dsg.cs.tcd.ie/node/54`
[25] aussi présent dans STEAM (cf. 2.2.2.24 page 52)

État actuel. MoCoA a été créé dans le cadre du projet Aithne [Ait]. La dernière publication dans le cadre de ce projet date de 2010, il est donc toujours actif. Le code source de MoCoA ne semble pas disponible en ligne.

2.2.2.20 PeerWare

Contexte de réalisation. PeerWare[26] [CP01] a été développé par Gianpaolo Cugola et Gian Pietro Picco[27] en 2001 au *Dipartimento di Electronica e Informazione, Politecnico di Milano* à Milan en Italie.

Objectif. PeerWare [CP01] est un *middleware* dont l'objectif est de fournir un support d'exécution pour les applications conçues pour les réseaux mobiles ad hoc dans lesquels les échanges se font en mode pair-à-pair. Il s'inspire des travaux sur LIME (cf. section 2.2.2.14). Il utilise à la fois un paradigme de type *Tuple Space* pour la synchronisation des nœuds et de type publish/subscribe (donc message) pour rendre possible des interactions évènementielles entre les nœuds.

Caractéristiques techniques (et suppositions implicites). PeerWare est un *middleware* écrit en Java à la fois orienté message (MOM) de type publish/subscribe et orienté *Tuple Space*. Les auteurs argumentent que le paradigme publish/subscribe est approprié pour la mise en place d'applications mobiles distribuées car ce type de système est purement basé sur un paradigme de message. Cependant, en utilisant un système purement asynchrone, avec uniquement des primitives de communication *stateless*, un *middleware* publish/subscribe supporte difficilement des primitives de coordination et de synchronisation. Au contraire, le paradigme de *Tuple Space*, centré sur un espace de données persistant et toujours accessible supporte facilement les primitives de coordination *state-based* et de synchronisation. PeerWare consiste en la fusion de ces deux paradigmes dans un unique modèle de coordination qui lie la notification d'évènement et la notion d'espace de données partagé dans des *Global Virtual Data Structures*.

La mobilité sous-jacente des nœuds supportée par PeerWare n'est pas précisément définie. Le prototype de PeerWare disponible en ligne fonctionne sur IP. Les types de matériels supportés ne sont pas spécifiés dans l'article qui traite de PeerWare que nous avons étudié.

Par ailleurs, comme ce *middleware* repose sur un paradigme à base de *Tuple Space*, la relation de voisinage entre les nœuds est supposée symétrique. Les applications qui l'utilisent ne peuvent donc pas tirer parti des liens asymétriques qui peuvent exister dans le réseau.

État actuel. L'unique publication qui traite de PeerWare est [CP01] ; elle date de 2001. Le code source de ce *middleware* est disponible en ligne et la dernière version publiée date du 3 janvier 2003.

[26] `http://peerware.sourceforge.net/`

[27] aussi présent(e) dans LIME (cf. section 2.2.2.14 page 42)

2.2.2.21 Proem

Contexte de réalisation. Proem[28] [KSS+01] a été développé par Gerd Kortuem, Jay Schneider, Dustin Preuitt, Thaddeus G. C. Thompson, Stephen Fickas et Zary Segall de 1995 à 2001 au *Department of Computer Science, University of Oregon, Eugene, Oregon* aux États-Unis.

Objectif. Proem [KSS+01] a pour objectif de permettre aux individus qui se rencontrent physiquement d'échanger de manière sécurisée des informations ou des fichiers. Il permet le développement et le déploiement d'applications pour les réseaux mobiles ad hoc.

Caractéristiques techniques (et suppositions implicites). Proem est un *framework* écrit en Java. Il est composé d'un *kit* de développement (un ensemble d'interfaces et de classes Java permettant le développement rapide d'applications pair-à-pair mobiles appelées *peerlets*) et d'un *middleware* orienté message (MOM). Proem se place dans le contexte des Personal Area Network (PAN) qui peuvent être constitués de dispositifs appelés *wearable devices* [BS99]. Proem gère les technologies de communication des *wearable devices* que sont Bluetooth, la norme 802.15 de l'IEEE [IEE], Genuity's BodyLAN [Bar96] et Zimmermann intra-body [Zim96].

Sécurité. Les auteurs de Proem s'intéressent à la sécurité et au respect de la vie privée. Ce dernier aspect est original mais aucun mécanisme particulier ne semble cependant avoir été intégré. Ils mettent en avant leur intérêt pour la sécurité liée à l'utilisation de la technologie Bluetooth en soulignant toutefois que celle-ci ne peut être mise en place dynamiquement.

Les auteurs proposent dans [KSS+01] d'utiliser un procédé biométrique pour permettre à un dispositif d'authentifier son utilisateur. De plus, dans certains contextes, l'utilisateur peut souhaiter rester anonyme (l'exemple d'un logiciel de partage de fichiers MP3 est alors décrit dans [KSS+01]). Les auteurs supposent que chaque dispositif mobile possède un identifiant unique. La manière dont cette unicité est garantie n'est pas précisée dans l'article qui traite de Proem que nous avons étudié. Le développement de mécanismes de sécurité dans un environnement totalement distribué en utilisant un système à base de clefs publiques fait parti des travaux futurs qui ne semblent pas avoir vu le jour.

État actuel. Le code source n'est pas disponible en ligne à l'extérieur du laboratoire dans lequel Proem a été développé.

2.2.2.22 SCOMET

Contexte de réalisation. SCOMET[29] [SAAPL08] a été développé par Marc Sanchez-Artigas, Marcel Arrufat, Gerard Paris, Pedro Garcia Lopez et Antonio F. Gomez Skarmeta

[28] `http://wearables.cs.uoregon.edu/proem/`
[29] `http://www.ist-popeye.eu/`

de 2006 à 2008 au sein des Universités de Tarragone et de Murcie, en Espagne.

Objectif. SCOMET [SAAPL08] est un *middleware* dont l'objectif est de permettre la mise en place de services collaboratifs sur les réseaux mobiles ad hoc. Il propose un protocole original de type *Application Layer Multicast protocol (ALM)* nommé OMCAST.

Caractéristiques techniques (et suppositions implicites). SCOMET est un *middleware* orienté message (MOM). La mobilité sous-jacente est supposée faible car les nœuds sont immobiles la plupart du temps dans le cas d'utilisation décrit dans [SAAPL08]. L'exemple de mobilité qui est donné dans [SAAPL08] est celui d'une personne assise dans une salle de réunion (donc immobile) qui au bout d'un certain temps, se déplace vers une autre salle de réunion. La communication entre les nœuds repose sur du routage. OLSR a été utilisé pour les tests (cf. ci-après). Les réseaux déconnectés ne sont pas pris en compte : le réseau sous-jacent est connexe à tout instant.

SCOMET utilise le protocole UDP, cependant les technologies de communications sous-jacentes gérées ne sont pas spécifiées. Des tests ont été réalisés sur des PC. Par ailleurs, la configuration requise pour l'exécution de SCOMET et les systèmes d'exploitation supportés ne sont pas précisés dans l'article qui traite de SCOMET que nous avons étudié. SCOMET est écrit en Java, ce qui facilite son déploiement sur des dispositifs hétérogènes en terme de système d'exploitation.

Résultats des simulations et des tests.

Scénario. Supposons trois salles de réunions A, B et C dans lesquelles se trouvent des dispositifs mobiles. Dans chaque salle, le graphe qui représente le réseau est complet. Supposons que certains dispositifs de la salle A peuvent communiquer avec certains dispositifs de la salle B et que certains dispositifs de la salle B peuvent communiquer avec certains dispositifs de la salle C. L'objectif de ce test est de montrer que grâce à SCOMET et à OLSR, les dispositifs de la salle A peuvent communiquer avec les dispositifs de la salle C. Les tests montrent les délais de transmission des messages. De plus, lors de ces tests, deux dispositifs mobiles de la salle A se sont déplacés vers la salle B et la salle C. Nous pouvons constater que la mobilité des nœuds est supposée très faible dans cette simulation. Ils sont la plupart du temps immobiles et seule une petite partie des nœuds qui composent le réseau se déplacent.

Test d'OMCAST. OMCAST est un protocole qui permet le maintien d'un arbre couvrant et ainsi la mise en place de communications en multicast dans un réseau ad hoc. Il s'appuie sur le fait que toutes les communication radio sont de type *broadcast* pour optimiser les interactions entre les nœuds. Dans [SAAPL08], les auteurs donnent les résultats d'une simulation faisant intervenir le protocole OMCAST. Ils étudient notamment la surcharge réseau due à ce protocole ainsi que le taux de livraison des paquets. Ce dernier est toujours supérieur 75%, pour toutes les bandes passantes testées (de 1 à 4 Kbps).

État actuel. SCOMET fait parti du projet IST POPEYE qui s'est terminé le 31 juillet 2008. Le code source relatif au projet POPEYE est disponible en ligne depuis l'URL `http://sourceforge.net/projects/popeye-cwe/`.

2.2.2.23 SELMA

Contexte de réalisation. SELMA[30] (*Self-Organized Makertplace-based Middleware For Mobile Ad hoc Networks*) [FGLS04] a été développé par Daniel Görgen[31], Hannes Frey[31], Johannes K. Lehnert[31] et Peter Sturm[31] en 2004 à l'Université de Trier, en Allemagne.

Objectif. L'objectif de SELMA est de fournir des solutions de communication pour les réseaux ad hoc fortement mobiles.

Caractéristiques techniques (et suppositions implicites). SELMA est un *middleware* à la fois orienté message (MOM) et agents mobiles. Il utilise des messages pour la découverte de voisinage et des agents mobiles pour la communication entre les applications. Il a été créé pour les environnements mobiles fortement dynamiques et ne fait donc pas de supposition sur la mobilité sous-jacente des nœuds.

Les applications communiquent grâce aux agents mobiles. Afin de faciliter la communication dans un réseau éparse, les agents mobiles se déplacent tous jusqu'à la "place du marché" (qui est unique). La "place du marché" est un lieu où la densité des nœuds est la plus importante. La position de celle-ci peut évoluer au cours du temps, en fonction du déplacement des nœuds. SELMA suppose que chaque dispositif est capable de déterminer sa position grâce à une puce GPS par exemple. La connaissance de la position de tous les nœuds permet à chacun d'eux de calculer la position de la place du marché.

SELMA utilise les technologies Wi-Fi et Bluetooth. Les tests qui ont été effectués ont utilisé le Wi-Fi en mode ad hoc. Il n'est pas précisé si l'implémentation existante peut utiliser la technologie Bluetooth.

Aucune implémentation de SELMA pour terminaux mobiles ne semble exister. Il existe en revanche une implémentation sur PC. Les systèmes d'exploitation supportés ne sont pas précisés dans l'article de SELMA que nous avons étudié.

Gestion d'énergie. Les auteurs argumentent que l'utilisation d'une place de marché facilite les échanges entre les applications (diminue les pertes de paquets) grâce à la densité des nœuds présents. Cependant, afin d'utiliser ce système, chaque nœud doit être capable de se positionner dans l'espace. Il n'est pas fait mention du surcoût énergétique dû au maintien de cette information (si les nœuds sont par exemple contraints de calculer en permanence leur position grâce aux signaux GPS).

Résultats des simulations et des tests. Aucun résultat de simulation n'est présenté dans l'article de SELMA que nous avons étudié.

[30] `http://www.syssoft.uni-trier.de/soul`

[31] aussi présent(e) dans GecGo (cf. section 2.2.2.10 page 38)

Les auteurs présentent cependant une application nommée UbiBay. UbiBay est un système de vente aux enchères décentralisé. Chaque utilisateur peut mettre en vente des produits ou acheter des produits mis en vente par d'autres utilisateurs. Cette application, comme toutes les applications écrites pour SELMA, fonctionne grâce à des agents mobiles. Pour l'achat d'un produit, l'utilisateur indique à un agent mobile qu'il créé le produit à acheter et le prix maximum à ne pas dépasser. L'agent mobile se déplace ensuite de proche en proche jusqu'à la place du marché et y reste jusqu'à la fin de l'enchère (sauf si un autre agent mobile surenchérit au delà du prix maximum précisé par l'utilisateur lors de la création de l'agent mobile). L'agent mobile retourne ensuite sur le nœud d'origine afin d'indiquer à l'utilisateur s'il a acheté le produit. Concernant la vente, le procédé est sensiblement le même. L'utilisateur indique à l'agent mobile la description du produit à mettre en vente. L'agent mobile se déplace ensuite jusqu'à la place du marché pour mettre en vente ce produit. Lorsque la vente se termine, il retourne sur le nœud d'origine en indiquant le prix auquel le produit s'est effectivement vendu (s'il a été vendu) et les informations sur l'acquéreur. Une vidéo de la simulation de l'application UbiBay est disponible en ligne à l'adresse `http://www.syssoft.uni-trier.de/soul/download/multimedia/ubibay/`.

État actuel. SELMA fait partie du projet SOUL[32] (cf. section 2.2.2.10 page 38). La seule publication traitant de ce *middleware* est [FGLS04]. Le développement de SELMA ne semble plus actif.

2.2.2.24 STEAM

Contexte de réalisation. STEAM[33] (*Scalable Timed Events And Mobility*) [MC03a, MC03b, MC02] a été développé par René Meier et Vinny Cahill[34] de 2001 à 2004 au *Department of Computer Science, Trinity College, Dublin*, en Irlande. Il s'inscrit dans le projet CORTEX (*CO-operating Real-time senTient objects : architecture and EXperiment evaluation*). Les partenaires de ce projet européen sont l'Université de Lisbonne (Portugal), l'Université de Lancaster (Royaume Uni), le Trinity College (Irlande) et l'Université d'Ulm (Allemagne). Ses partenaires industriels sont IONA Technologies, ParaRede, Hewlett-Packard Laboratories et Daimler Chrysler AG.

Objectif. STEAM [MC03a, MC03b, MC02] est un *middleware event-based* créé pour les réseaux mobiles ad hoc dont l'objectif est de permettre la définition au niveau applicatif de filtres sur les évènements afin de définir leur propagation et ainsi d'optimiser la bande passante.

Caractéristiques techniques (et suppositions implicites). STEAM est un *middleware* orienté message qui ne fait pas de supposition sur la mobilité des nœuds. Il utilise la technologie de communication IEEE 802.11b (Wi-Fi). Dans [MC03a], des expérimentations ont été menées sur trois PC portables, chaque PC exécutant plusieurs versions du

[32] `http://www.syssoft.uni-trier.de/soul`
[33] `http://cortex.di.fc.ul.pt/`
[34] aussi présent(e) dans MoCoA (cf. section 2.2.2.19 page 47)

middleware. Il n'est pas fait mention de leurs configurations. Les systèmes d'exploitation supportés ne sont pas non plus spécifiés.

STEAM propose un système de localisation des évènements lorsque ceux-ci sont créés. La description d'un évènement contient donc aussi une position. STEAM permet à l'application de définir des filtres sur les évènements (par exemple sur leurs positions) afin de permettre au *middleware* de déterminer le comportement à adopter lors de la réception de ceux-ci. Ces évènements peuvent être remontés au niveau applicatif et/ou diffusés aux voisins.

Résultats des simulations et des tests. Le scénario d'évaluation de STEAM consiste à modéliser l'intersection entre les routes *North Circular Road* et *Prussia Street* qui se trouvent à Dublin. La simulation est basée sur de vraies données, fournies par la mairie de Dublin, et qui décrivent les mouvements des véhicules à cette intersection et la couleur des feux sur 24 heures consécutives. Cette simulation mesure le nombre d'évènements qui seraient transmis aux différents véhicules passant par cette intersection en fonction de l'utilisation ou non des filtres sur la position des évènements définis dans STEAM. Les auteurs montrent ainsi que ce nombre d'évènements décroît d'environ 95%.

État actuel. La dernière publication relative à STEAM date de 2003. Aucun code n'est disponible sur la page relative au projet dans lequel s'est inscrit STEAM, le projet CORTEX.

2.2.2.25 TOTA

Contexte de réalisation. TOTA[35] (*Tuples On The Air*) [MZ03, MZL03, MZ04a, MZ04b, MZ05, MVZ05] a été développé par Marco Mamei, Franck Zambonelli, Matteo Vasirani et Laetizia Leonardi de 2003 à 2005 à l'*University of Modena and Reggio Emilia, Reggio Emilia*, en Italie.

Objectif. TOTA [MZ03, MZL03, MZ04a, MZ04b, MZ05, MVZ05] a pour objectif de faciliter le développement d'applications dans les réseaux mobiles ad hoc.

Caractéristiques techniques (et suppositions implicites). TOTA est un *middleware* orienté *Tuple Space* à la fois *location-based* et *content-based*. Il repose sur la propagation de *Tuples* sur le réseau. Un *Tuple* TOTA T est défini en fonction de son contenu C et de sa règle de propagation P, on a donc $T = (C, P)$. C est un ensemble ordonné de champs typés représentant l'information que transporte le *Tuple*. Sa règle de propagation P détermine la manière dont le *Tuple* doit être distribué et propagé dans le réseau. Elle peut par exemple inclure le "rayon" de propagation du *Tuple* (la distance en nombre de sauts à laquelle le *Tuple* doit être propagé et la direction cardinale de propagation, par exemple Nord, Sud, Est et Ouest) et la manière dont cette propagation peut être affectée par la

[35] `http://agentgroup.unimo.it/wiki/index.php/TOTA`

présence ou l'absence d'autres *Tuples* dans le système. Les auteurs de ce middleware compare leur approche à une approche évènementielle. Dans TOTA, la propagation n'est pas gérée par un modèle à base de publish/subscribe comme dans les modèles évènementiels classiques, mais elle est inscrite dans le *Tuple* lui-même. Contrairement à un évènement, un *Tuple* peut changer son contenu pendant sa propagation. Une règle de propagation peut aussi faire intervenir un algorithme de routage tel que GPSR [MZ03].

Les auteurs supposent cependant dans [MZ03] qu'un mécanisme de routage tel que GPSR peut être employé efficacement. La mobilité des nœuds est donc supposée suffisamment faible pour qu'un algorithme de routage tel que GPSR puisse être efficace.

Les technologies de communication gérées sont les technologies compatibles IEEE 802.11b (Wi-Fi). TOTA a été implémenté sur des PDA iPAQs sous Linux disposant d'une machine virtuelle J2ME (*Java 2 Micro Edition*) qui dispose de CDC (*Connected Device Configuration*).

L'API de TOTA est composée des fonctions suivantes :

1. public void **inject** (TotaTuple tuple) ; Cette fonction est utilisée pour injecter dans le réseau le *Tuple* passé en argument. Une fois injecté, il se propage dans le réseau tout en respectant ses règles de propagation.
2. public Vector **read** (Tuple template) ; Cette fonction est utilisée pour accéder au *Tuple Space* local et retourne les *Tuples* qui y sont présents et qui correspondent au *template* passé en paramètre.
3. public Vector **readOneHop** (Tuple template) ; Cette fonction retourne les *Tuples* présents dans les *Tuple Spaces* des voisins directs et qui correspondent au *template* passé en paramètre.
4. public Tuple **keyrd** (Tuple template) ; Chaque *Tuple* TOTA possède un identifiant unique invisible au niveau applicatif qui permet d'y accéder rapidement, sans prendre en compte son contenu. Il peut cependant exister plusieurs *Tuples* avec le même identifiant mais avec un contenu différent. En effet, lors de la création d'un *Tuple*, un identifiant unique est affecté à celui-ci. Lorsqu'il se propage, le *Tuple* est dupliqué. Il peut alors changer son contenu mais ne peut pas modifier son identifiant. Cette fonction cherche donc le (ou les) *Tuple(s)* qui possède(nt) le même identifiant que le *template* passé en argument. Cette fonction est le plus souvent utilisée pour vérifier si un *Tuple* spécifique a changé de contenu localement.
5. public Vector **keyrdOneHop** (Tuple template) ; Cette fonction cherche le ou les *Tuples* qui possèdent le même identifiant que le *Tuple template* passé en argument. Cette fonction est le plus souvent utilisée pour vérifier si un *Tuple* spécifique a changé de contenu dans le *Tuple Space* des voisins directs.
6. public Vector **delete** (Tuple template) ; La fonction **delete** cherche les *Tuples* de l'espace local correspondant au *Tuple template* passé en paramètre, les supprime et les retourne.
7. public void **subscribe** (Tuple template, ReactiveComponent comp, String rct) ; Les fonctions **subscribe** et **unsubscribe** sont définies pour la gestion des événements. Les auteurs argument que dans TOTA, tout évènement (arrivée d'un *Tuple*, connexion ou déconnexion d'un pair) peut être représenté par un *Tuple*. La fonction

subscribe associe une méthode à invoquer en réaction à l'arrivée du *Tuple template* passé en paramètre.

8. public void **unsubscribe** (Tuple template, ReactiveComponent comp) ; La fonction **unsubscribe** supprime l'association entre une méthode et l'arrivée d'un *Tuple* créée par la fonction **subscribe**.

Résultats des simulations et des tests. Dans [MZ04a], les auteurs cherchent à montrer que TOTA est capable de supporter le passage à l'échelle. Les changements topologiques qui peuvent intervenir dans le réseau sous-jacent entraînent des modifications des *Tuples* qui se trouvent dans les *Tuple Spaces*. Les auteurs de TOTA étudient le nombre de *Tuples* qui doivent être mis à jour lors d'un changement topologique afin de déterminer si le système peut passer à l'échelle. Ils montrent qu'un changement topologique local à peu de conséquences sur les nœuds qui se trouvent à plusieurs sauts. Pour cela, ils effectuent trois simulations avec une densité du réseau (c'est à dire le nombre moyen de voisins par nœuds) valant successivement 5,7, 7,2 puis 8,8. Ces valeurs sont le résultat de la variation du nombre de nœuds sur une superficie constante, le nombre de nœuds valant tour à tour 150, 200 puis 250 respectivement.

Dans [MVZ05], les auteurs présentent un exemple à base de "particules mobiles" compatibles avec les futures micro et nano technologies. Ces particules sont contraintes (elles exécutent le même code et elles n'ont connaissance ni des distances qui les séparent des autres particules, ni de la direction dans laquelle ces particules se trouvent, ni du temps) et ne sont capables que d'actions limitées : elles peuvent injecter et lire des *Tuples* dans le *Tuple Space* et se déplacer. TOTA s'inspire des mécanismes de morphogénèse biologique pour permettre aux nœuds du réseau de s'auto-organiser. La morphogénèse est un phénomène encore difficilement expliqué en biologie. Elle concerne la manière dont les cellules s'organisent pour la formation de structures biologiques. Par exemple, l'embrayon du drosophile utilise un morphogène (une protéine) qui se diffuse le long de l'embrayon. Elle est utilisée par les cellules pour déterminer si elles sont situées au niveau de la tête, du thorax ou la région abdominale. TOTA permet notamment aux nœuds de s'organiser en forme de roue, d'anneau ou de polygone (de triangle à pentagone). De plus, une modification brutale de la structure n'empêche pas les nœuds de se réorganiser afin de tendre en permanence vers la structure initiale.

État actuel. TOTA semble toujours actif. La dernière publication date de 2008 [MZ09]. Le code source de TOTA est disponible en ligne.

2.2.2.26 XMIDDLE

Contexte de réalisation. XMIDDLE[36] [MCE01, MCZE02, ZCME02] a été développé par Licia Capra[37], Wolfgang Emmerich[37], Cecilia Mascolo[37,38], Mirco Musolesi[38]) et Stefanos Zachariadis de 2001 à 2002 au *Department of Computer Science, University College*

[36] `http://xmiddle.sourceforge.net/`
[37] aussi présent(e) dans CARISMA (cf. section 2.2.2.5 page 31)
[38] aussi présent(e) dans EMMA (cf. section 2.2.2.8 page 36)

London, au Royaume-Uni.

Objectif. XMIDDLE [MCE01, MCZE02, ZCME02] a pour objectif le partage de documents XML entre des terminaux mobiles hétérogènes.

Caractéristiques techniques (et suppositions implicites). XMIDDLE est un *middleware* orienté message (MOM), voire même document, écrit en Java, et qui a été créé pour les réseaux mobiles ad hoc dont le graphe sous-jacent peut être déconnecté. Les auteurs indiquent que des communications ne peuvent avoir lieu que lorsque les dispositifs concernés sont mutuellement à portée l'un de l'autre. Il s'agit de communications deux-à-deux et les liens de communication sont donc supposés symétriques. Ce middleware ne permet pas de tirer parti des liens de communication asymétriques qui peuvent exister dans le réseau.

L'objectif de XMIDDLE est de fournir un service de partage de documents XML. Chaque document XML possède un propriétaire. XMIDDLE permet d'accéder aux documents partagés même en mode hors-ligne (c'est à dire lorsque le terminal qui souhaite modifier le document XML n'est plus connecté au terminal propriétaire). Afin de gérer les conflits qui peuvent survenir (dus à des modifications simultanées sur différentes versions distribuées), il fournit un mécanisme/protocole de réconciliation afin de faire converger les différentes versions du même document. Un protocole de réconciliation spécifique peut aussi être spécifié par l'application. XMIDDLE est en cela réflexif.

Les technologies de communication que peut gérer XMIDDLE sont l'IrDA, le Bluetooth et le 802.11b. Plus généralement, XMIDDLE fonctionne au dessus de UDP/IP, donc il peut fonctionner au dessus de toute technologie de communication capable de fournir une pile UDP/IP.

Résultats des simulations et des tests. XMIDDLE a été implémenté en Java et fonctionne sur des PDA Compaq iPAQ sous Linux. Certains paquetages utilisés par XMIDDLE (comme Xerces et Xalan) ne sont pas compatibles avec CLDC (*Connected Limited Device Configuration*) : XMIDDLE a donc besoin d'une machine virtuelle compatible avec Java en version 1.3.1.

État actuel. Le code source est disponible en ligne. La dernière version date du 9 mai 2004. Le projet ne semble donc plus actif.

Chapitre 3

CiMAN, un modèle formel pour les réseaux ad hoc fortement mobiles

Sommaire

Glossaire

Entités

Ensemble	Eléments	Description	Pages
$\mathcal{A}$	$a, b, c, ...$	liens logiques (réception)	65
$\overline{\mathcal{A}}$	$\overline{a}, \overline{b}, \overline{c}, ...$	liens logiques (émission)	65
$\bigwedge = \mathcal{A} \cup \overline{\mathcal{A}}$	$a, \overline{a}, b, \overline{b}, ...$	liens logiques	65
$\dot{\mathcal{A}}$	$\dot{a}, \dot{b}, \dot{c}, ...$	liens physiques (reception)	65
$\ddot{\mathcal{A}}$	$\ddot{a}, \ddot{b}, \ddot{c}, ...$	liens physiques (émission)	65
$\dot{\bigwedge} = \dot{\mathcal{A}} \cup \ddot{\mathcal{A}}$	$\dot{a}, \ddot{a}, \dot{b}, \ddot{b}, ...$	liens physiques	65
$\mathring{\mathcal{A}}$	$\mathring{a}, \mathring{b}, \mathring{c}, ...$	liens sans fil (réception)	65
$\mathring{\overline{\mathcal{A}}}$	$\mathring{\overline{a}}, \mathring{\overline{b}}, \mathring{\overline{c}}, ...$	liens sans fil (émission)	65
$\mathring{\bigwedge} = \mathring{\mathcal{A}} \cup \mathring{\overline{\mathcal{A}}}$	$\mathring{a}, \mathring{\overline{a}}, \mathring{b}, \mathring{\overline{b}}, ...$	liens sans fil	65
$\vec{\mathcal{A}}$	$\vec{a}, \vec{b}, \vec{c}, ...$	déplacements	65
$\mathcal{L} = \bigwedge \cup \dot{\bigwedge} \cup \mathring{\bigwedge} \cup \vec{\mathcal{A}}$	l	étiquettes	65
$Act = \mathcal{L} \cup \{\tau, \dot{\tau}\}$	$\alpha, \beta, ...$	actions	65
Act^*	$r, s, ...$	séquences d'actions	65
I	$i, j, ...$	ensemble d'indexation	65
$\breve{\mathcal{P}}$	$\breve{P}, \breve{P}_i, \breve{Q}, \breve{Q}_i$	expression de processus	65
$\mathcal{P}$	P, P_i, Q, Q_i	processus agent	66
$\breve{\mathcal{C}}$	$\breve{C}, \breve{C}_i$	expression d'unité de calcul	65
$\mathcal{C}$	C, C_i	unité de calcul agent	66
$\breve{\mathcal{N}}$	$\breve{N}, \breve{N}_i$	expression de nœud	65
$\mathcal{N}$	N, N_i	nœud agent	66
$\breve{\mathcal{M}}$	$\breve{M}, \breve{M}_i$	expression de message	65
$\breve{\mathcal{U}}$	$\breve{U}, \breve{U}_i$	expression d'une entité	65
$\mathcal{U}$	U, U_i	entité agent	66
$\mathcal{W}$	$w, w_i, ...$	ensemble des positions	90

Relations d'équivalence

Symbole	Description	Pages
$\equiv$	équivalence syntaxique	66
$\triangle$	bisimulations	68

Expressions de processus

Syntaxe	Description	Pages
0	processus agent inactif	71
$a(x).\breve{P}$	réception d'une valeur provenant d'un processus hébergé sur la même unité de calcul	71
$\overline{a}(e).\breve{P}$	envoi d'une valeur vers un processus hébergé sur la même unité de calcul	71
$\breve{P}[b/a]$	renommage de a en b	71
$\breve{P}_1 + \breve{P}_2$	somme ou choix	71
$\breve{P}_1 \mid \breve{P}_2$	composition parallèle	71
$\breve{P} \setminus a$	restriction	71
$\dot{a}(x).\breve{P}$	réception d'une valeur provenant d'une autre unité de calcul	71
$\ddot{a}(e).\breve{P}$	envoi d'une valeur vers les autres unités de calcul	71
$\mathring{a}\,(x).\breve{P}$	réception d'une valeur OTA[1]	71
$\mathring{a}^{\circ}\,(e).\breve{P}$	envoi d'une valeur OTA	71
$\vec{w}.\breve{P}$	déplacement	71

Expressions d'unités de calcul

Syntaxe	Description	Pages
$[\breve{P}]$	unité de calcul qui exécute $\breve{P}$	77
$\breve{C}[b/a]$	renommage de a en b	77
$\breve{C}_1 \parallel \breve{C}_2$	composition parallèle	77
$\breve{C} \diagdown\!\!\diagdown a$	restriction	77

Expressions de nœuds/messages

Syntaxe	Description	Pages
${}_i[\![\breve{C}]\!]^G_w$	nœud i à la position w qui possède un graphe de mobilité G	89
${}^{\alpha}_{i}(\!|e|\!)^G_W$	message i aux positions de l'ensemble W qui possède un graphe de mobilité G et qui a été émis sur le canal α	90

[1] OTA = *Over The Air*

Syntaxe	Description	Pages
$\breve{M}_1 \mid\mid\mid \breve{M}_2$	composition parallèle de messages	90
$\breve{N}_1 \mid\mid\mid \breve{N}_2$	composition parallèle de nœuds	89
$\breve{O} \mid\mid\mid \breve{M}$	composition parallèle du monde et d'un message	90
$\breve{O} \mid\mid\mid \breve{N}$	composition parallèle du monde et d'un nœud	90

Actions, transitions

Syntaxe	Description	Pages
τ	action inter-processus silencieuse ou parfaite	65
$\dot{\tau}$	action inter-unité de calcul silencieuse	65
$\xrightarrow{\alpha}$	transition faisant apparaître l'événement α	66
$\xrightarrow{r}$	séquence de transitions	66
$\vartriangle\overset{\alpha}{\Rightarrow}$	transition dans laquelle les éléments de $nobs(\vartriangle)$ sont ignorés	67
$\vartriangle\overset{r}{\Rightarrow}$	séquence de transitions dans laquelle les éléments de $nobs(\vartriangle)$ sont ignorés	68

La modélisation formelle d'un système quel qu'il soit repose souvent sur des hypothèses que les auteurs dudit système jugent pertinentes voire réalistes. Le domaine des réseaux mobiles ad hoc ne déroge pas à cette règle. En effet, les modèles formels correspondant sont là aussi basés sur un certain nombre d'hypothèses, hypothèses qui concernent notamment les capacités de communication et la mobilité des nœuds qui composent ces réseaux. La validité des résultats obtenus dans ce cadre dépend directement de la validité des hypothèses sur lesquelles ils reposent. Après avoir précisé (i) pourquoi les hypothèses les plus fréquemment adoptées ne sont pas réalistes, et (ii) les conséquences de certaines de ces hypothèses sur les applications et les algorithmes développés dans ce contexte (cf. chapitre 1), nous avons étudié les modèles de calcul pour les réseaux mobiles ad hoc existant (cf. chapitre 2) et mis en évidence l'absence de lien direct entre les hypothèses sur lesquelles ils reposent et la réalité.

Nos travaux sur les réseaux mobiles ad hoc fortement mobiles nous ont conduit à la question de la modélisation et de la vérification des applications et des algorithmes que nous avons conçus pour ce contexte. Nous souhaitons en effet modéliser ces applications et ces algorithmes afin de les valider formellement dans un contexte de réseau ad hoc fortement mobile que nous jugeons réaliste. Pour cela, nous limitons notamment les hypothèses faites (i) sur la nature des nœuds qui composent le réseau, (ii) sur les technologies de communication qu'ils utilisent, et (iii) sur leur mobilité. Le modèle de calcul que nous avons ainsi développé, CiMAN, repose sur l'absence de la plupart des hypothèses classiques, souvent aussi nombreuses que peu réalistes. En diminuant le nombre d'hypothèses sur lesquelles il repose, nous plaçons notre modèle dans un cadre plus général que celui des modèles de calcul pour réseaux mobiles ad hoc existants. Nous précisons finement les hypothèses sur lesquelles repose notre modèle après avoir donné une vision intuitive de celui-ci.

3.1 Vision intuitive du modèle

Un réseau mobile ad hoc est composé de nœuds tels que des téléphones mobiles, des PDA, des PC portables, des tablettes, des robots, etc. Afin d'apporter de la sécurité dans ces réseaux, nous supposons que les nœuds peuvent être équipés de *Secure Elements* tels que des cartes à puce, des cartes SIM, etc.

Il est très important de noter la diversité des types de communication qui peuvent prendre place dans un tel réseau. Les modèles de calcul étudiés dans le chapitre 2 sont des modèles de calcul à deux niveaux dans lesquels les processus sont embarqués dans des entités mobiles appelées nœuds. La communication entre deux processus embarqués sur un même nœud n'est pas modélisée dans ces systèmes et seuls les processus embarqués sur des nœuds différents peuvent échanger des messages. Ces modèles ne permettent donc pas de modéliser les systèmes composés de nœuds qui embarquent des processus capables à la fois de communications locales (c'est à dire entre des processus hébergés sur un même nœud) et de communications distantes (c'est à dire entre processus hébergés sur des nœuds distincts).

Afin de rapprocher CiMAN des réseaux que nous considérons, nous modélisons les communications entre les processus qui se trouvent sur un même nœud. Il devient donc nécessaire de distinguer les communications internes à une unité de calcul des communications entre processus embarqués sur des unités de calcul différentes. Les évènements observables associés à ces communications ne doivent donc pas être identiques. En effet, dans le cas contraire, il ne serait pas possible de modéliser la sécurité du système. Par exemple, il est possible pour un dispositif (unité de calcul) tierce d'écouter les messages qui circulent entre deux unités de calcul que sont un téléphone et sa carte SIM par exemple ; mais il peut lui être impossible d'écouter les communications entre deux processus de la même unité de calcul. Notre modèle doit donc permettre de distinguer les communications entre deux processus hébergés dans la même unité de calcul de celles se déroulant entre deux processus situés sur le même nœud mais dans des unités de calcul différentes. Il doit aussi permettre de distinguer ces communications des communications entre deux processus hébergés sur des nœuds distincts. Notre modèle est donc composé de trois niveaux (cf. figure 3.9). Le premier est celui concernant les processus. Un processus est une expression qui définit un comportement, c'est à dire les actions que peut effectuer une entité. Le second est le niveau unité de calcul. Une unité de calcul héberge un ensemble de processus. Le troisième et dernier niveau est le niveau nœud. Un nœud est un ensemble d'unités de calcul ; il possède une position géographique et un graphe de mobilité. Chacun de ces niveaux apporte des primitives de communication originales dont les caractéristiques dépendent directement de leur abstraction à partir d'un contexte de iMANet (cf. section 1.2 page 14). Nous détaillons ces différents niveaux dans les sections suivantes.

3.1.1 Les processus

Le premier niveau, le niveau "le plus bas", est celui des processus. Un processus est une expression qui définit un ensemble d'actions locales et de communications. Notre vision du processus est la même que la vision de R. Milner [Mil82, Mil89]. Plus précisément, l'ensemble des processus qui peuvent être définis en utilisant la définition de R. Milner est

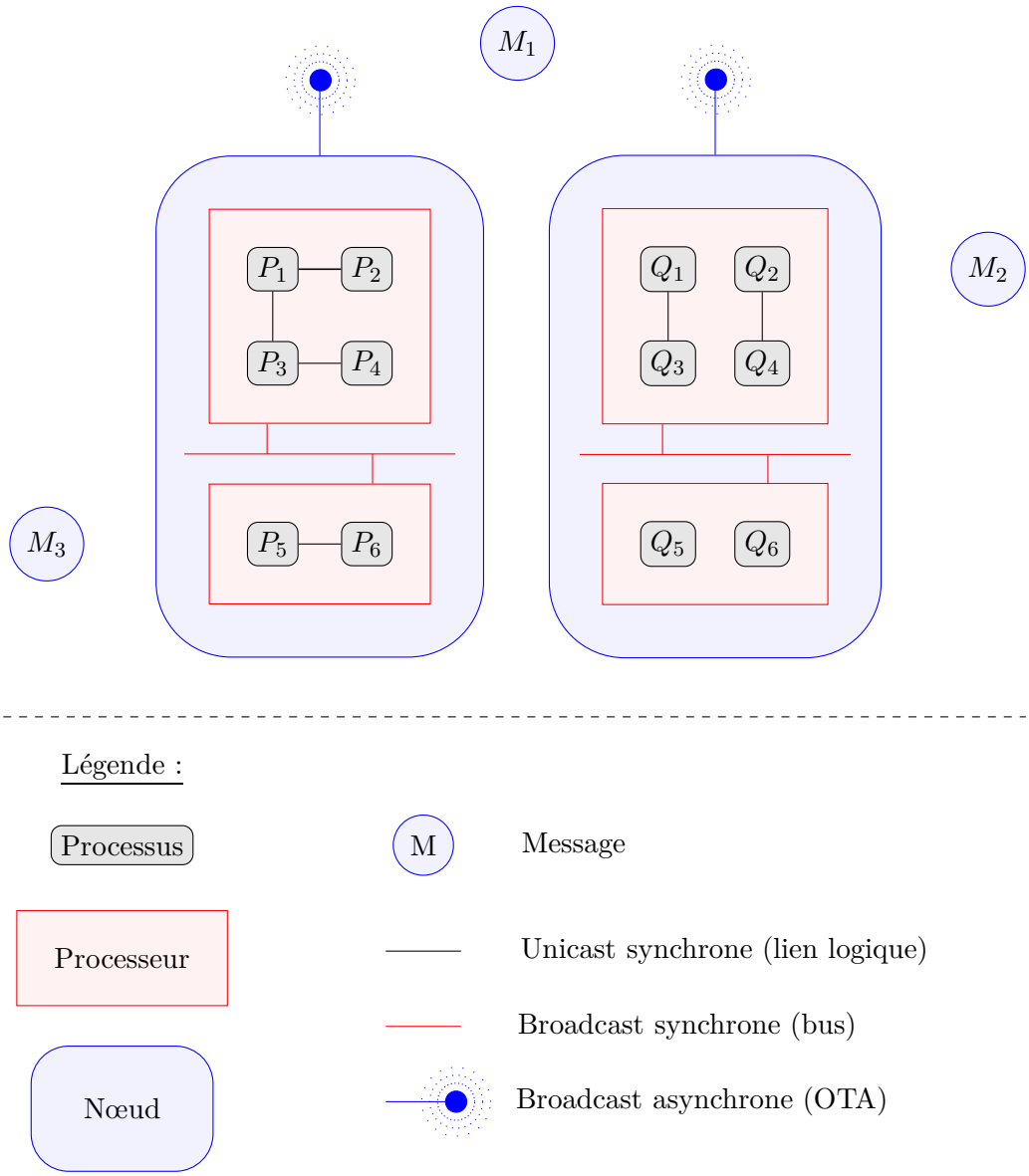


FIG. 3.9: Un exemple de système

inclus dans l'ensemble des processus que notre modèle permet de définir.

De la même manière qu'en CCS, un processus est un ensemble d'actions. Nous avons défini trois types d'action, chacun correspondant à un niveau de communication différent :

1. Les actions internes à l'ensemble des processus. Ces actions permettent à deux processus qui sont sur la même unité de calcul de communiquer entre eux. Nous les notons $a, \overline{a}, ...$ L'action τ qui peut apparaître dans un processus correspond à un évènement non observable, et ce peut être en particulier une communication (cf. section 3.3 page 70).
2. Les actions inter-unités de calcul et intra-nœuds. Elles permettent à des processus

qui se trouvent sur des unités de calcul différentes mais sur le même nœud de communiquer (via un bus). Il s'agit d'une opération de *broadcast* atomique (simultanéïté de l'émission et des réceptions) sur un bus (tous les processus en réception reçoivent le message). Nous les écrivons $\dot{a}, \ddot{a}, ...$ (cf. section 3.4 page 76). Les règles d'inférences applicables au niveau des unités de calcul rendent les actions d'émission $\ddot{a}, \ddot{b}, ...$ non bloquantes et les actions de réception $\dot{a}, \dot{b}, ...$ bloquantes.

3. Les actions de communication inter-nœud. Elles permettent à un processus de communiquer (par radio) avec un ou plusieurs processus hébergés sur différents nœuds. Nous les écrivons $\mathring{a}$, $\mathring{\dot{a}}$, ... La première est une action de réception, la seconde une action d'émission (cf. section 3.5 page 89). De même que pour les actions $\dot{a}, \ddot{a}, ...$, les règles d'inférences applicables au niveau des nœuds et des messages rendent les actions d'émission $\mathring{\ddot{a}}$, $\mathring{\ddot{b}}$, ... non bloquantes et les actions de réception $\mathring{a}, \mathring{b}, ...$ bloquantes.

3.1.2 Les unités de calcul

Chaque processus est hébergé sur une unité de calcul. Celle-ci peut contenir plusieurs processus en parallèle. Les processus hébergés sur des unités de calcul différentes communiquent entre eux en utilisant une opération de *broadcast* atomique directement inspirée de CBS [Pra93, Pra95]. Ceci signifie que nous supposons que dans le monde réel les unités de calcul sont reliées entre elles par un bus par exemple. Des processus hébergés sur des unités de calcul différentes communiquent donc en utilisant les opérations du type $\dot{a}$ et $\ddot{a}$. La première opération est bloquante et permet la réception d'un message. Au contraire, la seconde opération est une diffusion non bloquante. Les détails concernant les unités de calcul et les opérations de communication associées sont donnés en section 3.4 page 76.

3.1.3 Les nœuds et les messages

Les unités de calcul sont hébergées sur les nœuds. Un nœud est un ensemble d'unités de calcul solidairement mobiles qui possèdent la même position et le même graphe de mobilité. Les processus contenus dans les unités de calcul contenues par un nœud communiquent entre eux en utilisant une opération de *broadcast* non atomique. Ils peuvent recevoir et envoyer des messages grâce aux opérations $\mathring{a}$ et $\mathring{\dot{a}}$. Comme indiqué ci-dessus, la première opération est bloquante et permet au processus de recevoir un message. La seconde opération est non bloquante et provoque la création d'un message.

Les messages sont des données *Over The Air* (OTA). Ils ne possèdent pas forcément une unique position. En effet, une fois diffusé par ondes radio, un message peut être présent à plusieurs positions en même temps. Un message est donc une entité à part entière qui possède son propre graphe de mobilité, que nous appelons dans ce cas *graphe de propagation*.

Graphe de mobilité et graphe de propagation. Ces graphes modélisent les déplacements possibles des nœuds et des messages OTA. Les sommets du graphe de mobilité

correspondent à un ensemble de positions, ses arcs correspondent aux déplacements possibles entre ces positions. Le graphe de mobilité d'un nœud ou d'un message est donc un graphe orienté.

3.1.4 Le monde réel tel que nous le voyons

Comme expliqué précédemment, les modèles reposent souvent, voire toujours, sur un ensemble d'hypothèses. Nous présentons maintenant celles sur lesquelles s'appuie CiMAN. Elles concernent par exemple la mobilité des nœuds et des messages ainsi que les caractéristiques des primitives de communication.

Hypothèses de communication. Nous présentons tout d'abord les hypothèses qui concernent les actions de communication sur lesquelles repose CiMAN. Ce sont les suivantes :

1. La communication entre deux processus hébergés par une même unité de calcul est synchrone et atomique.
2. La communication entre deux processus hébergés par des unités de calcul distinctes mais appartenant au même nœud est de type *broadcast* atomique. La réception est bloquante mais l'émission ne l'est pas.
3. La communication entre deux processus hébergés par des unités de calcul qui n'appartiennent pas au même nœud est de type *broadcast* asynchrone non atomique. Comme précédemment, la réception est bloquante mais l'émission ne l'est pas.

Autres hypothèses.

1. Les déplacements des nœuds et des messages peuvent être discrétisés.
2. La réception d'un message M par un processus P qui est dans un nœud N dépend non seulement de la présence d'une action de réception explicite par P, mais aussi des positions de M et N.
3. Les déplacements, l'envoi et la réception d'un message sont atomiques (aucune action ni aucun évènement ne peut avoir lieu pendant ces opérations).

3.2 Définitions générales

Avant de définir formellement les processus, unités de calcul, nœuds et messages, nous avons besoin d'un ensemble de définitions fondamentales. En particulier nous introduisons les bisimulations. En effet, CiMAN est un système de transitions étiquetées sur lequel nous avons défini de telles relations qui nous permettent de démontrer des propositions spécifiques de notre modèle.

3.2.1 Définitions des éléments de base

Nous commençons par donner les définitions fondamentales. Celles-ci permettent de simplifier les définitions et les propositions qui concernent spécifiquement chacun des niveaux processus, unités de calcul et nœuds.

Définition 1. *I est un ensemble infini d'indexation. Nous utilisons $i, j, \ldots$ pour dénoter les éléments de I.*

Définition 2. *$\breve{\mathcal{P}}$ (resp. $\breve{\mathcal{C}}$, $\breve{\mathcal{N}}$, $\breve{\mathcal{M}}$) est l'ensemble des expressions agentiques des processus (resp. des unités de calcul, des nœuds, des messages) de CiMAN. La grammaire qui définit les éléments de $\breve{\mathcal{P}}$ (resp. $\breve{\mathcal{C}}, \breve{\mathcal{N}}, \breve{\mathcal{M}}$) est donnée en section 3.3.1 page 70 (resp. section 3.4.1 page 77, section 3.5.1 page 89, section 3.5.1 page 89). Nous utilisons $\breve{P}$, $\breve{P}_i$, $\breve{Q}$ et $\breve{Q}_i$ (resp. $\breve{C}$ et $\breve{C}_i$, $\breve{N}$ et $\breve{N}_i$, $\breve{M}$ et $\breve{M}_i$) pour dénoter les éléments de $\breve{\mathcal{P}}$ (resp. $\breve{\mathcal{C}}$, $\breve{\mathcal{N}}$, $\breve{\mathcal{M}}$).*

Définition 3. *$\breve{\mathcal{U}} = \breve{\mathcal{P}} \cup \breve{\mathcal{C}} \cup \breve{\mathcal{N}} \cup \breve{\mathcal{M}}$ est l'ensemble des expressions agentiques de CiMAN. Nous utilisons $\breve{U}$, $\breve{U}_i$ pour dénoter les éléments de $\breve{\mathcal{U}}$.*

Définition 4. *$\mathcal{A}$, $\overline{\mathcal{A}}$, $\dot{\mathcal{A}}$, $\ddot{\mathcal{A}}$, $\mathring{\mathcal{A}}$, $\mathring{\dot{\mathcal{A}}}$ et $\vec{\mathcal{A}}$[2] sont des ensembles infinis de noms tous deux à deux disjoints. Nous utilisons $a, b, c, \ldots$ pour dénoter les éléments de $\mathcal{A}$, $\overline{a}, \overline{b}, \overline{c}, \ldots$ pour dénoter les éléments de $\overline{\mathcal{A}}$; $\dot{a}, \dot{b}, \dot{c}, \ldots$ pour dénoter les éléments de $\dot{\mathcal{A}}$; $\ddot{a}, \ddot{b}, \ddot{c}, \ldots$ pour dénoter les éléments de $\ddot{\mathcal{A}}$; $\mathring{a}, \mathring{b}, \mathring{c}, \ldots$ pour dénoter les éléments de $\mathring{\mathcal{A}}$; $\mathring{\dot{a}}$, $\mathring{\dot{b}}$, $\mathring{\dot{c}}$, ..., pour dénoter les éléments de $\mathring{\dot{\mathcal{A}}}$; $\vec{a}, \vec{b}, \vec{c}, \ldots$ pour dénoter les élements de $\vec{\mathcal{A}}$. Nous définissons les bijections $\overline{\ }$, $\dot{\ }$ et $\mathring{\ }$ telles que :*

$$\overline{\ } : \begin{array}{ccc} \mathcal{A} & \to & \overline{\mathcal{A}} \\ a & \mapsto & \overline{a} \end{array} \qquad \dot{\ } : \begin{array}{ccc} \dot{\mathcal{A}} & \to & \ddot{\mathcal{A}} \\ \dot{a} & \mapsto & \ddot{a} \end{array} \qquad \mathring{\ } : \begin{array}{ccc} \mathring{\mathcal{A}} & \to & \mathring{\dot{\mathcal{A}}} \\ \mathring{a} & \mapsto & \mathring{\dot{a}} \end{array}$$

Définition 5. *$\bigwedge = \mathcal{A} \cup \overline{\mathcal{A}}$, $\dot{\bigwedge} = \dot{\mathcal{A}} \cup \ddot{\mathcal{A}}$ et $\mathring{\bigwedge} = \mathring{\mathcal{A}} \cup \mathring{\dot{\mathcal{A}}}$.*

Remarque : $\overline{\mathcal{A}}$ est donc le complémentaire de $\mathcal{A}$ sur $\bigwedge$, $\ddot{\mathcal{A}}$ est le complémentaire de $\dot{\mathcal{A}}$ sur $\dot{\bigwedge}$ et $\mathring{\dot{\mathcal{A}}}$ est le complémentaire de $\mathring{\mathcal{A}}$ sur $\mathring{\bigwedge}$.

Définition 6. *$\mathcal{L} = \bigwedge \cup \dot{\bigwedge} \cup \mathring{\bigwedge} \cup \vec{\mathcal{A}}$ est l'ensemble des étiquettes de CiMAN.*

Définition 7. *$Act = \mathcal{L} \cup \{\tau, \dot{\tau}\}$ est l'ensemble des actions possibles d'un processus. Nous utilisons $\alpha, \beta, \ldots$ pour dénoter les éléments de Act.*

Comme nous le verrons plus loin, τ peut modéliser une communication et $\dot{\tau}$ une émission restreinte à certaines unités de calcul. De la même manière qu'en CCS, nous définissons plusieurs relations d'équivalence en CiMAN qui diffèrent par la considération (observation) ou non des évènements tels que τ et $\dot{\tau}$ (cf. section 3.2.4).

Définition 8. *Act^* est l'ensemble des séquences d'actions possibles dans CiMAN.*

Définition 9. *Pour simplifier les écritures à venir nous notons*

1. *$Act(\mathcal{P}) = \bigwedge \cup \{\tau\}$*
2. *$Act(\mathcal{C}) = Act(\mathcal{P}) \cup \dot{\bigwedge} \cup \{\dot{\tau}\}$*

[2] $\vec{\mathcal{A}}$ est l'ensemble des actions de déplacements des nœuds. Il n'existe pas d'ensemble complémentaire à $\vec{\mathcal{A}}$.

3. $Act(\mathcal{N}) = Act(\mathcal{C}) \cup \mathring{\bigwedge} \cup \vec{\mathcal{A}}$[3]

Remarque : $Act(\mathcal{P})$ = **Act** tel que défini par R. Milner dans [Mil89].

Définition 10. *$\mathcal{P}$ (resp. $\mathcal{C}$, $\mathcal{N}$) est l'ensemble des processus (resp. des unités de calcul, des nœuds) agents de CiMAN. Un processus P (resp. une unité de calcul C, un nœud N) agent est une expression agentique d'un processus (resp. d'une unité de calcul, d'un nœud) tel que $\mathcal{L}(P) \subseteq Act(\mathcal{P})$ (resp. $\mathcal{L}(C) \subseteq Act(\mathcal{C})$, $\mathcal{L}(N) \subseteq Act(\mathcal{N})$). Nous utilisons P, P_i, Q et Q_i (resp. C et C_i, N et N_i) pour dénoter les élements de $\mathcal{P}$ (resp. $\mathcal{C}$, $\mathcal{N}$). On parle aussi de processus (resp. unité de calcul, nœud) clos(e).*

Définition 11. *$\mathcal{U} = \mathcal{P} \cup \mathcal{C} \cup \mathcal{N}$ est l'ensemble des agents de CiMAN; c'est à dire des processus, unités de calcul et nœuds clos. Nous utilisons U, U_i pour dénoter les éléments de $\mathcal{U}$.*

3.2.2 Système de transitions étiquetées

Définition 12. *Un système de transitions étiquetées (LTS) sur Act est un couple $(\breve{\mathcal{U}}, \mathcal{T})$ tel que*
– $\breve{\mathcal{U}}$ est un ensemble d'expressions agentiques de CiMAN.
– $\mathcal{T} \subseteq (\breve{\mathcal{U}} \times Act \times \breve{\mathcal{U}})$ est une relation ternaire appelée relation de transition.

Notation. *Si $(\breve{U}, \alpha, \breve{U}') \in \mathcal{T}$ nous écrivons $\breve{U} \xrightarrow{\alpha} \breve{U}'$. Nous écrivons aussi $\breve{U} \xrightarrow{\alpha}$ pour signifier qu'il existe $\breve{U}'$ tel que $\breve{U} \xrightarrow{\alpha} \breve{U}'$.*

Définition 13. *Nous définissons la séquence vide $\epsilon \in Act^*$ comme la séquence telle que pour tout $\breve{U} \in \breve{\mathcal{U}}$, $\breve{U} \xrightarrow{\epsilon} \breve{U}$.*

Notation. *Soit $r = \alpha_1 ... \alpha_n \in Act^*$. Si $\breve{U} \xrightarrow{\alpha_1} \breve{U}_1 \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} \breve{U}_n$ alors nous appelons $\breve{U}_n$ dérivée de $\breve{U}$ par r et nous notons $\breve{U} \xrightarrow{r} \breve{U}'$. Nous écrivons aussi $\breve{U} \xrightarrow{r}$ pour signifier qu'il existe $\breve{U}'$ tel que $\breve{U} \xrightarrow{r} \breve{U}'$.*

3.2.3 Sorte

Définition 14. *Soient $\breve{U} \in \mathcal{U}$ et $L \subseteq \mathcal{L}$. Si les actions de $\breve{U}$ et de toutes ses dérivées appartiennent à $L \cup \{\tau, \dot{\tau}\}$ alors nous disons que L est une sorte de $\breve{U}$, et on note $\breve{U} : L$.*

3.2.4 Relations d'équivalence

3.2.4.1 Equivalence syntaxique

Définition 15. *$P_1 \equiv P_2$ signifie que P_1 et P_2 sont syntaxiquement identiques.*

3.2.4.2 Bisimulations

Définition 16. *Nous définissons $obs(\sim) = Act$.*

[3] Il n'existe pas d'évènement $\mathring{\tau}$ qui résulterait d'une simultanéïté au niveau des actions inter-nœuds. De plus, il n'existe pas d'ensemble complémentaire à l'ensemble $\vec{\mathcal{A}}$ et encore moins de $\vec{\tau}$.

Définition 17. *Nous définissons les ensembles suivants :*

1. $\forall \triangle \in \{\sim\}, obs(\tilde{\triangle}) = obs(\triangle) - \{\tau\}$
2. $\forall \triangle \in \{\sim, \approx\}, obs(\dot{\triangle}) = obs(\triangle) - \{\dot{\tau}\}$
3. $\forall \triangle \in \{\sim, \approx, \dot{\sim}, \dot{\approx}\}, obs(\mathring{\triangle}) = obs(\triangle) - \mathring{\bigwedge}$
4. $\forall \triangle \in \{\sim, \approx, \dot{\sim}, \mathring{\sim}, \dot{\approx}, \mathring{\dot{\approx}}, \dot{\mathring{\sim}}, \mathring{\dot{\approx}}\}, obs(\vec{\triangle}) = obs(\triangle) - \vec{\mathcal{A}}$

Exemple : $obs(\dot{\sim}) = obs(\sim) - \{\dot{\tau}\}$

Notation. *Pour tout* $\triangle \in \{\sim, \approx, \dot{\sim}, \dot{\approx}\}$, *nous notons* $\overrightarrow{\triangle} = \vec{\triangle}$.

La notation précédente permet de simplifier les symboles des bisimulations.

Définition 18. $\blacktriangle = \{\sim, \approx, \dot{\sim}, \mathring{\sim}, \vec{\sim}, \dot{\approx}, \mathring{\approx}, \vec{\approx}, \mathring{\dot{\sim}}, \dot{\vec{\sim}}, \overrightarrow{\sim}, \mathring{\dot{\approx}}, \dot{\vec{\approx}}, \overrightarrow{\approx}, \mathring{\dot{\sim}}, \dot{\overrightarrow{\approx}}\}$

Définition 19. *Pour tout* $\triangle \in \blacktriangle$, $nobs(\triangle) = Act - obs(\triangle)$.

Le tableau 3.9 récapitule quels éléments appartiennent à chaque ensemble (✓ signifie "est élément de" ou "est inclus dans").

Ensemble \ $\triangle$	$\sim$	$\approx$	$\dot{\sim}$	$\mathring{\sim}$	$\vec{\sim}$	$\dot{\approx}$	$\mathring{\approx}$	$\vec{\approx}$	$\mathring{\dot{\sim}}$	$\dot{\vec{\sim}}$	$\overrightarrow{\sim}$	$\mathring{\dot{\approx}}$	$\dot{\vec{\approx}}$	$\overrightarrow{\approx}$	$\dot{\overrightarrow{\sim}}$	$\dot{\overrightarrow{\approx}}$
$\bigwedge$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\dot{\bigwedge}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
τ	✓		✓	✓	✓				✓	✓	✓				✓	
$\dot{\tau}$	✓	✓		✓	✓		✓	✓			✓			✓		
$\mathring{\bigwedge}$	✓	✓	✓		✓	✓		✓		✓			✓			
$\vec{\mathcal{A}}$	✓	✓	✓	✓		✓	✓		✓			✓				

TAB. 3.9: Définition des ensembles $obs(\triangle)$, $\triangle \in \blacktriangle$

CiMAN est constitué de plusieurs niveaux que sont les processus, unités de calcul et nœuds. Les évènements observables possibles sur lesquels reposent les relations d'équivalence définies sur chacun des niveaux dépendent précisément du niveau sur lequel la relation d'équivalence est définie. Nous définissons maintenant ces ensembles qui représentent donc l'ensemble des évènements observables possibles à un niveau donné du modèle.

Définition 20. *Pour tout* $\mathcal{X} \in \{\mathcal{P}, \mathcal{C}, \mathcal{N}\}$ *et pour tout* $\triangle \in \blacktriangle$, $obs_{\triangle}(\mathcal{X}) = Act(\mathcal{X}) \cap obs(\triangle)$.

Exemple : $obs_{\dot{\approx}}(\mathcal{C}) = Act(\mathcal{C}) \cap obs(\dot{\approx}) = (\bigwedge \cup \dot{\bigwedge} \cup \{\tau, \dot{\tau}\}) \cap (\bigwedge \cup \dot{\bigwedge} \cup \mathring{\bigwedge} \cup \vec{\mathcal{A}}) = \bigwedge \cup \dot{\bigwedge}$

Définition 21. *Soient* $\alpha \in Act$ *et* $\triangle \in \blacktriangle$. *La relation* $\triangle \overset{\alpha}{\Rightarrow}$ *est définie comme suit.* $\triangle \overset{\alpha}{\Rightarrow} \overset{def}{=} \overset{r}{\rightarrow}\overset{\alpha}{\rightarrow}\overset{s}{\rightarrow}$ *avec* $r, s \in nobs(\triangle)^*$. *Si* $\triangle = \sim$ *alors on notera aussi* $\triangle \overset{\alpha}{\Rightarrow}$ *par* $\overset{\alpha}{\rightarrow}$.

Exemples :
- $\approx \overset{\alpha}{\Rightarrow} \overset{\text{def}}{=} (\overset{\tau}{\rightarrow})^* \overset{\alpha}{\rightarrow} (\overset{\tau}{\rightarrow})^*$
- $\dot{\approx} \overset{\alpha}{\Rightarrow} \overset{\text{def}}{=} \overset{r}{\rightarrow}\overset{\alpha}{\rightarrow}\overset{s}{\rightarrow}$ avec $r, s \in \{\tau, \dot{\tau}\}^*$

Remarque : La transition $\approx\stackrel{\alpha}{\Rightarrow}$ que nous avons définie correspond à la transition $\stackrel{\alpha}{\Rightarrow}$ de CCS [Mil89].

Définition 22. *Soit la séquence $r \in Act^*$ telle que $r = \alpha_1 ... \alpha_n$. La relation $\vartriangle\stackrel{r}{\Rightarrow}$ est définie comme suit.* $\vartriangle\stackrel{r}{\Rightarrow} \stackrel{def}{=} \vartriangle\stackrel{\alpha_1}{\Rightarrow} ... \vartriangle\stackrel{\alpha_n}{\Rightarrow}$.

Définition 23 ($\vartriangle$-simulation)**.** *Soient $(\mathcal{U}, \mathcal{T})$ un LTS sur Act et $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{U}$ une relation binaire. $\mathcal{R}$ est une $\vartriangle$-simulation sur $(\mathcal{U}, \mathcal{T})$ si pour tout $(U_1, U_2) \in \mathcal{R}$ et pour tout $\alpha \in Act$, si $U_1 \vartriangle\stackrel{\alpha}{\Rightarrow} U_1'$ alors il existe U_2' tel que $U_2 \vartriangle\stackrel{\alpha}{\Rightarrow} U_2'$ et $(U_1', U_2') \in \mathcal{R}$.*

La condition pour que $\mathcal{R}$ soit une $\vartriangle$-simulation peut-être exprimée grâce au diagramme suivant :

$$\text{si} \quad \begin{array}{ccc} U_1 & \mathcal{R} & U_2 \\ \triangleright & & \\ \Downarrow \alpha & & \\ U_1' & & \end{array} \qquad \text{alors il existe } U_2' \text{ tel que} \qquad \begin{array}{ccc} & & U_2 \\ & & \triangleright \\ & & \Downarrow \alpha \\ U_1' & \mathcal{R} & U_2' \end{array}$$

On dit alors que U_2 $\vartriangle$-simule U_1 ou que U_1 est $\vartriangle$-simulé par U_2.

Rappel : La relation inverse $\mathcal{R}^{-1}$ d'une relation binaire $\mathcal{R}$ est l'ensemble des couples (y, x) tels que $(x, y) \in \mathcal{R}$.

Définition 24. *Une relation binaire $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{U}$ est une $\vartriangle$-bisimulation sur $(\mathcal{U}, \mathcal{T})$ si $\mathcal{R}$ et son inverse $\mathcal{R}^{-1}$ sont toutes les deux des $\vartriangle$-simulations. On dit aussi que U_1 et U_2 sont $\vartriangle$-bisimilaires ou $\vartriangle$-équivalents, écrit $U_1 \vartriangle U_2$, s'il existe une $\vartriangle$-bisimulation $\mathcal{R}$ telle que $U_1 \mathcal{R} U_2$.*

Une relation binaire $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{U}$ est donc une $\vartriangle$-bisimulation si, pour tout $\alpha \in Act$, les deux implications suivantes sont vérifiées :
– *si $U_1 \vartriangle\stackrel{\alpha}{\Rightarrow} U_1'$ alors, il existe U_2' tel que $U_2 \vartriangle\stackrel{\alpha}{\Rightarrow} U_2'$ et $U_1' \vartriangle U_2'$.*
– *si $U_2 \vartriangle\stackrel{\alpha}{\Rightarrow} U_2'$ alors, il existe U_1' tel que $U_1 \vartriangle\stackrel{\alpha}{\Rightarrow} U_1'$ et $U_1' \vartriangle U_2'$.*

Remarques :

1. Prenons le cas où $\vartriangle = \sim$ dans la définition 24. La $\sim$-bisimulation sur $(\mathcal{P}, \mathcal{T})$ correspond alors à la définition de la bisimulation forte de Milner dans [Mil89], la transition $\sim\stackrel{\alpha}{\Rightarrow}$ étant une transition $\stackrel{\alpha}{\rightarrow}$ au sens de Milner. On notera que le symbole associé ($\sim$) est le même dans CiMAN que dans CCS.
2. De même, considérons le cas où $\vartriangle = \approx$ dans la définition 24. La $\approx$-bisimulation sur $(\mathcal{P}, \mathcal{T})$ correspond à la définition de bisimulation faible de R. Milner dans [Mil89] car la transition $\approx\stackrel{\alpha}{\Rightarrow}$ correspond à la transition $\stackrel{\alpha}{\Rightarrow}$ de [Mil89]. On pourra noter que le symbole associé ($\approx$) est le même dans CiMAN que dans CCS.

Implications entre les relations de bisimilarité

CiMAN propose 16 relations d'équivalence, chacune permettant d'observer un sous-ensemble des évènements du système. Les implications entre ces relations d'équivalence sont directement liées aux évènements observables qui leurs sont associés. La proposition suivante et

son corollaire précisent que si deux entités sont équivalentes par l'observation d'un certain nombre d'évènements, alors elles le sont aussi par l'observation d'un sous-ensemble de ces évènements.

Proposition 25. *Soient $\breve{X}, \breve{X}'$ deux éléments du même ensemble $\breve{\mathcal{X}} \in \{\breve{\mathcal{P}}, \breve{\mathcal{C}}, \breve{\mathcal{N}}\}$. Pour tout $\alpha \in Act$ et pour tout $\triangle, \triangle' \in \blacktriangle$ tels que $nobs_{\triangle}(\breve{\mathcal{X}}) \subseteq nobs_{\triangle'}(\breve{\mathcal{X}})$, si $\breve{X} \triangle \overset{\alpha}{\Rightarrow} \breve{X}'$ alors $\breve{X} \triangle' \overset{\alpha}{\Rightarrow} \breve{X}'$.*

Démonstration. $\breve{X} \triangle \overset{\alpha}{\Rightarrow} \breve{X}'$ donc il existe $s = s_1.\alpha.s_2 \in Act(\breve{\mathcal{X}})^*$ telle que $s_1, s_2 \in nobs(\triangle)^*$, $\alpha \in Act$ et $\breve{X} \overset{s}{\rightarrow} \breve{X}'$. Or, par hypothèse, $nobs_{\triangle}(\breve{\mathcal{X}}) \subseteq nobs_{\triangle'}(\breve{\mathcal{X}})$ donc $s_1, s_2 \in nobs(\triangle')^*$. On a donc $\breve{X} \overset{s}{\rightarrow} \breve{X}'$ avec $s = s_1.\alpha.s_2 \in Act(\breve{\mathcal{X}})^*$ telle que $s_1, s_2 \in nobs(\triangle')^*$ d'où $\breve{X} \triangle' \overset{\alpha}{\Rightarrow} \breve{X}'$. □

Corollaire 26. *Soient X, X' deux éléments du même ensemble $\mathcal{X} \in \{\mathcal{P}, \mathcal{C}, \mathcal{N}\}$. Pour tout $\triangle, \triangle' \in \blacktriangle$ tels que $nobs_{\triangle}(\mathcal{X}) \subseteq nobs_{\triangle'}(\mathcal{X})$, si $X \triangle X'$ alors $X \triangle' X'$.*

Démonstration. Omise. □

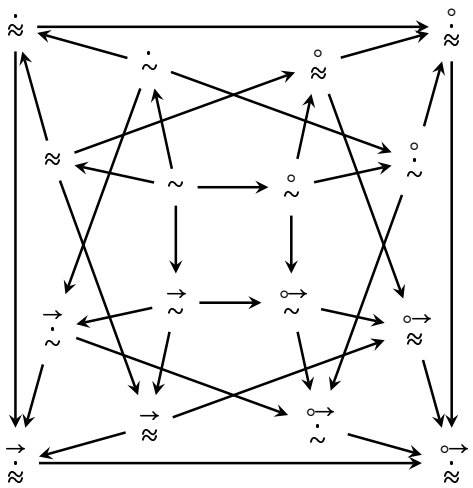


FIG. 3.10: Illustration du corollaire 26. (→ signifie implique).

Remarque : La relation d'implication est bien évidemment transitive et la clôture transitive n'est pas indiquée dans la figure 3.10. On peut aussi remarquer que plus le symbole représentant la relation d'équivalence (ensemble des éléments de ▲) est verticalement important, plus les évènements inobservables sont nombreux (et donc plus la relation est faible). La relation la plus forte est donc ~ et la plus faible est $\overset{\circ\rightarrow}{\approx}$.

Équivalences entre les relations de bisimilarité dans le cas général

La proposition suivante et son corollaire indiquent que si deux entités sont équivalentes par l'observation d'un certain nombre d'évènements, elles le sont aussi par l'observation

d'un nouvel ensemble d'évènements qui inclue le premier ensemble si la différence entre ces deux ensembles sont des évènements qui ne peuvent pas appartenir aux entités évaluées.

Proposition 27. *Soient* $\breve{X}, \breve{X}'$ *deux éléments du même ensemble* $\breve{\mathcal{X}} \in \{\breve{\mathcal{P}}, \breve{\mathcal{C}}, \breve{\mathcal{N}}\}$. *Pour tout* $\alpha \in Act$ *et pour tout* $\triangle, \triangle' \in \blacktriangle$ *tels que* $obs_{\triangle'}(\breve{\mathcal{X}}) = obs_{\triangle}(\breve{\mathcal{X}})$, $\breve{X} \triangle \overset{\alpha}{\Rightarrow} \breve{X}' \Leftrightarrow \breve{X} \triangle' \overset{\alpha}{\Rightarrow} \breve{X}'$.

Cette proposition permet de montrer des équivalences entre certaines relations de bisimilarité en fonction du niveau sur lequel elles sont utilisées.

Exemple : Pour $P, P' \in \mathcal{P}$, $obs_{\sim}(\mathcal{P}) = obs_{\vec{\sim}}(\mathcal{P})$ donc $P \sim P' \Leftrightarrow P \vec{\sim} P'$. Ceci provient du fait qu'au niveau processus ($\mathcal{P}$), il n'existe pas de transition par $\dot{\wedge}$, $\mathring{\wedge}$ et $\vec{\mathcal{A}}$.

Démonstration. Par hypothèse, $obs_{\triangle'}(\breve{\mathcal{X}}) = obs_{\triangle}(\breve{\mathcal{X}})$ donc

1. $obs_{\triangle'}(\breve{\mathcal{X}}) \subseteq obs_{\triangle}(\breve{\mathcal{X}})$ et donc par la proposition 25, si $\breve{X} \triangle \overset{\alpha}{\Rightarrow} \breve{X}'$ alors $\breve{X} \triangle' \overset{\alpha}{\Rightarrow} \breve{X}'$.
2. $obs_{\triangle}(\breve{\mathcal{X}}) \subseteq obs_{\triangle'}(\breve{\mathcal{X}})$ et donc par la proposition 25, si $\breve{X} \triangle' \overset{\alpha}{\Rightarrow} \breve{X}'$ alors $\breve{X} \triangle \overset{\alpha}{\Rightarrow} \breve{X}'$.

On a donc bien $\breve{X} \triangle \overset{\alpha}{\Rightarrow} \breve{X}' \Leftrightarrow \breve{X} \triangle' \overset{\alpha}{\Rightarrow} \breve{X}'$. □

Corollaire 28. *Soient* X, X' *deux éléments du même ensemble* $X \in \{\mathcal{P}, \mathcal{C}, \mathcal{N}\}$. *Pour tout* $\triangle, \triangle' \in \blacktriangle$ *tels que* $obs_{\triangle'}(\mathcal{X}) = obs_{\triangle}(\mathcal{X})$, $X \triangle X' \Leftrightarrow X \triangle' X'$.

Démonstration. Directe en utilisant la proposition 27 sur la définition 24. □

3.2.4.3 Equivalences de traces

Définition 29. *Soit* $\breve{U} \in \breve{\mathcal{U}}$. $Traces(\breve{U}) = \{s \in Act^* \mid \exists\ \breve{U}' \in \breve{\mathcal{U}},\ \breve{U} \overset{s}{\rightarrow} \breve{U}'\}$

Définition 30. *Soient* $U_1 \in \mathcal{U}$ *et* $U_2 \in \mathcal{U}$. U_1 *et* U_2 *sont équivalents en terme de traces, écrit* $U_1 \equiv_t U_2$ *si et seulement si* $Traces(U_1) = Traces(U_2)$.

3.3 Les processus

Dans la suite, x désigne une variable et e une valeur.

3.3.1 Grammaire

L'ensemble des processus est défini par la grammaire suivante :

$\breve{P}$	::=	0	*processus vide ou inactif*
	$\mid$	$\tau.\breve{P}$	*action inter-processus silencieuse ou parfaite*
	$\mid$	$a(x).\breve{P}$	*action de réception inter-processus*
	$\mid$	$\overline{a}(e).\breve{P}$	*action d'émission inter-processus*
	$\mid$	$\breve{P}[b/a]$	*renommage de a en b, avec* $a, b \in \mathcal{L}$
	$\mid$	$\breve{P}_1 + \breve{P}_2$	*choix*
	$\mid$	$\breve{P}_1 \mid \breve{P}_2$	*composition parallèle*
	$\mid$	$\breve{P} \setminus a$	*restriction,* $a \in \bigwedge$
	$\mid$	$\dot{\tau}.\breve{P}_1$	*action inter-unité de calcul dite silencieuse ou cachée (restriction)*
	$\mid$	$\dot{a}(x).\breve{P}$	*action de réception inter-unité de calcul*
	$\mid$	$\ddot{a}(e).\breve{P}$	*action d'émission inter-unité de calcul*
	$\mid$	$\mathring{a}\,(x).\breve{P}$	*action de réception inter-nœud*
	$\mid$	$\mathring{\overline{a}}\,(e).\breve{P}$	*action d'émission inter-nœud*
	$\mid$	$\vec{w}.\breve{P}$	*déplacement vers la position w*

La constante A est un agent dont le sens est donné par une équation de définition. Nous supposons donc que pour toute constante A, il existe une équation de la forme $A \stackrel{\text{def}}{=} P$ qui la définit.

Dans la suite, pour noter $(P \setminus a) \setminus b$, on écrira $P \setminus \{a, b\}$.

Définition 31. *Une fonction de renommage f est une application de la forme :*

$$\begin{array}{rcccc} f & : & \mathcal{L} & \rightarrow & \mathcal{L} \\ & & l & \mapsto & f(l) \end{array}$$

Pour noter $(P[b/a])[c/d]$, on notera $P[f]$ avec f une fonction de renommage. Ici, f est définie de la façon suivante :

$$f(l) = \begin{cases} b & \text{si l=a} \\ d & \text{si l=c} \\ l & \text{sinon} \end{cases}$$

Remarque : τ correspond à une action silencieuse (qui peut être une communication) au niveau des processus, $\dot{\tau}$ correspond à une action silencieuse (qui peut être "émission cachée", cf. règle P.Res page 72) au niveau des unités de calcul. Il n'existe pas de $\mathring{\tau}$ car la communication entre les nœuds (actions de l'ensemble $\mathring{\bigwedge}$) se fait en deux temps. Le premier est l'émission du message, qui le place dans l'espace des messages, le second est la réception du message depuis cet espace. Cette communication asynchrone ne fait donc pas apparaître d'évènement $\mathring{\tau}$ (qui résulterait d'une simultanéïté qui n'existe pas ici).

3.3.2 Transitions

Rappel : $\alpha \in Act$, $l \in \bigwedge = \mathcal{A} \cup \overline{\mathcal{A}}$ (pour mémoire, $\bigwedge \subset Act$)

Préfixe : P.Act $\qquad \dfrac{}{\alpha.\breve{P} \stackrel{\alpha}{\rightarrow} \breve{P}}$

Cette règle permet aux expressions agentiques de processus d'évoluer en commettant l'évènement préfixe de l'expression.

Somme : P.Sum$_j$ $$\frac{\breve{P}_j \xrightarrow{\alpha} \breve{P}'_j}{\Sigma_{i\in I}\breve{P}_i \xrightarrow{\alpha} \breve{P}'_j}\ (j \in I)$$

L'opérateur + est un opérateur de choix (branche). Si une expression agentique de processus est capable d'évoluer par α alors la somme de cette expression avec d'autres expressions agentiques peut évoluer par α, éliminant ainsi toutes les autres branches.

Composition 1 : P.Com$_1$ $$\frac{\breve{P} \xrightarrow{\alpha} \breve{P}'}{\breve{P} \mid \breve{Q} \xrightarrow{\alpha} \breve{P}' \mid \breve{Q}}$$

La composition ou mise en parallèle de processus permet à ceux-ci d'évoluer indépendamment l'un de l'autre.

Composition 2 : P.Com$_2$ $$\frac{\breve{Q} \xrightarrow{\alpha} \breve{Q}'}{\breve{P} \mid \breve{Q} \xrightarrow{\alpha} \breve{P} \mid \breve{Q}'}$$

Cette règle est la symétrique de la règle précédente (P.Com$_1$).

Composition 3 : P.Com$_3$ $$\frac{\breve{P} \xrightarrow{l} \breve{P}' \quad \breve{Q} \xrightarrow{\overline{l}} \breve{Q}'}{\breve{P} \mid \breve{Q} \xrightarrow{\tau} \breve{P}' \mid \breve{Q}'}\ (l \in \bigwedge)$$

τ est ici le résultat de la réalisation simultanée de deux actions complémentaires. Il peut-être interprêté comme une communication synchrone entre deux processus.

Restriction : P.Res $$\frac{\breve{P} \xrightarrow{\alpha} \breve{P}'}{\breve{P} \setminus L \xrightarrow{\alpha} \breve{P}' \setminus L}\ (\alpha, \overline{\alpha} \notin L)$$

Par construction, $L \subseteq \bigwedge$ (cf. Grammaire page 70). Cette règle définit une restriction au niveau des processus, restriction qui se limite à des éléments de l'ensemble $\bigwedge$ car restreindre un processus (en dehors d'une unité de calcul) sur un élément de $\dot{\bigwedge}$ ou $\mathring{\bigwedge}$ n'aurait pas de sens[4]. La restriction permet d'imposer une synchronisation (communication blocante) au niveau des processus.

[4]Les restrictions sur les actions de $\dot{\bigwedge}$ se font au niveau des unités de calcul par les règles C.Res$_1$ et C.Res$_2$ (cf. page 79).

Renommage : P.Rel $\dfrac{\breve{P} \overset{\alpha}{\to} \breve{P}'}{\breve{P}[f] \overset{f(\alpha)}{\to} \breve{P}'[f]}$

Cette règle renomme l'ensemble des étiquettes l de $\breve{P}$ en $f(l)$.

Constantes : P.Con $\dfrac{\breve{P} \overset{\alpha}{\to} \breve{P}'}{A \overset{\alpha}{\to} \breve{P}'}\ (A \overset{\text{def}}{=} \breve{P})$

Cette règle définit que la constante A possède les mêmes transitions que l'expression agentique qui la définit.

3.3.3 Sorte

Proposition 32. *Pour tout $\breve{P}$ et $L \subseteq \mathcal{L}$, L est une sorte de $\breve{P}$ si et seulement si pour tout α tel que $\breve{P} \overset{\alpha}{\to} \breve{P}'$ alors*

1. $\alpha \in L \cup \{\tau, \dot{\tau}\}$
2. *L est une sorte de $\breve{P}'$*

Démonstration. Triviale. □

Chaque agent possède une sorte minimale mais qui n'est pas toujours simple ni même possible à déterminer. En effet, si une étiquette appartient à P de manière syntaxique, seule l'exécution permet de déterminer si P peut effectuer l'action correspondante. Ce problème est indécidable (cf. CCS [Mil89]).

Définition 33. *La sorte syntaxique de l'expression agentique $\breve{P}$ notée $\mathcal{L}(\breve{P})$ (que nous appelons parfois simplement sorte de $\breve{P}$) est définie comme suit :*

$$\begin{array}{lcl}
\mathcal{L}(0) & = & \varnothing \\
\mathcal{L}(l.\breve{P}) & = & \{l\} \cup \mathcal{L}(\breve{P})\ \ \forall l \in \mathcal{L} \\
\mathcal{L}(\tau.\breve{P}) & = & \mathcal{L}(\breve{P}) \\
\mathcal{L}(\dot{\tau}.\breve{P}) & = & \mathcal{L}(\breve{P}) \\
\mathcal{L}(\sum_i \breve{P}_i) & = & \bigcup_i \mathcal{L}(\breve{P}_i) \\
\mathcal{L}(\breve{P} \mid \breve{Q}) & = & \mathcal{L}(\breve{P}) \cup \mathcal{L}(\breve{Q}) \\
\mathcal{L}(\breve{P} \setminus L) & = & \mathcal{L}(\breve{P}) - (L \cup \overline{L}) \\
\mathcal{L}(\breve{P}[f]) & = & \{f(l) \mid l \in \mathcal{L}(\breve{P})\}
\end{array}$$

De plus, pour toute équation de définition d'une constante $A \overset{def}{=} P$, nous définissons que la sorte associée à cette constante est telle que $\mathcal{L}(P) \subseteq \mathcal{L}(A)$.

Nous devons maintenant montrer que la fonction $\mathcal{L}(\breve{P})$ de la définition 33 que nous avons définie est bien une sorte de $\breve{P}$. Pour prouver cela nous montrons tout d'abord la proposition suivante.

Proposition 34. *Soit* $\breve{P} \xrightarrow{\alpha} \breve{P}'$. *Alors*

1. $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$
2. $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$

Démonstration. Par induction sur les transitions de la forme $\breve{P} \xrightarrow{\alpha} \breve{P}'$. L'hypothèse d'induction est que la proposition est vraie pour $\breve{P}'$ qui possède n constructeurs. Soit $\breve{P}$ un processus qui possède $n+1$ constructeurs.

Cas 1. Par P.Act. $\breve{P} \equiv \alpha.\breve{P}'$ donc $\breve{P} \xrightarrow{\alpha} \breve{P}'$. Or, d'après la définition 33, $\mathcal{L}(\breve{P}) = \{\alpha\} \cup \mathcal{L}(\breve{P}')$ et donc

1. $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$
2. $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$

Cas 2. Par P.Sum$_j$. $\breve{P} \equiv \Sigma_i \breve{P}_i$, et $\breve{P}_j \xrightarrow{\alpha} \breve{P}'$ donc il existe j tel que $\breve{P}_j \xrightarrow{\alpha} \breve{P}'$. On a donc $\breve{P}_j$ qui satisfait par induction la proposition 34 et donc

1. $\alpha \in \mathcal{L}(\breve{P}_j) \cup \{\tau, \dot{\tau}\}$
2. $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P}_j)$

Or, par la définition 33, $\mathcal{L}(\breve{P}) = \bigcup_i \mathcal{L}(\breve{P}_i)$. Par conséquent :

– (1) : $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$
– et (2) : $\mathcal{L}(\breve{P}_j) \subseteq \mathcal{L}(\breve{P})$

Cas 3. Par P.Com$_1$. $\breve{P} \equiv \breve{P}_1 \mid \breve{P}_2$ donc $\breve{P}_1 \mid \breve{P}_2 \xrightarrow{\alpha} \breve{P}_1' \mid \breve{P}_2$. Par induction, $\breve{P}_1$ satisfait la proposition 34. Comme $\mathcal{L}(\breve{P}) = \mathcal{L}(\breve{P}_1) \cup \mathcal{L}(\breve{P}_2)$ (par la définition 33), on a bien $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$ et $\mathcal{L}(\breve{P}_1 \mid \breve{P}_2) \subseteq \mathcal{L}(\breve{P})$.

Cas 4. Par P.Com$_2$. Similaire.

Cas 5. Par P.Com$_3$. Similaire.

Cas 6. Par P.Res. $\breve{P} \xrightarrow{\alpha} \breve{P}'$ et $\alpha, \overline{\alpha} \notin L$ donc $\breve{P} \setminus L \xrightarrow{\alpha} \breve{P}' \setminus L$. Par conséquent, d'après la définition 33, $\mathcal{L}(\breve{P}) = \mathcal{L}(\breve{P}') - (L \cup \overline{L})$. Or, par induction,

1. $\alpha \in \mathcal{L}(\breve{P}') \cup \{\tau, \dot{\tau}\}$
2. $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$

Comme $\alpha, \overline{\alpha} \notin L \cup \overline{L}$ par hypothèse, on a bien

– (1) : $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$
– et (2) : $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$

Cas 7. Par P.Rel. Similaire.

Cas 8. Par P.Con. $A \stackrel{\text{def}}{=} P$ tel que $P \xrightarrow{\alpha} P'$. Alors, par induction, 34,

1. $\alpha \in \mathcal{L}(\breve{P}') \cup \{\tau, \dot{\tau}\}$
2. $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$

Or, comme la définition 33 impose que $\mathcal{L}(\breve{P}) \subseteq \mathcal{L}(A)$, on a bien

– $\alpha \in \mathcal{L}(A) \cup \{\tau, \dot{\tau}\}$
– $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(A)$

□

Corollaire 35. $\breve{P} : \mathcal{L}(\breve{P})$

Démonstration. Nous allons montrer que $\mathcal{L}(\breve{P})$ satisfait la définition 14. Soit α une action d'une dérivée $\breve{P}'$ de $\breve{P}$. Par itération de la proposition 34(2), on a $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$. De plus, grâce à la proposition 34(1), on a $\alpha \in \mathcal{L}(\breve{P}') \cup \{\tau, \dot{\tau}\}$, et ainsi $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$. □

3.3.4 Congruence observationnelle

Définition 36. *Un contexte de processus C_P est, informellement parlant, une expression de processus avec un trou représenté par []. Plus formellement, un contexte de processus est grammaticalement défini par :*

$$C_P \quad ::= \quad [] \quad \Big| \quad \alpha.C_P + P \quad \Big| \quad C_P \setminus a \quad \Big| \quad C_P \mid P \quad \Big| \quad P \mid C_P$$

Définition 37. *Soient P_1 et P_2 deux éléments de $\mathcal{P}$ et $\mathcal{R}$ une relation d'équivalence (c'est à dire réflexive, symétrique et transitive) sur $\mathcal{P} \times \mathcal{P}$. On dit que $\mathcal{R}$ est une congruence sur $\mathcal{P} \times \mathcal{P}$ si pour tout élément $P_1, P_2 \in \mathcal{P}$, et pour tout contexte de processus C_P, $P_1 \mathcal{R} P_2 \Rightarrow C_P[P_1] \mathcal{R} C_P[P_2]$. On dit alors que P_1 et P_2 sont égaux modulo $\mathcal{R}$ ou (observationnellement) congruents modulo $\mathcal{R}$.*

Proposition 38. *Si P_1 et P_2 sont des éléments de $\mathcal{P}$ tels que pour tout $\alpha \in Act(\mathcal{P})$,*
– si $P_1 \xrightarrow{\alpha} P_1'$ alors, il existe P_2' tel que $P_2 \approx\!\!\Rightarrow^{\alpha} P_2'$ et $P_1' \approx P_2'$
– si $P_2 \xrightarrow{\alpha} P_2'$ alors, il existe P_1' tel que $P_1 \approx\!\!\Rightarrow^{\alpha} P_1'$ et $P_1' \approx P_2'$
alors P_1 et P_2 sont égaux.

Démonstration. Par mimétisme, cf. [Mil89]. □

Proposition 39. $\sim \Rightarrow = \Rightarrow \approx$.

Démonstration. Par mimétisme, cf. [Mil89]. □

3.3.5 Relations entre processus

Les propriétés CCS qui portent sur les agents restent vraies dans notre modèle au niveau des processus agents. Autrement dit, si $P \in \mathcal{P}$ alors P est un processus au sens CCS, c'est à dire que $\mathcal{L}(P) \subseteq \mathbf{Act}, \mathbf{Act}$ tel que défini par R. Milner [Mil89] et que la grammaire et les règles de transition sont à ce niveau les même qu'en CCS. Il ne nous est donc pas nécessaire de démontrer à nouveau toutes les propriétés que R. Milner a déjà démontrées pour CCS.

En particulier, les propriétés suivantes restent vraies dans CiMAN.

Proposition 40. *(the monoid laws)*

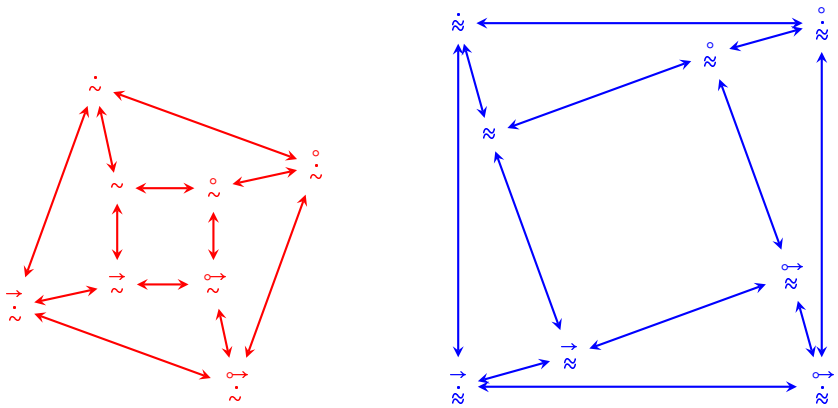
1. $P + Q \sim Q + P$
2. $P + (Q + R) \sim (P + Q) + R$
3. $P + P \sim P$
4. $P + 0 \sim P$

Proposition 41. *(the static laws)*

1. $P \mid Q \sim Q \mid P$
2. $P \mid (Q \mid R) \sim (P \mid Q) \mid R$
3. $P \mid 0 \sim P$
4. $P \setminus L \sim P$ *si* $\mathcal{L}(P) \cap (L \cup \overline{L}) = \varnothing$
5. $P \setminus K \setminus L \sim P \setminus K \cup L$
6. $P[f] \setminus L \sim P \setminus f^{-1}(L)[f]$
7. $(P \mid Q) \setminus L \sim P \setminus L \mid Q \setminus L$ *si* $\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \overline{L}) = \varnothing$
8. $P[Id] \sim P$

3.3.5.1 Équivalences entre les relations de bisimilarité dans $\mathcal{P} \times \mathcal{P}$.

Au niveau processus seul, le seul évènement qui peut ne pas être observé est τ. Les bisimulations (cf. définition 18) qui considèrent les évènements τ sont donc équivalentes (cf. figure 3.11a). Symétriquement, les bisimulations qui ne considèrent pas les évènements τ sont équivalentes (cf. figure 3.11b).



(a) τ observable (b) τ pas observable

FIG. 3.11: Illustration du corollaire 28 au niveau processus seul.

3.3.5.2 Implications entre les relations de bisimilarité dans $\mathcal{P} \times \mathcal{P}$.

La figure 3.12 est l'union des figures 3.10 et 3.11. Elle illustre les implications entre les relations de bisimilarité des corollaires 26 et 28 (i.e. ici $P \vartriangle P' \Rightarrow P \tilde{\vartriangle} P'$). Elle représente l'ensemble des équivalences et implications sur les bisimulations au niveau processus seul.

3.4 Les unités de calcul

Dans la suite, x désigne une variable, e est une valeur.

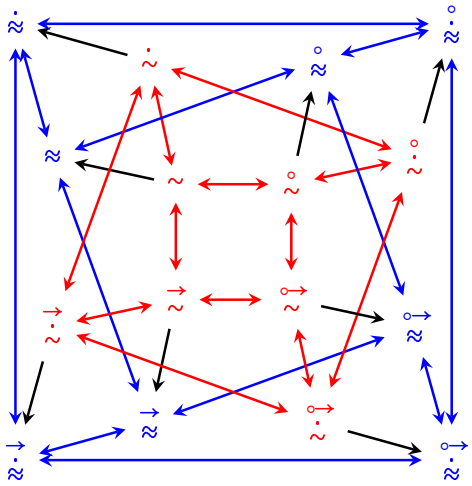


Fig. 3.12: Résumé des implications entre les relations de bisimilarité dans $\mathcal{P} \times \mathcal{P}$.

3.4.1 Grammaire

$\breve{C}$	::=	$[\breve{P}]$	*unité de calcul exécutant le processus* P
	|	$\breve{C}[b/a]$	*renommage de* a *en* b*, avec* $a, b \in \mathcal{L}$
	|	$\breve{C}_1 \parallel \breve{C}_2$	*composition parallèle*
	|	$\breve{C} \setminus a$	*restriction,* $a \in \bigwedge \cup \dot{\bigwedge}$

3.4.2 Transitions

Rappel : $\alpha \in Act$, $\dot{l} \in \dot{\mathcal{A}}$ et $\ddot{l} \in \ddot{\mathcal{A}}$

Préfixe : C.Act $\quad \dfrac{\breve{P} \xrightarrow{\alpha} \breve{P}'}{[\breve{P}] \xrightarrow{\alpha} [\breve{P}']} \ (\alpha \notin \dot{\mathcal{A}})$

Si un processus embarqué dans une unité de calcul évolue par une transition, alors il peut évoluer de la même façon au sein de l'unité de calcul qui l'embarque sauf si l'action appartient à l'ensemble $\dot{\mathcal{A}}$. Dans, la suite, nous parlerons de l'évolution de l'unité de calcul pour parler de l'évolution du processus embarqué dans l'unité de calcul. Un processus qui émet sur un fil peut évoluer tout seul seulement s'il n'existe pas de processus qui peut recevoir dans les unités de calcul mises en parallèle de la sienne (ceci est imposé par C.Com_1 et C.Com_2).

Composition 1 : C.Com_1 $\quad \dfrac{\breve{C} \xrightarrow{\alpha} \breve{C}'}{\breve{C} \parallel [\breve{P}] \xrightarrow{\alpha} \breve{C}' \parallel [\breve{P}]} \ (\alpha \in \ddot{\mathcal{A}} \Rightarrow \breve{P} \overset{\dot{\alpha}}{\nrightarrow})$

La composition parallèle permet à des unités de calcul d'évoluer indépendamment les

unes des autres. Si l'étiquette de la transition appartient à $\ddot{\mathcal{A}}$ alors le processus ne peut évoluer que s'il n'y a pas d'unité de calcul en parallèle qui est dans la capacité de recevoir. Autrement dit, un processus en écoute sur un bus (action $\alpha \in \dot{\mathcal{A}}$) en parallèle d'un processus qui émet sur ce bus (action $\dot{\alpha} \in \ddot{\mathcal{A}}$) depuis une autre unité de calcul reçoit instantanément ce message. En revanche, si personne n'écoute alors l'émission par $\alpha \in \ddot{\mathcal{A}}$ est non blocante. Cette opération modélise une communication sur un bus sur lequel, lors d'une émission, les entités connectées et en écoute reçoivent le message émis. On considère qu'il n'y a pas de collision ou plutôt qu'il existe un mécanisme (que nous ne modélisons pas) qui permet d'éviter les collisions, comme en CBS [Pra93, Pra95]. Cependant, les différences notables avec ce dernier sont que (i) CiMAN propose des canaux de communication plutôt qu'un espace unique ou un bus global, (ii) nous ne supposons pas que les processus sont capables de recevoir à chaque instant.

Il est à noter que les actions de l'ensemble $\dot{\bigwedge}$ ne permettent pas la communication entre les processus d'une même unité de calcul. Par exemple, un message envoyé sur un canal $\ddot{a}$ ne peut jamais être reçu par un processus qui écoute sur $\dot{a}$ si ce processus se situe sur la même unité de calcul que le processus émetteur.

Composition 2 : C.Com$_2$
$$\frac{\breve{C} \xrightarrow{\alpha} \breve{C}'}{[\breve{P}] \parallel \breve{C} \xrightarrow{\alpha} [\breve{P}] \parallel \breve{C}'} \ (\alpha \in \ddot{\mathcal{A}} \Rightarrow \breve{P} \overset{\dot{\alpha}}{\not\rightarrow})$$

Cette règle est symétrique de la règle précédente.

Composition 3 : C.Com$_3$
$$\frac{\breve{C} \xrightarrow{\ddot{l}} \breve{C}' \qquad \breve{P} \xrightarrow{\dot{l}} \breve{P}'}{\breve{C} \parallel [\breve{P}] \xrightarrow{\ddot{l}} \breve{C}' \parallel [\breve{P}']}$$

Cette règle définit l'opération de *broadcast*. Contrairement aux actions du niveau processus, $\breve{C}_1 \xrightarrow{\ddot{a}(v)} \breve{C}_1'$ n'implique pas forcément que $\breve{C}_1 \parallel \breve{C}_2 \xrightarrow{\ddot{a}(v)} \breve{C}_1' \parallel \breve{C}_2$. En effet, l'équivalent des actions du niveau processus ne peut se produire que si personne n'attend en réception (cf. C.Com$_1$ et C.Com$_2$). La dérivation ci-dessous illustre les communications de type *one-to-many* :

$$\frac{\dfrac{\breve{C}_1 \xrightarrow{\ddot{a}} \breve{C}_1' \qquad \breve{P}_2 \xrightarrow{\dot{a}} \breve{P}_2'}{\breve{C}_1 \parallel [\breve{P}_2] \xrightarrow{\ddot{a}} \breve{C}_1' \parallel [\breve{P}_2'] \qquad \breve{P}_3 \xrightarrow{\dot{a}} \breve{P}_3'}}{(\breve{C}_1 \parallel [\breve{P}_2]) \parallel [\breve{P}_3] \xrightarrow{\ddot{a}} (\breve{C}_1' \parallel [\breve{P}_2']) \parallel [\breve{P}_3']}$$

Le schéma ci-dessous présente en noir les transitions possibles des processus et en rouge les transitions possibles des unités de calcul.

C_1		C_2		C_3	
$[\ddot{a}.\breve{P}]$	$\|\|$	$[\dot{a}.\breve{Q}]$	$\|\|$	$[\dot{a}.\breve{R}]$	
$\ddot{a}\downarrow\downarrow\ddot{a}$		$\downarrow\dot{a}$		$\downarrow\dot{a}$	La seule règle applicable est C.Com$_3$
$C_1 \,\|\|\, C_2$			$\|\|$	$[\dot{a}.\breve{Q}]$	
$\downarrow\ddot{a}$				$\downarrow\dot{a}$	La seule règle applicable est C.Com$_3$
$C_1 \,\|\|\, C_2 \,\|\|\, C_3$					
$\downarrow\ddot{a}$					La seule règle applicable est C.Com$_3$
$[\breve{P}] \,\|\|\, [\breve{Q}] \,\|\|\, [\breve{R}]$					

Toutes les unités de calcul en écoute sur un canal reçoivent donc simultanément un message émis sur ce canal. Notons que si une unité de calcul peut émettre sur un fil sans se bloquer si personne n'est dans l'état de recevoir. Dès qu'une réception est possible alors l'émission devient bloquante et ne peut avoir lieu sans la réception correspondante.

Composition 4 : C.Com$_4$ $\dfrac{\breve{P} \xrightarrow{\dot{l}} \breve{P}' \quad \breve{C} \xrightarrow{\ddot{l}} \breve{C}'}{[\breve{P}] \,||\, \breve{C} \xrightarrow{\ddot{l}} [\breve{P}'] \,||\, \breve{C}'}$

Cette règle est symétrique de la règle précédente.

Restriction : C.Res$_1$ $\dfrac{\breve{C} \xrightarrow{\alpha} \breve{C}'}{\breve{C} \setminus L \xrightarrow{\alpha} \breve{C}' \setminus L}\ (\alpha \notin L' = L_1 \cup \overline{L}_1 \cup \dot{L}_1,\ L_1 = L \cap \bigwedge \text{ et } \dot{L}_1 = L \cap \ddot{\mathcal{A}})$

Cette règle permet de définir la transition induite par une action qui n'appartient pas à l'ensemble $L \subseteq \bigwedge \cup \dot{\bigwedge}$ des restrictions. Une action $\alpha \in \mathit{Act}$ fait évoluer l'unité de calcul par la transition α si seulement si $\alpha \notin L$.

Restriction : C.Res$_2$ $\dfrac{\breve{C} \xrightarrow{\alpha} \breve{C}'}{\breve{C} \setminus L \xrightarrow{\dot{\tau}} \breve{C}' \setminus L}\ (\alpha \in L \wedge \alpha \in \ddot{\mathcal{A}})$

Cette règle restreint une action d'émission aux unités de calcul qui composent $\breve{C}$. Seules les actions appartenant à l'ensemble $\ddot{\mathcal{A}}$ sont concernées par cette restriction. L'action d'émission $\alpha \in \ddot{\mathcal{A}}$ fait évoluer le système par la transition $\dot{\tau}$, stoppant ainsi la diffusion du message aux unités de calcul autres que celles qui composent $\breve{C}$. Cette opération définit un "fil/bus blindé" entre plusieurs unités de calcul.

Renommage : C.Rel $\dfrac{\breve{C} \xrightarrow{\alpha} \breve{C}'}{\breve{C}[f] \xrightarrow{f(\alpha)} \breve{C}'[f]}$

Cette règle renomme l'ensemble des étiquettes l de $\breve{C}$ en $f(l)$.

3.4.3 Sorte

Proposition 42. *Pour tout $\breve{C}$ et $L \subseteq \mathcal{L}$, L est une sorte de $\breve{C}$ si et seulement si pour tout α tel que $\breve{C} \xrightarrow{\alpha} \breve{C}'$ alors*

1. *$\alpha \in L \cup \{\tau, \dot{\tau}\}$*
2. *L est une sorte de $\breve{C}'$*

Démonstration. Triviale. □

Définition 43. *La sorte syntaxique de l'expression agentique $\breve{C}$ notée $\mathcal{L}(\breve{C})$ (que nous appelons parfois simplement sorte de $\breve{C}$) est définie comme suit :*

$$\begin{array}{lcl} \mathcal{L}([\breve{P}]) & = & \mathcal{L}(\breve{P}) \\ \mathcal{L}(\breve{C}_1 \parallel \breve{C}_2) & = & \mathcal{L}(\breve{C}_1) \cup \mathcal{L}(\breve{C}_2) \\ \mathcal{L}(\breve{C} \setminus L) & = & \mathcal{L}(\breve{C}) - L' \text{ avec } L' = L_1 \cup \overline{L}_1 \cup \dot{L}_1,\ L_1 = L \cap \bigwedge \text{ et } \dot{L}_1 = L \cap \ddot{\mathcal{A}} \\ \mathcal{L}(\breve{C}[f]) & = & \{f(l) \mid l \in \mathcal{L}(\breve{C})\} \end{array}$$

Nous devons maintenant montrer que la fonction $\mathcal{L}(\breve{C})$ de la définition 43 que nous avons définie est bien une sorte de $\breve{C}$. Pour prouver cela nous montrons tout d'abord la proposition suivante.

Proposition 44. *Soit $\breve{C} \xrightarrow{\alpha} \breve{C}'$. Alors*

1. *$\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$*
2. *$\mathcal{L}(\breve{C}') \subseteq \mathcal{L}(\breve{C})$*

Démonstration. Par induction sur les transitions de la forme $\breve{C} \xrightarrow{\alpha} \breve{C}'$. La proposition est vraie pour $\breve{C} = [0]$. L'hypothèse d'induction est que la proposition est vraie pour $\breve{C}'$ qui possède n constructeurs. Soit $\breve{C}$ une unité de calcul qui possède $n+1$ constructeurs. Montrons que la proposition est vraie pour $\breve{C}$.

Cas 1. $\breve{C} \equiv [\breve{P}]$ et $\breve{P} \xrightarrow{\alpha} \breve{P}'$ (C.Act)

Par la définition 43 on a $\mathcal{L}(\breve{C}) = \mathcal{L}(\breve{P})$ (i)

et par induction

1. $\alpha \in \mathcal{L}(\breve{P}) \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{P}') \subseteq \mathcal{L}(\breve{P})$ (iii)

donc

1. $\alpha \in \mathcal{L}([\breve{P}])$ par (i)
2. $\begin{array}{lcll} \mathcal{L}([\breve{P}']) & = & \mathcal{L}(\breve{P}') & \text{par la définition 43} \\ & \subseteq & \mathcal{L}(\breve{P}) & \text{par (iii)} \\ & \subseteq & \mathcal{L}([\breve{P}]) & \text{par (i)} \end{array}$

Cas 2. $\breve{C} \equiv \breve{C}' \parallel [\breve{P}']$ et $\breve{C}' \xrightarrow{\alpha} \breve{C}''$ (C.Com$_1$)
Par la définition 43 on a $\mathcal{L}(\breve{C}) = \mathcal{L}(\breve{C}') \cup \mathcal{L}([\breve{P}'])$ (i)
et par induction

1. $\alpha \in \mathcal{L}(\breve{C}') \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}'') \subseteq \mathcal{L}(\breve{C}')$ (iii)

donc

1. $\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$ par (i) et (ii)
2. $\begin{array}{lcll} \mathcal{L}(\breve{C}'' \parallel [\breve{P}']) & = & \mathcal{L}(\breve{C}'') \cup \mathcal{L}([\breve{P}']) & \text{par la définition 43} \\ & \subseteq & \mathcal{L}(\breve{C}') \cup \mathcal{L}([\breve{P}']) & \text{par (iii)} \\ & \subseteq & \mathcal{L}(\breve{C}' \parallel [\breve{P}']) & \text{par (i)} \end{array}$

Cas 3. $\breve{C} \equiv [\breve{P}'] \mid\mid \breve{C}'$ et $\breve{C}' \xrightarrow{\alpha} \breve{C}''$ (C.Com$_2$) : Similaire.

Cas 4. $\breve{C} \equiv \breve{C}' \parallel [\breve{P}']$ et $\breve{C}' \xrightarrow{\ddot{l}} \breve{C}''$ et $\breve{P}' \xrightarrow{\dot{l}} \breve{P}''$ (C.Com$_3$)
Par la définition 43 on a $\mathcal{L}(\breve{C}) = \mathcal{L}(\breve{C}') \cup \mathcal{L}([\breve{P}'])$ (i)
et par induction

1. $\ddot{l} \in \mathcal{L}(\breve{C}') \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}'') \subseteq \mathcal{L}(\breve{C}')$ (iii)
3. $\mathcal{L}([\breve{P}'']) \subseteq \mathcal{L}([\breve{C}'])$ (iv)

donc

1. $\ddot{l} \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$ par (i) et (ii)
2. $\begin{array}{lcll} \mathcal{L}(\breve{C}'' \parallel [\breve{P}'']) & = & \mathcal{L}(\breve{C}'') \cup \mathcal{L}([\breve{P}'']) & \text{par la définition 43} \\ & \subseteq & \mathcal{L}(\breve{C}') \cup \mathcal{L}([\breve{P}']) & \text{par (iii) et (iv)} \\ & \subseteq & \mathcal{L}(\breve{C}' \parallel [\breve{P}']) & \text{par (i)} \end{array}$

Cas 5. $\breve{C} \equiv [\breve{P}'] \mid\mid \breve{C}'$ et $\breve{P}' \xrightarrow{\dot{l}} \breve{P}''$ et $\breve{C}' \xrightarrow{\ddot{l}} \breve{C}''$ (C.Com$_4$) : Similaire.

Cas 6. $\breve{C} \equiv \breve{C}' \setminus_L$ et $\breve{C}' \xrightarrow{\alpha} \breve{C}''$ (C.Res$_1$)
Par la définition 43 on a $\mathcal{L}(\breve{C}) = \mathcal{L}(\breve{C}') - L'$ avec $L' = L_1 \cup \overline{L}_1 \cup \dot{L}_1$, $L_1 = L \cap \bigwedge$ et $\dot{L}_1 = L \cap \ddot{\mathcal{A}}$) (i)
et par induction

1. $\alpha \in \mathcal{L}(\breve{C}') \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}'') \subseteq \mathcal{L}(\breve{C}')$ (iii)

donc

1. $\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$ par (i)
2. $\begin{array}{lcll} \mathcal{L}(\breve{C}'' \setminus_L) & = & \mathcal{L}(\breve{C}'') - L' & \\ & \text{avec} & L' = L_1 \cup \overline{L}_1 \cup \dot{L}_1,\ L_1 = L \cap \bigwedge \text{ et } \dot{L}_1 = L \cap \ddot{\mathcal{A}} & \text{par la définition 43} \\ & \subseteq & \mathcal{L}(\breve{C}') - L' & \text{par (iii)} \\ & \subseteq & \mathcal{L}(\breve{C}' \setminus_L) & \text{par (i)} \end{array}$

Cas 7. $\breve{C} \equiv \breve{C}' \setminus\!\!\setminus_L$ et $\breve{C}' \xrightarrow{\alpha} \breve{C}''$ (C.Res$_2$)

Par la définition 43 on a $\mathcal{L}(\breve{C}) = \mathcal{L}(\breve{C}') - L'$ avec $L' = L_1 \cup \overline{L}_1 \cup \dot{L}_1$, $L_1 = L \cap \bigwedge$ et $\dot{L}_1 = L \cap \ddot{\mathcal{A}}$) (i)

et par induction

1. $\alpha \in \mathcal{L}(\breve{C}') \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}'') \subseteq \mathcal{L}(\breve{C}')$ (iii)

donc

1. $\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$ par (i)
2. $\mathcal{L}(\breve{C}'' \setminus\!\!\setminus_L) = \mathcal{L}(\breve{C}'') - L'$

   avec $L' = L_1 \cup \overline{L}_1 \cup \dot{L}_1$, $L_1 = L \cap \bigwedge$ et $\dot{L}_1 = L \cap \ddot{\mathcal{A}}$ par la définition 43

   $\subseteq \mathcal{L}(\breve{C}') - L'$ par (iii)

   $\subseteq \mathcal{L}(\breve{C}' \setminus\!\!\setminus_L)$ par (i)

Cas 8. $\breve{C} \equiv \breve{C}' \setminus\!\!\setminus [f]$ et $\breve{C}' \xrightarrow{\alpha} \breve{C}''$ (C.Rel) : Similaire.

□

Corollaire 45. $\breve{C} : \mathcal{L}(\breve{C})$

Démonstration. Nous allons montrer que $\mathcal{L}(\breve{C})$ satisfait la définition 14. Soit α une action d'une dérivée $\breve{C}'$ de $\breve{C}$. Par itération de la proposition 44(2), on a $\mathcal{L}(\breve{C}') \subseteq \mathcal{L}(\breve{C})$. De plus, grâce à la proposition 44(1), on a $\alpha \in \mathcal{L}(\breve{C}') \cup \{\tau, \dot{\tau}\}$, et ainsi $\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$. □

Corollaire 46. *Si $P \in \mathcal{P}$ alors $\mathcal{L}([P]) \subseteq \bigwedge$.*

Démonstration. Par la définition 43, pour tout $P \in \mathcal{P}$, $\mathcal{L}([P]) = \mathcal{L}(P)$. Or, par la définition 10, $\mathcal{L}(P) \subseteq Act(\mathcal{P}) = \bigwedge \cup \{\tau\}$. Or $\tau \notin \mathcal{L}(P)$ donc $\mathcal{L}(P) \subseteq \bigwedge$. □

3.4.4 Congruence observationnelle

Définition 47. *Un contexte d'unité de calcul C_C est, informellement parlant, une expression d'unité de calcul avec un trou représenté par []. Plus formellement, un contexte d'unité de calcul est grammaticalement défini par :*

$$C_C \quad ::= \quad [] \;\Big|\; C_C \parallel C \;\Big|\; C \parallel C_C \;\Big|\; C_C \setminus\!\!\setminus \ddot{a}$$

Définition 48. *Soient C_1 et C_2 deux éléments de $\mathcal{C}$ et $\mathcal{R}$ une relation d'équivalence (c'est à dire réflexive, symétrique et transitive) sur $\mathcal{C} \times \mathcal{C}$. On dit que $\mathcal{R}$ est une congruence sur $\mathcal{C} \times \mathcal{C}$ si pour tout élément $C_1, C_2 \in \mathcal{C}$, et pour tout contexte d'unité de calcul C_C, $C_1 \; \mathcal{R} \; C_2 \Rightarrow C_C[C_1] \; \mathcal{R} \; C_C[C_2]$. On dit alors que C_1 et C_2 sont égaux modulo $\mathcal{R}$ ou congruents modulo $\mathcal{R}$.*

Définition 49. *Pour tout $C_1, C_2 \in \mathcal{C}$, on dit que $C_1 \dot{\approx}_C C_2$ si et seulement si $C_1 \dot{\approx} C_2$ et pour tout $C' \in \mathcal{C}$, $C_1 \parallel C' \dot{\approx} C_2 \parallel C'$.*

Proposition 50. *$\dot{\approx}_C$ est une congruence sur $\mathcal{C} \times \mathcal{C}$.*

Démonstration. On souhaite démontrer que si C_1 et C_2 sont deux éléments de $\mathcal{C}$ alors, pour tout contexte C_C, $C_1 \dot{\approx}_C C_2 \Rightarrow C_C[C_1] \dot{\approx}_C C_C[C_2]$.
Pour $C_C = []$ on a bien $C_C[C_1] \dot{\approx}_C C_C[C_2]$.
On suppose que pour tout C'_C comportant moins de constructeurs que C_C on a $C_1 \dot{\approx}_C C_2 \Rightarrow C'_C[C_1] \dot{\approx} C'_C[C_2]$.

1. Si $C_C = C'_C \parallel C$ alors, par définition de $\dot{\approx}_C$, on a bien $C_1 \dot{\approx}_C C_2 \Rightarrow C_C[C_1] \dot{\approx} C_C[C_2]$.
2. Symétriquement, si $C_C = C \parallel C'_C$ alors $C_1 \dot{\approx}_C C_2 \Rightarrow C_C[C_1] \dot{\approx} C_C[C_2]$.
3. De manière évidente, si $C_C = C'_C \setminus_{\ddot{a}}$ alors $C_1 \dot{\approx}_C C_2 \Rightarrow C_C[C_1] \dot{\approx} C_C[C_2]$.

□

3.4.5 Relations entre processus et unités de calcul

Proposition 51. *S'il existe une transition pour une expression agentique de processus alors cette transition existe aussi pour l'unité de calcul qui l'embarque, et réciproquement. Pour tout $P \in \mathcal{P}$ et $\alpha \in Act$, $P \xrightarrow{\alpha} P' \Leftrightarrow [P] \xrightarrow{\alpha} [P']$.*

Démonstration.

1. ($\Rightarrow$) $P \in \mathcal{P}$ donc $\alpha \in Act(\mathcal{P})$ donc si $P \xrightarrow{\alpha} P'$ alors $[P] \xrightarrow{\alpha} [P']$ (C.Act).
2. ($\Leftarrow$) Par l'absurde. Supposons que $\neg(\forall \alpha \in Act,\ [P] \xrightarrow{\alpha} [P'] \Rightarrow P \xrightarrow{\alpha} P')$, c'est à dire $(\exists\ \alpha \in Act,\ [P] \xrightarrow{\alpha} [P'] \wedge P \overset{\alpha}{\nrightarrow} P')$. Notons aussi que $P \in \mathcal{P}$ donc d'après le corollaire 46, $\mathcal{L}([P]) \subseteq \bigwedge$. Les transitions qui permettent aux unités de calcul d'évoluer sont les règles C.$*$, donc au moins une de ces règles doit permettre cette transition sur les unités de calcul sans la même transition sur les processus. Nous examinons donc les règles d'évolution des unités de calcul pour déterminer si une telle transition existe.
   (a) C.Act n'est évidemment pas cette règle car la condition d'inférence est $P \xrightarrow{\alpha} P'$.
   (b) C.Com$_1$ n'est pas non plus cette règle car $\alpha \in \bigwedge$ et donc la condition d'inférence dépend directement de l'application de la règle C.Act, ce qui n'est pas possible (cf. (a)).
   (c) C.Com$_2$. Similaire.
   (d) C.Com$_3$ ne peut pas être appliquée car $\dot{l}$ et $\ddot{l}$ ne sont pas des éléments de $\bigwedge$
   (e) C.Com$_4$. Similaire.
   (f) C.Res$_1$. Tout comme pour les règles C.Com$_1$ et C.Com$_2$, la condition d'inférence dépend directement de l'application de la règle C.Act, ce qui n'est pas possible (cf. (a)).
   (g) C.Res$_2$. Cette règle n'est pas applicable car $\alpha \in \bigwedge$ implique $\alpha \notin \ddot{\mathcal{A}}$.
   (h) C.Rel. Tout comme pour la règle R.Res$_1$, la condition d'inférence dépend directement de l'application de la règle C.Act, ce qui n'est pas possible (cf. (a)).

Il n'existe donc pas $\alpha \in Act$ tel que $[P] \xrightarrow{\alpha} [P']$ et $P \overset{\alpha}{\nrightarrow} P'$. Nous avons donc montré que $\forall \alpha \in Act,\ [P] \xrightarrow{\alpha} [P'] \Leftrightarrow P \xrightarrow{\alpha} P'$. □

Proposition 52. *Pour tout $P \in \mathcal{P}$ et $s \in Act^*$, $P \xrightarrow{s} P' \Leftrightarrow [P] \xrightarrow{s} [P']$.*

Démonstration. Par récurrence sur la proposition 51. □

Proposition 53. ***(dite de non exécution spontanée ou d'unité de calcul implicite)*** *Pour tout $P \in \mathcal{P}$, $P \sim [P]$*

Démonstration. Directe depuis la proposition 51. □

Proposition 54. *Pour tout $P \in \mathcal{P}$ et $Q \in \mathcal{P}$, $P \sim Q \Leftrightarrow [P] \sim [Q]$*

Démonstration. $P \sim [P]$ et $Q \sim [Q]$ (proposition 53) donc, par transitivité de $\sim$, on a $P \sim Q \Leftrightarrow [P] \sim [Q]$ □

3.4.6 Relations entre unités de calcul

3.4.6.1 Équivalences entre les relations de bisimilarité dans $\mathcal{C} \times \mathcal{C}$

Les unités de calcul seules ne peuvent effectuer d'actions appartenant aux ensembles $\mathring{\Lambda}$ et $\vec{\mathcal{A}}$. Par conséquent, les bisimulations qui prennent en compte ces actions sont équivalentes à celles qui ne les prennent pas en compte au niveau unité de calcul. Nous leur appliquons donc la proposition 27 page 70 et le corollaire 28 page 70. Les équivalences entre les bisimulations au niveau unité de calcul sont données en figure 3.13. Bien que la relation d'implication représentée soit transitive, la clôture transitive n'est pas indiquée dans la figure 3.13.

3.4.6.2 Implications entre les relations de bisimilarité dans $\mathcal{C} \times \mathcal{C}$.

La figure 3.14 est l'union des figures 3.10 et 3.13. Elle illustre les implications entre les relations de bisimilarité des corollaires 26 et 28 (par exemple $C \vartriangle C' \Rightarrow C \vec{\vartriangle} C'$). Elle représente l'ensemble des équivalences et implications sur les bisimulations au niveau unité de calcul. Comme précédemment, la clôture transitive n'est pas indiquée dans la figure 3.14.

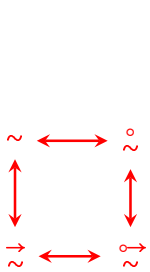
3.4.7 Exemples d'utilisation du niveau unité de calcul

Nous donnons maintenant quelques exemples qui utilisent les processus et les unités de calcul que nous avons définis.

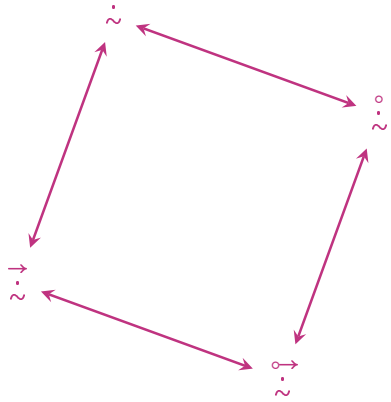
3.4.7.1 Election

L'objectif d'un algorithme d'élection est de distinguer exactement une des entités du réseau alors que toutes les entités exécutent ce même algorithme. Nous considérons ici des entités anonymes et nous montrons qu'il existe une expression d'unité de calcul C qui autorise une élection dans le réseau :

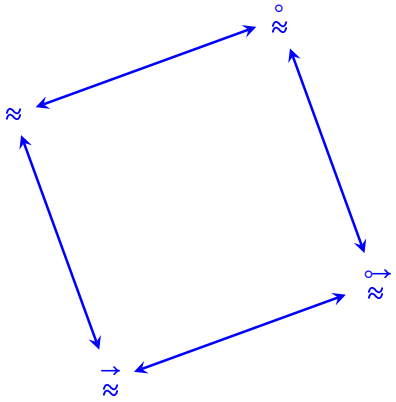
$$C = [\ddot{a}.\breve{P} + \dot{a}.0]$$



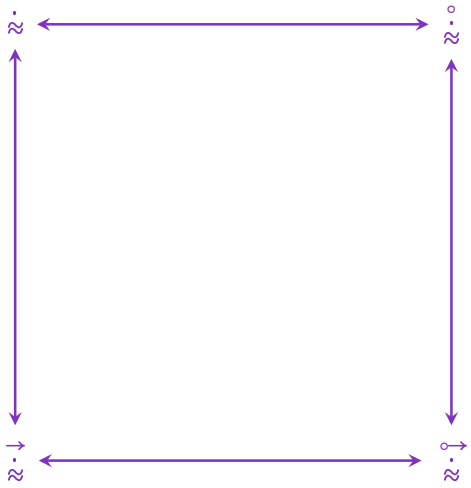
(a) τ et $\dot{\tau}$ observables



(b) τ observable et $\dot{\tau}$ pas observable



(c) τ pas observable et $\dot{\tau}$ observable



(d) τ et $\dot{\tau}$ pas observables

Fig. 3.13: Implication des relations de bisimilarité dans $\mathcal{C} \times \mathcal{C}$. $\rightarrow$ signifie implique.

Trivialement, pour tout $n \in \mathbb{N}^*$, $\underbrace{C \parallel C \parallel ... \parallel C}_{n \text{ fois}} \overset{\ddot{a}}{\rightarrow} [\check{P}] \parallel \underbrace{[0] \parallel ... \parallel [0]}_{n-1 \text{ fois}}$

Le principe repose sur l'approche triviale suivante : la première unité de calcul qui émet un message est élue. Il n'est pas possible que les processus embarqués dans les autres unités de calcul (et qui ne sont pas élus) ne reçoivent pas le message. Il n'existe pas d'algorithme équivalent au niveau des processus seuls.

En 2007, Maria Grazia Vigliotti et al. ont proposé une solution au problème de l'élection [VPP06] dans un réseau de processus modélisé en CCS. Le principe est le suivant : les processus cherchent à s'éliminer entre eux. Un processus éliminé avertit l'ensemble des autres processus de son élimination. Les processus comptent ainsi le nombre de processus

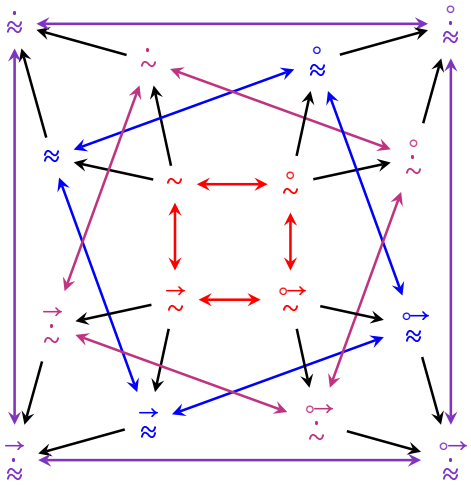


FIG. 3.14: Implication des relations de bisimilarité dans $\mathcal{C} \times \mathcal{C}$. $\rightarrow$ signifie implique.

éliminés. Lorsqu'un processus compte $k-1$ éliminés (s'il y a k processus au départ), il peut en conclure qu'il est le dernier encore en lice et donc qu'il est élu. Leur solution repose sur l'indéterminisme introduit par l'opérateur + qui permet de "casser" la symétrie qui existe dans le réseau, ce qui explique ce résultat qui pourrait paraître étonnant au vu des résultats exposés par Angluin dans [Ang80]. Sans la connaissance du nombre de processus par chacun d'eux, il n'existe pas d'algorithme d'élection en CCS, et donc au niveau processus seul de CiMAN. C'est l'objet de la proposition suivante.

Proposition 55. *Il n'existe pas de processus agents $P \in \mathcal{P}$ tel que pour tout $n \in \mathbb{N}^*$ $\left|_{i=1}^{n} P \xrightarrow{s} \right|_{i=1}^{n-1} PE \mid E$ avec $E \not\equiv PE$.*

Démonstration. Nous démontrons cette proposition par l'absurde. Soit $P \in \mathcal{P}$ tel que la proposition soit vraie. On a alors $\left|_{i=1}^{1} P = P \xrightarrow{s} E\right.$. On a donc $P \mid P \xrightarrow{s} E \mid P \xrightarrow{s} E \mid E$. P n'existe donc pas. □

Cet exemple permet de mettre en avant l'expressivité de la communication par *broadcast* entre les unités de calcul. Il y a une barrière de synchronisation globale implicite lors de l'émission d'un message au niveau unités de calcul. Cet exemple permet de se familiariser avec les opérations du niveau unité de calcul et de mettre en lumière une des différences importante entre le niveau processus et le niveau unité de calcul.

3.4.7.2 Mémoire tampon ordonnée

On définit l'expression agentique de processus $\breve{CR}_{\alpha,\beta}$ ($\alpha \in \dot{\bigwedge}$ et $\beta \in \bigwedge$) comme suit :

$$\breve{CR}_{\alpha,\beta}(x.l) \stackrel{\text{def}}{=} \begin{cases} \alpha(y).\breve{CR}_{\alpha,\beta}(y) & \text{si x.l} = 0 \\ \alpha(y).\breve{CR}_{\alpha,\beta}(x.l.y) + \beta(x).\breve{CR}_{\alpha,\beta}(l) & \text{sinon} \end{cases}$$

$\breve{CR}_{\alpha,\beta}$ définit une mémoire tampon ordonnée. Tous les messages reçus par cette mémoire sur le canal α sont stockés puis redistribués dans le même ordre sur le canal β.

Une mémoire tampon sans perte de message. Par mémoire tampon sans perte de message, on entend que les messages envoyés sur un canal physique $\alpha \in \dot{\Lambda}$ doivent être reçus. Autrement dit, si l'expression agentique du processus de mémoire tampon est nommée $\breve{P}$, alors, quelle que soit l'expression agentique d'une unité de calcul $\breve{C}'$ en parallèle de $\breve{P}$ exécuté seul ou en parallèle d'autres processus dans $\breve{C}$ ($\breve{C} \stackrel{\text{def}}{=} [\breve{P}]$ ou $\breve{C} \stackrel{\text{def}}{=} [\breve{P} \mid \breve{P}']$ pour tout $\breve{P}' \in \breve{\mathcal{P}}$), la règle C.Com$_1$ (resp. C.Com$_2$) n'est pas applicable sur $\breve{C}' \parallel \breve{C}$ (resp. $\breve{C} \parallel \breve{C}'$) ou leurs dérivés. Un message émis depuis C' sur α ne peut donc pas être perdu. Cette propriété peut être décrite par la propriété *AlwaysReadyToReceive* (qui est suffisante mais pas nécessaire) et que nous exprimons de la manière suivante :

Propriété 56. *Soient $\breve{P} \in \breve{\mathcal{P}}$ et $\dot{a} \in \dot{\mathcal{A}}$.*

$$AlwaysReadyToReceive(\breve{P}, \dot{a}) = \breve{P} \stackrel{\dot{a}}{\rightarrow} \breve{P}' \wedge AlwaysReadyToReceive(\breve{P}', \dot{a}).$$

Proposition 57. *$AlwaysReadyToReceive(\breve{CR}_{\alpha,\beta}, \alpha)$ est vraie.*

Démonstration. Pour tout x et pour tout y, $\breve{CR}_{\alpha,\beta}(x) \stackrel{\alpha(y)}{\rightarrow} \breve{CR}_{\alpha,\beta}(x.y)$ □

Proposition 58. *Pour tout $\breve{P} \in \breve{\mathcal{P}}$, $AlwaysReadyToReceive(\breve{P} \mid \breve{CR}_{\alpha,\beta}, \alpha)$ est vraie.*

Démonstration. Directe par l'utilisation de la proposition 57 et des règles C.Com$_3$ et C.Com$_4$. □

Une mémoire tampon ordonnée (FIFO).

Définition 59. *Soient $\alpha \in Act$ tel que $\alpha \notin \{in, \overline{out}\}$, v et v' des valeurs et V une séquence de valeurs. Soit S la grammaire telle que*

$$S(v.V) \rightarrow \alpha.S(v.V) + in(v').S(v.V.v') + \overline{out}(v).S(V)$$

*$\breve{P}$ est **FIFO** sur $(in, \overline{out})$ si pour toute trace $t \in T(P)$, t peut être construite par la grammaire S.*

Proposition 60. *$\breve{CR}_{\alpha,\beta}$ est FIFO sur (α, β).*

Démonstration. Triviale. □

Proposition 61. *Soit $\breve{P} \in \breve{\mathcal{P}}$. Si $\alpha \notin \mathcal{L}(\breve{P})$ alors $\breve{CR}_{\alpha,\beta} \mid \breve{P}$ est FIFO sur (α, β).*

Démonstration. Omise. □

3.4.7.3 Utilisation de la mémoire tampon ordonnée

Réécriture d'une expression agentique de processus pour l'utilisation d'une mémoire tampon ordonnée. Nous définissons une opération de réécriture Φ d'une expression agentique de processus qui a pour effet de garantir qu'il n'existe pas de message perdu par les processus embarqués dans l'unité de calcul réécrite (i.e. envoyé par une unité de calcul tierce mais non reçu) sur le canal $\dot{a} \in \dot{\mathcal{A}}$.

Soit $\breve{P}$ une expression agentique telle que $\dot{a} \in \mathcal{L}(\breve{P})$ et $a \notin \mathcal{L}(\breve{P})$. Afin de permettre à $\breve{P}$ de ne perdre aucun message arrivant sur le canal $\dot{a}$, nous le réécrivons en $\Phi_{\dot{a},a}(\breve{P}) = (\phi_{\dot{a},a}(\breve{P}) \mid \breve{CR}_{\dot{a},a}) \setminus a$ avec $\phi_{\dot{a},a}$ définie sur la structure de $\breve{P}$ comme suit :

– Inaction

  1. $\phi_{\dot{a},a}(0) = 0$

– Action

  1. $\phi_{\dot{a},a}(\dot{a}.\breve{P}) = a.\phi_{\dot{a},a}(\breve{P})$
  2. $\phi_{\dot{a},a}(\alpha.\breve{P}) = \alpha.\phi_{\dot{a},a}(\breve{P})$ si $\alpha \neq \dot{a}$

– Somme $\phi_{\dot{a},a}(\breve{P}_1 + \breve{P}_2) = \phi_{\dot{a},a}(\breve{P}_1) + \phi_{\dot{a},a}(\breve{P}_2)$
– Composition $\phi_{\dot{a},a}(\breve{P}_1 \mid \breve{P}_2) = \phi_{\dot{a},a}(\breve{P}_1) \mid \phi_{\dot{a},a}(\breve{P}_2)$
– Restriction $\phi_{\dot{a},a}(\breve{P} \setminus b) = \phi_{\dot{a},a}(\breve{P}) \setminus b$ (pour mémoire $b \neq a$ car $a \notin \mathcal{L}(\breve{P})$).
– Renommage $\phi_{\dot{a},a}(\breve{P}[c/b]) = \phi_{\dot{a},a}(\breve{P})[c/b]$ (pour mémoire $b \neq a$ car $a \notin \mathcal{L}(\breve{P})$).

Au lieu de recevoir sur $\dot{a}$ (canal physique ou fil), le processus reçoit donc sur a (canal logique) et $\breve{CR}_{\dot{a},a}$ fait le lien entre le canal physique et le canal logique.

Exemple simple. $\Phi_{\dot{a},a}(\dot{a}(x).\ddot{b}(x).0) = (a(x).\ddot{b}(x).0 \mid \breve{CR}_{\dot{a},a}) \setminus a$.

Exemple de comptage de ping/pong. Par cet exemple, nous montrons de quelle manière le processus $\breve{CR}$ empêche la perte de message entre unités de calcul. Le processus $\breve{P}$ à pour objectif de compter les unités de calcul qui partagent un canal (ou bus) avec l'unité de calcul sur laquelle il est embarqué. Les processus $\breve{Q}$ existent pour répondre à la requête de "ping" du processus $\breve{P}$ et ainsi être comptés.

Soient les processus suivants :

$\breve{P} \stackrel{\text{def}}{=} \ddot{ping}.0 \mid \breve{CPT}(0)$

$\breve{Q} \stackrel{\text{def}}{=} \dot{ping}.\ddot{pong}.0$

$\breve{CPT}(i) \stackrel{\text{def}}{=} \ddot{pong}.\overline{print}(i+1).CPT(i+1)$

Soit $C = ([P] \parallel [Q] \parallel [Q]) \setminus_{\ddot{ping},\ddot{pong}}$

Nous présentons ci-dessous les deux exécutions possibles de C.

$$
\begin{array}{lcl}
C & \equiv & [\ddot{ping}.0 \mid \dot{pong}.\overline{print}(1).CPT(1)] \parallel [\dot{ping}.\ddot{pong}.0] \parallel [\dot{ping}.\ddot{pong}.0] \\
 & \xrightarrow{\ddot{ping}} & [0 \mid \dot{pong}.\overline{print}(1).CPT(1)] \parallel [\ddot{pong}.0] \parallel [\ddot{pong}.0] \\
 & \xrightarrow{\ddot{pong}} & [0 \mid \overline{print}(1).CPT(1)] \parallel [0] \parallel [\ddot{pong}.0] \\
 & \xrightarrow{\overline{print}(1)} & [0 \mid \dot{pong}.\overline{print}(2).CPT(2)] \parallel [0] \parallel [\ddot{pong}.0] \\
 & \xrightarrow{\ddot{pong}} & [0 \mid \overline{print}(2).CPT(2)] \parallel [0] \parallel [0] \\
 & \xrightarrow{\overline{print}(2)} & [0 \mid CPT(2)] \parallel [0] \parallel [0] \\
 & \sim & 0
\end{array}
$$

$$
\begin{array}{lcl}
C & \equiv & [\ddot{ping}.0 \mid \dot{pong}.\overline{print}(1).CPT(1)] \parallel [\dot{ping}.\ddot{pong}.0] \parallel [\dot{ping}.\ddot{pong}.0] \\
 & \xrightarrow{\ddot{ping}} & [0 \mid \dot{pong}.\overline{print}(1).CPT(1)] \parallel [\ddot{pong}.0] \parallel [\ddot{pong}.0] \\
 & \xrightarrow{\ddot{pong}} & [0 \mid \overline{print}(1).CPT(1)] \parallel [0] \parallel [\ddot{pong}.0] \\
 & \xrightarrow{\ddot{pong}} & [0 \mid \overline{print}(1).CPT(1)] \parallel [0] \parallel [0] \\
 & \xrightarrow{\overline{print}(1)} & [0 \mid CPT(1)] \parallel [0] \parallel [0] \\
 & \sim & 0
\end{array}
$$

On a donc $C \dot{\approx} \overline{print}(1).(\overline{print}(2).0 + \dot{\tau}.0)$. Le processus P compte donc 1 ou 2 unités de calcul répondant au ping. Il arrive donc qu'une unité de calcul ne soit pas comptée car son message de réponse est perdu, en l'occurence quand $\breve{P}$ n'est pas en attente sur $\dot{ping}$.

On réécrit maintenant $\breve{P}$ par $\Phi_{\dot{ping},a}$:

$$
\begin{array}{lcl}
\Phi_{\dot{ping},a}(\breve{P}) & = & (\phi_{\dot{ping},a}(\ddot{ping}.0 \mid C\breve{P}T(0)) \mid C\breve{R}_{\dot{a},a}) \setminus a \\
 & & (\phi_{\dot{ping},a}(\ddot{ping}.0) \mid \phi_{\dot{ping},a}(C\breve{P}T(0)) \mid C\breve{R}_{\dot{a},a}) \setminus a \\
 & & (\phi_{\dot{ping},a}(\ddot{ping}.0) \mid \phi_{\dot{ping},a}(\dot{pong}.\overline{print}(1).C\breve{P}T(1)) \mid C\breve{R}_{\dot{a},a}) \setminus a \\
 & & (\ddot{ping}.0 \mid pong.\overline{print}(1).C\breve{P}T(1) \mid C\breve{R}_{\dot{a},a}) \setminus a
\end{array}
$$

On a alors $([\Phi_{\dot{a},a}(\breve{P})] \parallel [\breve{Q}] \parallel [\breve{Q}] \parallel [\breve{Q}]) \setminus_{\ddot{a}} \dot{\approx} \overline{p}(1).\overline{p}(2).0$. Les messages de réponse *pong* ne se perdent plus. La propriété *AlwaysReadyToReceive* est en effet toujours vraie sur $\Phi_{\dot{ping},a}(\breve{P})$ (par la proposition 58).

3.5 Les nœuds et les messages

Dans la suite, x désigne une variable, e est une valeur.

3.5.1 Grammaire

Un nœud est un ensemble d'unités de calcul solidairement mobiles qui possède une position et un graphe de mobilité.

$$
\begin{array}{lcll}
\breve{N} & ::= & {}_i[\![\breve{C}]\!]^G_w & {}_{id}[\![\textit{unité de calcul}]\!]^{\textit{graphe de mobilité}}_{\textit{position}} \\
 & \mid & \breve{N}_1 \mid\mid\mid \breve{N}_2 & \textit{composition parallèle de nœuds}
\end{array}
$$

Les messages, tout comme les nœuds se voient associer un graphe qui permet de définir leur mobilité. Une différence notable est qu'un message, contrairement à un nœud, peut se

trouver à plusieurs positions simultanément. Une fois créé, la mobilité d'un message n'est pas la conséquence d'une action (les messages ne peuvent pas effectuer d'actions), mais simplement une évolution inéluctable dictée par le graphe de propagation.

$$\breve{M} ::= {}^{\alpha}_{i}(\!| e |\!)^{G}_{W} \qquad {}^{canal}_{id}(\!|\, donnée \,|\!)^{graphe\ de\ propagation}_{ensemble\ de\ positions}$$
$$\mid \breve{M}_1 \,|||\, \breve{M}_2 \qquad composition\ parallèle\ de\ messages$$

Nous définissons que le monde est une composition parallèle de nœuds et de messages.

$$\begin{array}{lll} \breve{O} ::= & \breve{N} & le\ monde\ est\ composé\ de\ nœuds \\ \mid & \breve{M} & et\ de\ messages \\ \mid & \breve{O} \,|||\, \breve{N} & composition\ parallèle\ du\ monde\ et\ d'un\ nœud \\ \mid & \breve{O} \,|||\, \breve{M} & composition\ parallèle\ du\ monde\ et\ d'un\ message \end{array}$$

3.5.2 Notations, définitions utiles

Les nœuds et les messages que nous définissons dans CiMAN sont des entités mobiles qui, par conséquent, possèdent une position ou un ensemble de positions. Nous supposons que l'espace dans lequel les nœuds et les messages évoluent peut être discrétisé en un ensemble de positions.

Définition 62. *$\mathcal{W}$ est l'ensemble des positions possibles des nœuds et des messages dans CiMAN. Cet ensemble de positions forme l'espace dans lequel les nœuds et les messages évoluent. Nous utilisons w, w_i pour dénoter les éléments de $\mathcal{W}$.*

Définition 63. ***Un graphe de mobilité*** *G est un graphe orienté sans cycle de taille de 1 dont les sommets correspondent à des positions et les arcs aux déplacements possibles entre ces positions. G est donc de la forme $G = (V, E)$ avec $V \subseteq \mathcal{W}, E \subseteq V \times V$. $\mathcal{G}_N$ est l'ensemble de tous les graphes de mobilité possibles. Nous utilisons G, G_i pour dénoter les éléments de $\mathcal{G}_N$.*

Afin de définir les graphes de propagation de message, nous introduisons un sommet particulier que nous nommons PUITS qui ne possède par définition aucun arc sortant. Une fois ce puits atteint, un message ne peut donc plus se déplacer ; ce sommet ne pouvant pas faire partie des graphes de mobilité des nœuds, il est donc impossible pour un nœud de s'y déplacer et de recevoir ce message. Ce sommet nous permet d'exprimer la disparition et la perte de message.

Définition 64. ***Un graphe de propagation*** *G est un graphe orienté sans cycle de taille 1 dont les sommets correspondent à des positions et les arcs aux déplacements possibles entre ces positions. G est de la forme $G = (V, E)$ avec $V \subseteq (\mathcal{W} \cup \{\text{PUITS}\}), E \subseteq V \times V$ et tel que pour tout $w \in \mathcal{W}$, il n'existe pas $e \in E$ tel que $e = (\text{PUITS}, w)$. $\mathcal{G}_M$ est l'ensemble de tous les graphes de propagation possibles. Nous utilisons G, G_i pour dénoter les éléments de $\mathcal{G}_M$.*

Dans les travaux qui suivent, nous supposons toujours que pour chaque sommet des graphes de propagation, il existe un arc sortant vers le PUITS. Nous supposons donc que chaque message peut disparaître à tout instant.

On suppose que chaque nœud et chaque message possèdent une identité unique. Cette identité n'est pas connue du nœud ni du message, elle sert uniquement à simplifier certaines

écritures dans CiMAN. Nous définissons les *fonctions accesseurs* G et W qui permettent d'accéder au graphe de mobilité et de propagation et à la position d'un nœud ou d'un message grâce à son identité. Le fait que l'identité soit unique fait de ces fonctions des applications.

Définition 65. *Soit le nœud* ${}_i[\![C]\!]^G_w$ *(resp. le message* ${}^c_i(\!|\, e \,|\!)^G_W$*). Les fonctions* G *et* W *que nous définissons ici nous permettent d'accéder au graphe de mobilité (resp. au graphe de propagation) et à la position du nœud (resp. du message) qui possède l'identité* $i \in I$*.*

$$\begin{array}{rcccl} \mathsf{G} & : & I & \to & \mathcal{G} \\ & & i & \mapsto & G \end{array} \qquad\qquad \begin{array}{rcccl} \mathsf{W} & : & I & \to & \mathcal{W} \\ & & i & \mapsto & w \text{ (resp. } W \subseteq \mathcal{W} \cup \{\text{PUITS}\}\text{)} \end{array}$$

Notation Lorsque $i \in I$, $G \in \mathcal{G}$ et $w \in \mathcal{W}$ sont indéterminés (ou non nécessaires à la modélisation), on écrira $[\![C]\!]$ au lieu de ${}_i[\![C]\!]^G_w$.

3.5.3 Transitions

Rappel : $\alpha \in Act$

Préfixe 1 : N.Act$_1$ $\quad \dfrac{\breve{C} \overset{\alpha}{\to} \breve{C}'}{{}_i[\![\breve{C}]\!]^G_w \overset{\alpha}{\to} {}_i[\![\breve{C}']\!]^G_w} \; (\alpha \notin (\overset{\circ}{\bigwedge} \cup \vec{\mathcal{A}}))$

Si une unité de calcul embarquée dans un nœud évolue par une transition autre qu'une action de communication inter-nœuds ou une action de déplacement, alors le nœud qui embarque cette unité de calcul évolue lui aussi par cette transition.

Émission : N.Act$_2$ $\quad \dfrac{\breve{C} \overset{\alpha(e)}{\to} \breve{C}'}{{}_i[\![\breve{C}]\!]^G_w \overset{\alpha_w(e)}{\to} {}_i[\![\breve{C}']\!]^G_w \,|||\, {}^{\alpha}_{j}(\!|\, e \,|\!)^{G(i,w,\alpha)}_{\{w\}}} \; (\alpha \in \overset{\circ}{\overset{\circ}{\mathcal{A}}})$

Si une unité de calcul effectue une action de l'ensemble $\overset{\circ}{\overset{\circ}{\mathcal{A}}}$ (ensemble d'actions qui correspondent à des envois de messages) alors un message est créé. Ce message possède son propre graphe de propagation et ses propres positions. Un message radio se propageant dans l'espace peut se situer à plusieurs positions simultanément et se voit donc associer un ensemble de positions et non pas une position unique comme c'est le cas pour les nœuds. On notera que le graphe de propagation ainsi créé dépend de l'identité du nœud émetteur, de la position de ce dernier au moment de l'émission et du canal utilisé pour celle-ci. Ce graphe de propagation est variable car, par exemple, la portée du dispositif mobile dont il est issu peut dépendre d'un ensemble de paramètres. De plus, l'étiquette associée à la transition contient la position du nœud émetteur afin qu'une relation de congruence puisse être définie sur $\mathcal{N} \times \mathcal{N}$.

Réception : N.Act$_3$ $\quad \dfrac{\breve{C} \overset{\alpha(x)}{\to} \breve{C}'}{{}_i[\![\breve{C}]\!]^G_w \,|||\, {}^{\alpha}_{j}(\!|\, e \,|\!)^{G'}_W \overset{\alpha_w(e)}{\to} {}_i[\![\breve{C}'[e/x]]\!]^G_w \,|||\, {}^{\alpha}_{j}(\!|\, e \,|\!)^{G'}_W} \; (\alpha \in \overset{\circ}{\mathcal{A}} \wedge w \in W)$

Un nœud évolue par la transition α_w ($\alpha \in \mathring{\vec{\mathcal{A}}}$, $w \in \mathcal{W}$) ; i.e. reçoit un message si (i) il embarque une unité de calcul qui peut évoluer par la transition α, (ii) il se situe à la position w, et (iii) il est en parallèle d'un message qui se situe à la position w. De même que précédemment, l'étiquette associée à la transition contient la position du nœud récepteur afin qu'une relation de congruence puisse être définie sur $\mathcal{N} \times \mathcal{N}$.

Mobilité des nœuds : N.Mob $$\frac{\breve{C} \overset{\vec{w'}}{\to} \breve{C}'}{{}_i[\![\breve{C}]\!]^G_w \overset{\overrightarrow{ww'}}{\to} {}_i[\![\breve{C}']\!]^G_{w'}} ((w,w') \in E(G))$$

Un nœud évolue par la transition $\overrightarrow{ww'}$ (donc se déplace) si (i) l'unité de calcul qu'il embarque possède une transition $\vec{w}' \in \vec{\mathcal{A}}$, (ii) il se situe à la position w, et (iii) son graphe de mobilité G lui permet de se déplacer de w vers w'.

Mobilité des messages : M.Mob $$\frac{}{{}^a_i(\!|\, e \,|\!)^G_W \overset{\overrightarrow{WW'}}{\to} {}^a_i(\!|\, e \,|\!)^G_{W'}} (W' \subseteq \{w' \in \mathcal{W}, w \in W \wedge (w,w') \in E(G)\} \wedge W' \neq \varnothing)$$

Un message n'est pas un processus et par conséquent sa mobilité n'est pas la conséquence d'une quelconque action (un message ne peut pas effectuer d'action). La mobilité d'un message dépend donc uniquement de son graphe de propagation G. Un message, à partir d'un ensemble de positions, peut se déplacer vers un sous-ensemble non nul des successeurs dans le graphe de mobilité (sous-ensemble contenant éventuellement le PUITS) des sommets correspondant à ces positions.

Composition 1 : N.Com$_1$ $$\frac{\breve{N}_1 \overset{\alpha}{\to} \breve{N}_1'}{\breve{N}_1 \,|||\, \breve{N}_2 \overset{\alpha}{\to} \breve{N}_1' \,|||\, \breve{N}_2}$$

Si un nœud peut évoluer par une transition α alors il peut évoluer indépendamment des autres nœuds en parallèle.

Composition 2 : N.Com$_2$ $$\frac{\breve{N}_2 \overset{\alpha}{\to} \breve{N}_2'}{\breve{N}_1 \,|||\, \breve{N}_2 \overset{\alpha}{\to} \breve{N}_1 \,|||\, \breve{N}_2'}$$

Cette règle est la symétrique de la règle précédente.

Évolution du monde : O.Com$_1$ $$\frac{\breve{O} \overset{\alpha}{\to} \breve{O}'}{\breve{O} \,|||\, \breve{N} \overset{\alpha}{\to} \breve{O}' \,|||\, \breve{N}}$$

Les nœuds peuvent évoluer indépendamment des autres éléments du monde dans lequel ils sont. On notera que les seules règles qui autorisent une interaction entre un monde et les nœuds sont les règles N.Act_2 (émission d'un message par un nœud) et N.Act_3 (réception d'un message par un nœud).

Évolution du monde : O.Com$_2$ $\quad \dfrac{\breve{O} \xrightarrow{\alpha} \breve{O}'}{\breve{N} \,|||\, \breve{O} \xrightarrow{\alpha} \breve{N} \,|||\, \breve{O}'}$

Cette règle est la symétrique de la règle précédente.

Évolution du monde : O.Com$_3$ $\quad \dfrac{\breve{O} \xrightarrow{\alpha} \breve{O}'}{\breve{O} \,|||\, \breve{M} \xrightarrow{\alpha} \breve{O}' \,|||\, \breve{M}}$

Les messages peuvent évoluer indépendamment des éléments du monde dans lequel ils sont. Aucun élément du monde (nœud ou message) n'a d'influence sur l'évolution (c'est à dire la mobilité) d'un message une fois qu'il est créé.

Évolution du monde : O.Com$_4$ $\quad \dfrac{\breve{O} \xrightarrow{\alpha} \breve{O}'}{\breve{M} \,|||\, \breve{O} \xrightarrow{\alpha} \breve{M} \,|||\, \breve{O}'}$

Cette règle est la symétrique de la règle précédente.

3.5.4 Sorte

Proposition 66. *Pour tout $\breve{N}$ et $L \subseteq \mathcal{L}$, L est une sorte de $\breve{N}$ si et seulement si pour tout α tel que $\breve{N} \xrightarrow{\alpha} \breve{N}'$ alors*

1. $\alpha \in L \cup \{\tau, \dot{\tau}\}$
2. *L est une sorte de $\breve{N}'$*

Démonstration. Triviale. □

Définition 67. *La sorte syntaxique de l'expression agentique $\breve{N}$ notée $\mathcal{L}(\breve{N})$ (que nous appelons parfois simplement sorte de $\breve{N}$) est définie comme suit :*

$$
\begin{array}{lcll}
\mathcal{L}(\llbracket \breve{C} \rrbracket) & = & (\mathcal{L}(\breve{C}) \cap (\bigwedge \cup \dot{\bigwedge})) & \textit{(on interdit ici les } \mathring{a}, \ \mathring{a}^{\circ} \textit{ et } \vec{w}\textit{)} \\
& \cup & \{\alpha_w, \alpha \in \mathcal{L}(\breve{C}) \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} & \textit{(on ajoute ici les } \mathring{a}^{\circ}_w, \ \mathring{a}_w\mathring{}\textit{)} \\
& \cup & \{\vec{ww'}, w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}) \cap \vec{\mathcal{A}}\} & \textit{(on ajoute ici les } \vec{ww'}\textit{)} \\
& & & \textit{(i.e. on localise les actions d'émission OTA,} \\
& & & \textit{de réception associées et de déplacement)} \\
\mathcal{L}(\breve{N}_1 \,|||\, \breve{N}_2) & = & \mathcal{L}(\breve{N}_1) \cup \mathcal{L}(\breve{N}_2) &
\end{array}
$$

Nous devons maintenant montrer que la fonction $\mathcal{L}(\breve{N})$ de la définition 67 que nous avons définie est bien une sorte de $\breve{N}$. Pour prouver cela nous montrons tout d'abord la proposition suivante.

Proposition 68. *Soit* $\breve{N} \xrightarrow{\alpha} \breve{N}'$. *Alors*

1. $\alpha \in \mathcal{L}(\breve{N}) \cup \{\tau, \dot{\tau}\}$
2. $\mathcal{L}(\breve{N}') \subseteq \mathcal{L}(\breve{N})$

Démonstration. Par induction sur les transitions de la forme $\breve{N} \xrightarrow{\alpha} \breve{N}'$. La proposition est vraie pour $\breve{N} = [\![[0]]\!]$. L'hypothèse d'induction est que la proposition est vraie pour $\breve{N}'$ qui possède n constructeurs. Soit $\breve{N}$ un processus qui possède $n+1$ constructeurs.

Cas 1. $\breve{N} \equiv [\![\breve{C}]\!]$ et $\breve{C} \xrightarrow{\alpha} \breve{C}'$ et $(\alpha \notin (\mathring{\bigwedge} \cup \vec{\mathcal{A}}))$ (N.Act$_1$)

Par la définition 67 on a

$$\left.\begin{array}{rcl} \mathcal{L}([\![\breve{C}]\!]) & = & (\mathcal{L}(\breve{C}) \cap (\bigwedge \cup \dot{\bigwedge})) \\ & \cup & \{\alpha_{\vec{w}}, \alpha \in \mathcal{L}(\breve{C}) \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} \\ & \cup & \{w\vec{w}', w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}) \cap \vec{\mathcal{A}}\} \end{array}\right\} \text{(i)}$$

et par induction

1. $\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}') \subseteq \mathcal{L}(\breve{C})$ (iii)

donc

1. $\alpha \in \mathcal{L}([\![\breve{C}]\!]) \cup \{\tau, \dot{\tau}\}$ par (i)
2. $$\begin{array}{rcll} \mathcal{L}([\![\breve{C}']\!]) & = & (\mathcal{L}(\breve{C}') \cap (\bigwedge \cup \dot{\bigwedge})) & \\ & \cup & \{\alpha_{\vec{w}}, \alpha \in \mathcal{L}(\breve{C}') \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} & \\ & \cup & \{w\vec{w}', w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}') \cap \vec{\mathcal{A}}\} & \text{par la définition 67} \\ & \subseteq & (\mathcal{L}(\breve{C}) \cap (\bigwedge \cup \dot{\bigwedge})) & \\ & \cup & \{\alpha_{\vec{w}}, \alpha \in \mathcal{L}(\breve{C}) \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} & \\ & \cup & \{w\vec{w}', w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}) \cap \vec{\mathcal{A}}\} & \text{par (iii)} \\ & \subseteq & \mathcal{L}([\![\breve{C}]\!]) & \text{par (i)} \end{array}$$

Cas 2. $\breve{C} \equiv [\![\breve{C}]\!]$ et $\breve{C} \xrightarrow{\alpha} \breve{C}'$ et $(\alpha \in \mathring{\vec{\mathcal{A}}})$ (N.Act$_2$)

Par la définition 67 on a

$$\left.\begin{array}{rcl} \mathcal{L}([\![\breve{C}]\!]) & = & (\mathcal{L}(\breve{C}) \cap (\bigwedge \cup \dot{\bigwedge})) \\ & \cup & \{\alpha_{\vec{w}}, \alpha \in \mathcal{L}(\breve{C}) \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} \\ & \cup & \{w\vec{w}', w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}) \cap \vec{\mathcal{A}}\} \end{array}\right\} \text{(i)}$$

et par induction

1. $\alpha \in \mathcal{L}(\breve{C}) \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}') \subseteq \mathcal{L}(\breve{C})$ (iii)

donc

1. $\forall w \in \mathcal{W}, \alpha_w \in \mathcal{L}(\llbracket \breve{C} \rrbracket) \cup \{\tau, \dot{\tau}\}$ par (i)
2. $\begin{array}{rcll} \mathcal{L}(\llbracket \breve{C}' \rrbracket) & = & (\mathcal{L}(\breve{C}') \cap (\bigwedge \cup \dot{\bigwedge})) & \\ & \cup & \{\alpha_w, \alpha \in \mathcal{L}(\breve{C}') \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} & \\ & \cup & \{w \vec{w}', w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}') \cap \vec{\mathcal{A}}\} & \text{par la définition 67} \\ & \subseteq & (\mathcal{L}(\breve{C}) \cap (\bigwedge \cup \dot{\bigwedge})) & \\ & \cup & \{\alpha_w, \alpha \in \mathcal{L}(\breve{C}) \cap \mathring{\bigwedge} \wedge w \in \mathcal{W}\} & \\ & \cup & \{w \vec{w}', w \in \mathcal{W} \wedge w' \in \mathcal{L}(\breve{C}) \cap \vec{\mathcal{A}}\} & \text{par (iii)} \\ & \subseteq & \mathcal{L}(\llbracket \breve{C} \rrbracket) & \text{par (i)} \end{array}$

Cas 3. $\breve{C} \equiv \llbracket \breve{C} \rrbracket$ et $\breve{C} \xrightarrow{\alpha} \breve{C}'$ et $(\alpha \in \mathring{\mathcal{A}})$ (N.Act$_3$) : Similaire.

Cas 4. $\breve{C} \equiv \llbracket \breve{C} \rrbracket$ et $\breve{C} \xrightarrow{\alpha} \breve{C}'$ et $(\alpha \in \vec{\mathcal{A}})$ (N.Mob) : Similaire.

Cas 5. $\breve{N} \equiv \llbracket \breve{C}_1 \rrbracket \,|||\, \llbracket \breve{C}_2 \rrbracket$ et $\llbracket \breve{C}_1 \rrbracket \xrightarrow{\alpha} \llbracket \breve{C}_1' \rrbracket$ (N.Com$_1$)

Par la définition 67 on a

$\mathcal{L}(\llbracket \breve{C}_1 \rrbracket \,|||\, \llbracket \breve{C}_2 \rrbracket) = \mathcal{L}(\llbracket \breve{C}_1 \rrbracket) \cup \mathcal{L}(\llbracket \breve{C}_2 \rrbracket)$ (i)

et par induction

1. $\alpha \in \mathcal{L}(\breve{C}_1) \cup \{\tau, \dot{\tau}\}$ (ii)
2. $\mathcal{L}(\breve{C}_1') \subseteq \mathcal{L}(\breve{C}_1)$ (iii)

donc

1. $\alpha \in \mathcal{L}(\llbracket \breve{C}_1 \rrbracket \,|||\, \llbracket \breve{C}_2 \rrbracket) \cup \{\tau, \dot{\tau}\}$ par (i)
2. $\begin{array}{rcll} \mathcal{L}(\llbracket \breve{C}_1' \rrbracket \cup \llbracket \breve{C}_2 \rrbracket) & = & \mathcal{L}(\llbracket \breve{C}_1' \rrbracket) \cup \mathcal{L}(\llbracket \breve{C}_2 \rrbracket) & \text{par la définition 67} \\ & \subseteq & \mathcal{L}(\llbracket \breve{C}_1 \rrbracket) \cup \mathcal{L}(\llbracket \breve{C}_2 \rrbracket) & \text{par (iii)} \\ & \subseteq & \mathcal{L}(\llbracket \breve{C}_1 \rrbracket \,|||\, \llbracket \breve{C}_2 \rrbracket) & \text{par (i)} \end{array}$

Cas 6. $\breve{N} \equiv \llbracket \breve{C}_1 \rrbracket \,|||\, \llbracket \breve{C}_2 \rrbracket$ et $\llbracket \breve{C}_2 \rrbracket \xrightarrow{\alpha} \llbracket \breve{C}_2' \rrbracket$ (N.Com$_2$) : Similaire.

□

Corollaire 69. $\breve{N} : \mathcal{L}(\breve{N})$

Démonstration. Nous allons montrer que $\mathcal{L}(\breve{N})$ satisfait la définition 14. Soit α une action d'une dérivée $\breve{N}'$ de $\breve{N}$. Par itération de la proposition 68(2), on a $\mathcal{L}(\breve{N}') \subseteq \mathcal{L}(\breve{N})$. De plus, grâce à la proposition 68(1), on a $\alpha \in \mathcal{L}(\breve{N}') \cup \{\tau, \dot{\tau}\}$, et ainsi $\alpha \in \mathcal{L}(\breve{N}) \cup \{\tau, \dot{\tau}\}$. □

Corollaire 70. *Si* $C \in \mathcal{C}$ *alors* $\mathcal{L}(\llbracket C \rrbracket) \subseteq \bigwedge \cup \dot{\bigwedge}$.

Démonstration. Par la définition 67, pour tout $C \in \mathcal{C}$, $\mathcal{L}(\llbracket C \rrbracket) = \mathcal{L}(C)$. Or, par la définition 10, $\mathcal{L}(C) \subseteq Act(\mathcal{C}) = \bigwedge \cup \dot{\bigwedge} \cup \{\tau, \dot{\tau}\}$ et $\tau, \dot{\tau} \notin \mathcal{L}(C)$ donc $\mathcal{L}(C) \subseteq \bigwedge \cup \dot{\bigwedge}$. □

3.5.5 Congruence observationnelle

Définition 71. *Un contexte de nœud* C_N *est, informellement parlant, une expression de nœud avec un trou représenté par []. Plus formellement, un contexte de nœud est grammaticalement défini par :*

$$C_N \quad ::= \quad [] \quad \Big| \quad C_N \,|||\, N \quad \Big| \quad N \,|||\, C_N$$

Définition 72. *Soient N_1 et N_2 deux éléments de $\mathcal{N}$ et $\mathcal{R}$ une relation d'équivalence (c'est à dire réflexive, symétrique et transitive) sur $\mathcal{N} \times \mathcal{N}$. On dit que $\mathcal{R}$ est une congruence sur $\mathcal{N} \times \mathcal{N}$ si pour tout élément $N_1, N_2 \in \mathcal{N}$, et pour tout contexte de nœud C_N, $N_1 \mathcal{R} N_2 \Rightarrow C_N[N_1] \mathcal{R} C_N[N_2]$. On dit alors que N_1 et N_2 sont égaux modulo $\mathcal{R}$ ou congruents modulo $\mathcal{R}$.*

Définition 73. *Pour tout N_1, $N_2 \in \mathcal{N}$, on dit que $N_1 \dot{\approx}_N N_2$ si et seulement si $N_1 \dot{\approx} N_2$ et pour tout $N' \in \mathcal{N}$, $N_1 \,|||\, N' \dot{\approx} N_2 \,|||\, N'$.*

Proposition 74. *$\dot{\approx}_N$ est une congruence sur $\mathcal{N} \times \mathcal{N}$.*

Démonstration. Triviale. □

3.5.6 Relations entre unités de calcul et nœuds

Proposition 75. *S'il existe une transition pour une expression agentique d'unité de calcul alors cette transition existe aussi pour le nœud qui l'embarque, et réciproquement. Pour tout $C \in \mathcal{C}$ et $\alpha \in Act$, $C \xrightarrow{\alpha} C' \Leftrightarrow [\![C]\!] \xrightarrow{\alpha} [\![C']\!]$.*

Démonstration. En deux temps.

1. ($\Rightarrow$) $C \in \mathcal{C}$ donc $\alpha \in Act(\mathcal{C})$ donc si $C \xrightarrow{\alpha} C'$ alors $[\![C]\!] \xrightarrow{\alpha} [\![C']\!]$ (N.Act_1).
2. ($\Leftarrow$) Par l'absurde. Supposons que $\neg(\forall \alpha \in Act,\ [\![C]\!] \xrightarrow{\alpha} [\![C']\!] \Rightarrow C \xrightarrow{\alpha} C')$, c'est à dire $(\exists\ \alpha \in Act,\ [\![C]\!] \xrightarrow{\alpha} [\![C']\!] \wedge C \overset{\alpha}{\not\to} C')$. Notons aussi que $C \in \mathcal{C}$ donc d'après le corollaire 70, $\mathcal{L}([\![C]\!]) \subseteq \dot{\bigwedge} \cup \bigwedge$. Les transitions qui permettent aux unités de calcul d'évoluer sont les règles N.$*$, donc au moins une de ces règles permet cette transition. Nous les parcourons à la recherche de celle(s)-ci.
   (a) N.Act_1, N.Act_2, N.Act_3 et N.Mob ne sont évidemment pas ces règles car la condition d'inférence est $C \xrightarrow{\alpha} C'$.
   (b) N.Com_1 n'est pas non plus cette règle car $\alpha \in \dot{\bigwedge} \cup \bigwedge$ et donc la condition d'inférence dépend directement de l'application de la règle C.Act_1, or par hypothèse, $C \overset{\alpha}{\not\to} C'$.
   (c) N.Com_2. Similaire.

   En conclusion, les règles N.Com_1 et N.Com_2 dépendent de la condition d'application de N.Act_1, N.Act_2, N.Act_3 et N.Mob. Nous avons montré que celles-ci ne sont pas applicables. Il n'existe donc pas $\alpha \in Act$ tel que $[\![C]\!] \xrightarrow{\alpha} [\![C']\!]$ et $C \overset{\alpha}{\not\to} C'$.

Nous avons donc montré que $\forall \alpha \in Act,\ [\![C]\!] \xrightarrow{\alpha} [\![C']\!] \Leftrightarrow C \xrightarrow{\alpha} C'$. □

Proposition 76. *Pour tout $C \in \mathcal{C}$ et $s \in Act^*$, $C \xrightarrow{s} C' \Leftrightarrow [\![C]\!] \xrightarrow{s} [\![C']\!]$.*

Démonstration. Par induction sur la proposition 75. □

Proposition 77. ***(dite de nœud implicite)*** *Pour tout $C \in \mathcal{C}$, $C \sim [\![C]\!]$.*

Démonstration. Directe depuis la proposition 75. □

Proposition 78. *Pour tout $C \in \mathcal{C}$ et $C' \in \mathcal{C}$, $C \sim C' \Leftrightarrow [\![C]\!] \sim [\![C']\!]$*

Démonstration. $C \sim [\![C]\!]$ et $C' \sim [\![C']\!]$ (proposition 77) donc, par transitivité de $\sim$, $C \sim C' \Leftrightarrow [\![C]\!] \sim [\![C']\!]$ □

3.5.7 Relations entre nœuds

Il n'existe pas d'action que les nœuds ne peuvent effectuer. Il n'y a donc pas d'équivalence entre les relations de bisimulation au niveau $\mathcal{N} \times \mathcal{N}$. Autrement dit, toutes les bisimulations ont un sens différent lorsqu'elles concernent des nœuds ou des messages.

Les relations d'implication générales entre les bisimulations données par la proposition 25 page 69 et le corollaire 26 page 69 sont aussi les seules valables au niveau des nœuds. Nous les illustrons en figure 3.15 dans laquelle, bien que la relation d'implication soit transitive, la clôture transitive n'est pas indiquée.

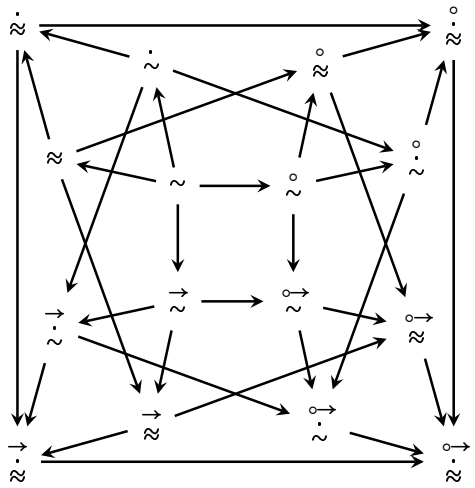


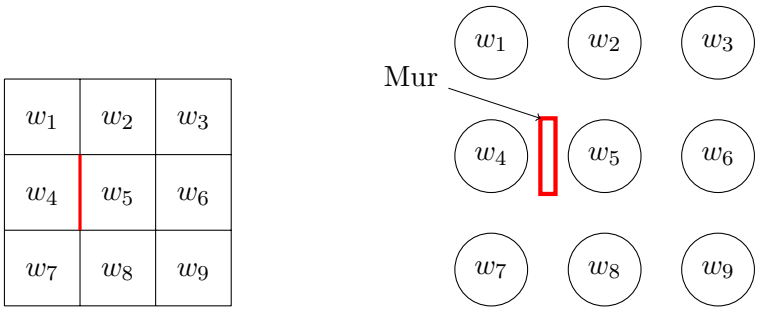
Fig. 3.15: Implication des relations de bisimilarité dans $\mathcal{N} \times \mathcal{N}$. $\rightarrow$ signifie implique.

3.5.8 Gestion de la mobilité

3.5.8.1 Mobilité par l'exemple

Mobilité des nœuds

Pour gérer la mobilité, nous supposons que l'espace dans lequel évoluent les nœuds peut être discrétisé. Prenons par exemple une fourmi (mobilité) équipée d'un capteur (calcul) se déplaçant dans l'espace décrit figure 3.16. Ses déplacements sembleront aléatoires à un observateur ; même si les déplacements sont contrôlés par le cerveau de la fourmi, il n'est pas possible d'en extraire un modèle de mobilité et les déplacements semblent donc



(a) Espace physique (b) Espace logique

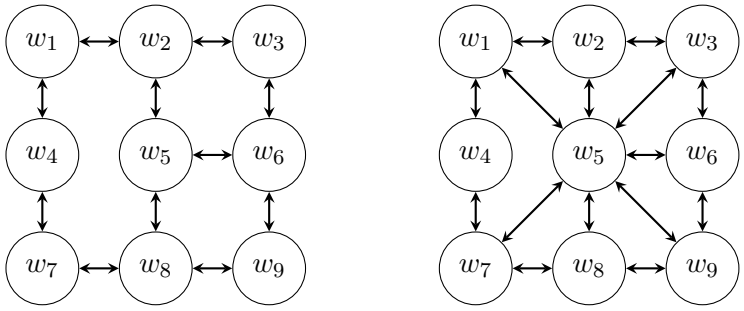
FIG. 3.16: Exemple d'espace

aléatoires[5]). Plus précisément, nous considérons que le nœud N correspondant à la fourmi exécute le processus

$$\breve{P} \stackrel{\text{def}}{=} \sum \vec{w}_i.\breve{P}$$

Si le graphe de mobilité de N est le graphe complet alors N est capable de se déplacer de toutes les positions vers toutes les positions. Dans ce cas, la mobilité de la fourmi ne respecte pas l'environnement physique tel que défini dans la figure 3.16 ; elle peut se déplacer par exemple de w_7 à w_6 instantanément, semblant alors se téléporter. Pour imposer à la fourmi le respect de l'environnement dans lequel elle évolue, nous utilisons un graphe de mobilité.

Le **graphe de mobilité** définit les déplacements possibles des nœuds dans l'environnement. Il donne ainsi de la cohérence à ses déplacements. La figure 3.17 donne deux exemples de graphe de mobilité possibles basés sur l'exemple de l'environnement décrit en figure 3.16. En l'occurrence, ils interdisent à la fourmi de se téléporter ou de traverser le mur.



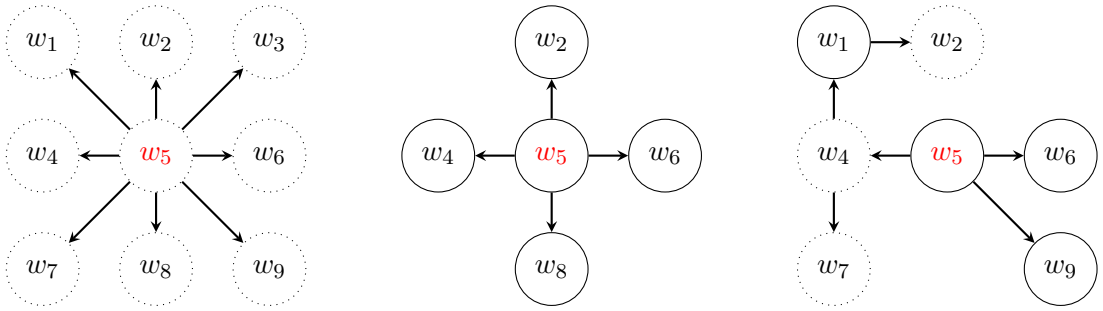
(a) Exemple 1 (b) Exemple 2

FIG. 3.17: Exemples de graphes de mobilité

Mobilité des messages

[5]On fait abstraction du modèle fourmi [NG09, MGS09], hors de propos ici.

Le **graphe de propagation d'un message** définit la manière dont ce dernier se propage à travers l'espace. Rappelons qu'un message peut se trouver à plusieurs positions simultanément. Comme le graphe de mobilité d'un nœud, il donne de la cohérence à la mobilité d'un message. Un graphe de propagation d'un message dépend de l'identité du nœud émetteur, de la position de celui-ci au moment où il émet et du canal utilisé pour la transmission. La figure 3.18 représente trois exemples de graphes de propagation possibles pour un message émis par un nœud depuis la position w_5 sur un canal $\alpha \in \ddot{\mathcal{A}}$. Afin d'alléger la représentation des graphes de propagation, on note en pointillés les sommets qui possèdent un arc sortant vers le PUITS ($v \in V$ est en pointillé ssi $(v, \text{PUITS}) \in E$). Le sommet rouge est celui depuis lequel le message est émis.



(a) Exemple 1 (b) Exemple 2 (c) Exemple 3

Fig. 3.18: Exemples de graphes de propagation de message

Classification des graphes de propagation

Dans un travail futur, il pourrait être intéressant de classer les graphes de propagation des messages en fonction de leurs caractéristiques. Par exemple, nous pouvons remarquer que dans l'exemple 1, le message émis depuis la position w_5 finit toujours par disparaître. Au contraire, dans l'exemple 2, le message ne disparaît jamais. Enfin, dans l'exemple 3, il est impossible de conclure.

Dans cette thèse nous considérons uniquement des graphes de propagation du type de l'exemple 1, c'est à dire dans lesquels les messages peuvent disparaître en tout point et à tout instant.

3.5.8.2 Étude d'un système sans connaissance de la mobilité des nœuds

Nous nous plaçons dans un contexte où tous les messages peuvent disparaître à tout instant. Nous montrons alors qu'étudier un système en considérant tous les cas de mobilité possibles des nœuds et des messages est équivalent à l'étudier sans mobilité.

Définition 79. *Un nœud N d'identité i situé à la position w est dit immobile si son graphe de mobilité G est tel que $G = (V, \varnothing)$). On note alors $N = {}_i[\![\breve{C}]\!]_w$. Si son identité i n'est pas utile, on l'omettra et on notera $[\![\breve{C}]\!]_w$.*

Soit la réécriture Φ_w dont l'objectif est de supprimer la mobilité des nœuds et de placer chacun d'eux à la position w.

- $\Phi_w(N_1 \,|||\, N_2) = \Phi_w(N_1) \,|||\, \Phi_w(N_2)$
- $\Phi_w(\llbracket \breve{C} \rrbracket_{w'}) = \llbracket \Phi_w(\breve{C}) \rrbracket_w$ pour tout $w' \in \mathcal{W}$
- $\Phi_w(\breve{C}_1 \parallel \breve{C}_2) = \Phi_w(\breve{C}_1) \parallel \Phi_w(\breve{C}_2)$
- $\Phi_w([\breve{P}]) = [\Phi_w(\breve{P})]$
- Inaction
  1. $\Phi_w(0) = 0$
- Action
  1. $\Phi_w(\alpha.\breve{P}) = \alpha.\Phi_w(\breve{P})$ si $\alpha \notin \vec{\mathcal{A}}$
  2. $\Phi_w(\alpha.\breve{P}) = \Phi_w(\breve{P})$ si $\alpha \in \vec{\mathcal{A}}$; on élimine les déplacements des nœuds.
- Somme $\Phi_w(\breve{P}_1 + \breve{P}_2) = \Phi_w(\breve{P}_1) + \Phi_w(\breve{P}_2)$
- Composition $\Phi_w(\breve{P}_1 \mid \breve{P}_2) = \Phi_w(\breve{P}_1) \mid \Phi_w(\breve{P}_2)$
- Restriction $\Phi_w(\breve{P} \setminus b) = \Phi_w(\breve{P}) \setminus b$.
- Renommage $\Phi_w(\breve{P}[c/b]) = \Phi_w(\breve{P})[c/b]$.

Proposition 80. *Soit N_i un ensemble de nœuds dont la mobilité est inconnue. On a* $N_1 \,|||\, N_2 \,|||\, ... \,|||\, N_k \overset{\rightarrow}{\sim} \Phi_w(N_1) \,|||\, \Phi_w(N_2) \,|||\, ... \,|||\, \Phi_w(N_k)$

La proposition 72 montre que l'étude du système $N_1 \,|||\, N_2 \,|||\, ... \,|||\, N_k$ avec comme mobilité des nœuds l'ensemble des combinaisons possibles des graphes de mobilité est équivalent (modulo les déplacements des nœuds, actions de l'ensemble $\vec{\mathcal{A}}$) à l'étude du système $\Phi_w(N_1) \,|||\, \Phi_w(N_2) \,|||\, ... \,|||\, \Phi_w(N_k)$ dans lequel les nœuds sont immobiles et tous ont la même position.

L'intérêt de la proposition précédente est de pouvoir être utilisée par exemple pour (i) montrer qu'un résultat reste valable qu'elle que soit la mobilité des nœuds et des messages, et (ii) montrer l'impossibilité d'un résultat qu'elle que soit la mobilité des nœuds et des messages. En effet, tout résultat (même d'impossibilité) prouvé par l'utilisation de la proposition 80 et de la réécriture associée est toujours valable ; i.e. reste vrai qu'elle que soit la mobilité des nœuds et des messages du système.

Démonstration. Montrons que l'étude d'une composition parallèle de nœuds qui se déplacent et dont on ne connaît ni la position ni la mobilité (et donc dont toutes les mobilités sont à considérer) est équivalente à l'étude de ce même système dans lequel tous les nœuds sont immobiles et à la même position et dans lequel tous les messages peuvent disparaître.

1. (Absence de mobilité $\Leftarrow$ Toutes les mobilités) De manière évidente, l'absence de mobilité est une mobilité particulière et est donc incluse dans toutes les mobilités.
2. (Absence de mobilité $\Rightarrow$ Toutes les mobilités) Nous raisonnons par l'absurde. Nous cherchons donc à montrer qu'il existe un comportement dans le système avec mobilité qui ne peut pas être simulé par une absence de mobilité ou inversement.
   - On suppose qu'il existe une mobilité qui appartient à l'ensemble des mobilités possibles telle qu'il existe des communications possibles entre les nœuds et telle qu'elles sont impossibles en l'absence de mobilité. Ce comportement est impossible car en l'absence de mobilité, les nœuds sont tous à la même position, et donc,

quelles que soient les communications qui prennent place dans le système avec mobilité, elles peuvent aussi avoir lieu dans le système sans mobilité.
– On suppose maintenant qu'il existe une mobilité qui appartient à l'ensemble des mobilités possibles telle qu'une communication peut échouer entre deux nœuds et telle qu'elle réussit systématiquement ("réussit systématiquement" est le contraire de "peut échouer") en l'absence de mobilité. Ce comportement est aussi impossible car la perte de message peut aussi se produire en l'absence de mobilité, et même lorsque les nœuds sont à la même position.

□

3.5.9 Perspectives de recherche

Nous présentons ici les pistes de réflexion que nous avons ouvertes concernant la mobilité des nœuds. Cette section n'est pas essentielle dans la définition du modèle mais ouvre des perspectives de recherche intéressantes.

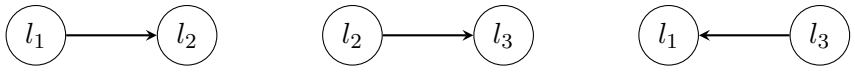
3.5.9.1 Construction de méta-graphes de communication

Intérêt. L'objectif ici est d'étudier les conséquences de la mobilité d'un nœud et/ou des messages qu'il peut émettre sur les interactions globales possibles entre les nœuds et donc sur ce qu'ils peuvent réaliser globalement au travers de leurs échanges éventuels.

Or, la mobilité des éléments d'un système et donc la capacité d'interaction et de coopération de ces éléments s'exprime par la mobilité :
– des processus, indépendamment les uns des autres,
– des messages ; la mobilité d'un message dépend de la position depuis laquelle il est émis, du nœud émetteur (la portée de chaque nœud peut être différente), et du canal sur lequel il est émis.

Exemple : On considère le cas trivial de trois nœuds N_1, N_2 et N_3 évoluant dans le même monde.

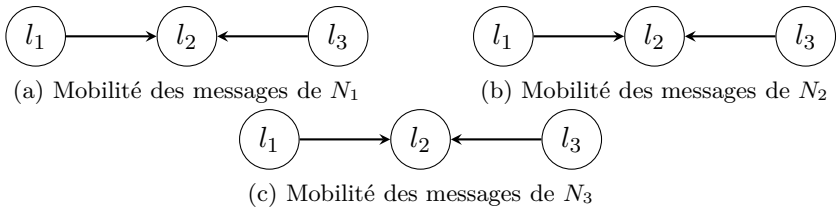
Évolution des processus et des messages. La mobilité des nœuds N_1, N_2 et N_3 est présentée figure 3.19. La mobilité des messages associés est présenté figure 3.20.



(a) Mobilité de N_1 (b) Mobilité de N_2 (c) Mobilité de N_3

Fig. 3.19: Mobilité des processus

Évolution du système. Les évolutions possibles d'un système (en terme de position des nœuds) sont représentées par un graphe orienté dont les sommets correspondent à une configuration possible du système, i.e de l'ensemble des nœuds qui le composent, et les arcs aux transitions entre ces configurations. L'ensemble des sommets de ce graphe correspond



(a) Mobilité des messages de N_1 (b) Mobilité des messages de N_2

(c) Mobilité des messages de N_3

FIG. 3.20: Mobilité des messages

à l'ensemble des états possibles du système. L'évolution en terme de position des nœuds du système des figures 3.19 et 3.20 peut-être vue figure 3.21.

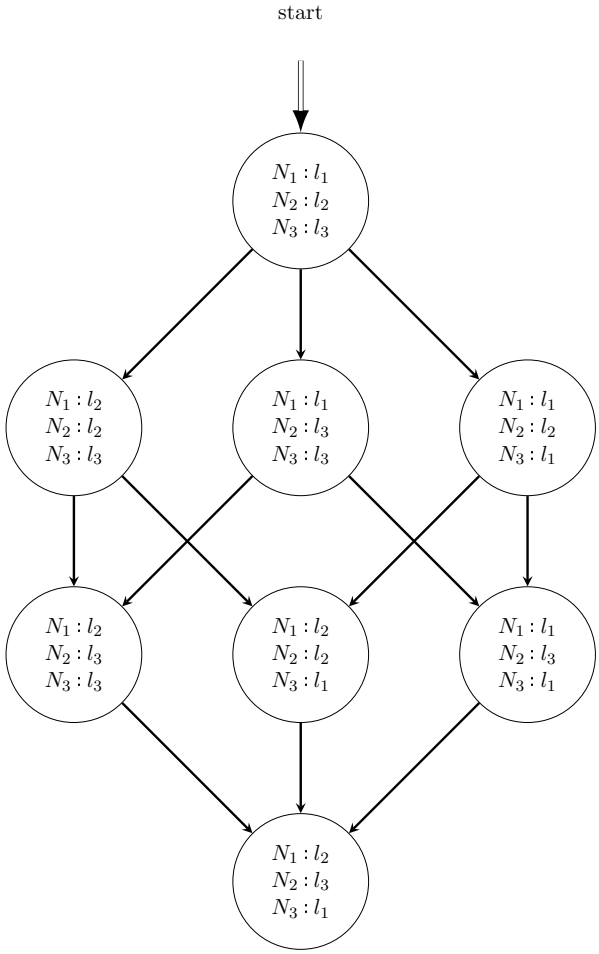


FIG. 3.21: Évolution du système en terme de position des nœuds

La figure 3.21 est le produit cartésien des graphes de la figure 3.19. Associé aux graphes de mobilité des messages, il nous permet de contruire un graphe qui représente les évolutions possibles des interactions elles aussi éventuelles entre les nœuds (cf. figure 3.22).

Nous expliquons sa construction par la construction de deux de ses sommets.

Au départ, les positions de N_1, N_2 et N_3 sont respectivement l_1, l_2 et l_3. Grâce au graphe de mobilité des messages, on peut constater qu'à ces positions, les messages émis par N_1 peuvent atteindre l_2 (la position de N_2), ceux de N_2 ne peuvent pas atteindre les nœuds qui se trouvent à une position différente de la sienne et N_3 peut émettre des message qui peuvent atteindre l_2 (la position de N_2). Ces informations permettent de conclure que dans l'état initial, N_1 peut envoyer des messages uniquement à N_2, que N_2 ne peut en envoyer ni à N_1, ni à N_3 et enfin que N_3 ne peut en envoyer qu'à N_2 (cf. sommet "start" de la figure 3.22).

La situtation finale (lorsqu'il n'existe plus déplacements de possibles pour les nœuds) fait apparaître qu'un échange éventuel de messages peut avoir lieu de N_2 vers N_1 et de N_3 vers N_1. N_1 est à la position l_2 et les messages qu'il émet ne peuvent être reçus que par les nœuds qui se trouvent à la même position (il n'y en a aucun). Les messages émis par N_2 qui se trouve à la position l_3 peuvent atteindre les nœuds de sa propre position et ceux situés en l_2 ; c'est à dire le nœud N_1. De manière similaire, les messages émis par N_3 ne peuvent atteindre que N_1.

L'exemple que nous donnons dans cette section et que nous représentons en figure 3.22 permet notamment de conclure qu'il n'existe pas forcément de moment pendant lequel N_1 est succeptible de pouvoir envoyer un message qui pourrait atteindre N_3. En effet les évolutions successives possibles représentées par les arcs bleus en figure 3.22 ne permettent jamais à N_1 d'envoyer un message qui pourrait atteindre N_3.

Ce que l'on pourrait remarquer ou mettre en avant à partir d'un graphe du type de celui de la figure 3.22. On pourrait utiliser ce type de graphe pour montrer des propriétés qui sont vraies quelle que soit la mobilité des nœuds :

1. N_i ne peut jamais communiquer avec N_j.
2. les nœuds N_i et N_j ne peuvent pas communiquer symétriquement.
3. il n'existe pas de chemin au cours du temps entre N_i et N_j.

Les résultats précédents sont des résultats d'impossibilité car on ne peut jamais garantir que deux nœuds vont réussir à communiquer.

Exemple : Soit la fonction $succ(N)$ qui retourne l'ensemble des positions atteignables par le nœud N. Un exemple de résultat d'imposibilité est le suivant. Soient N_1 et N_2 deux nœuds. Si $succ(N_1) \cap succ(N_2) = \varnothing$ (N_1 et N_2 ne sont pas et ne seront jamais à la même position) et si pour tout $\alpha \in \mathcal{L}(N_1) \cap \overset{\circ}{\bigwedge}$, pour tout $w \in succ(N_1)$ et pour tout $w' \in succ(N_2)$, $w' \notin V(\mathsf{G}(1, w, \alpha))$ avec $\mathsf{G}(1, w, \alpha)$ l'ensemble des sommets que peuvent atteindre les messages envoyés par le nœud N_1 à la position w en utilisant le canal α (quel que soit le canal de communication utilisé, les messages envoyés depuis l'ensemble des positions possibles de N_1 n'atteignent aucune des positions que peut avoir N_2) alors, quelle que soit la mobilité réelle de N_1 et N_2, N_2 ne peut pas recevoir de message de N_1.

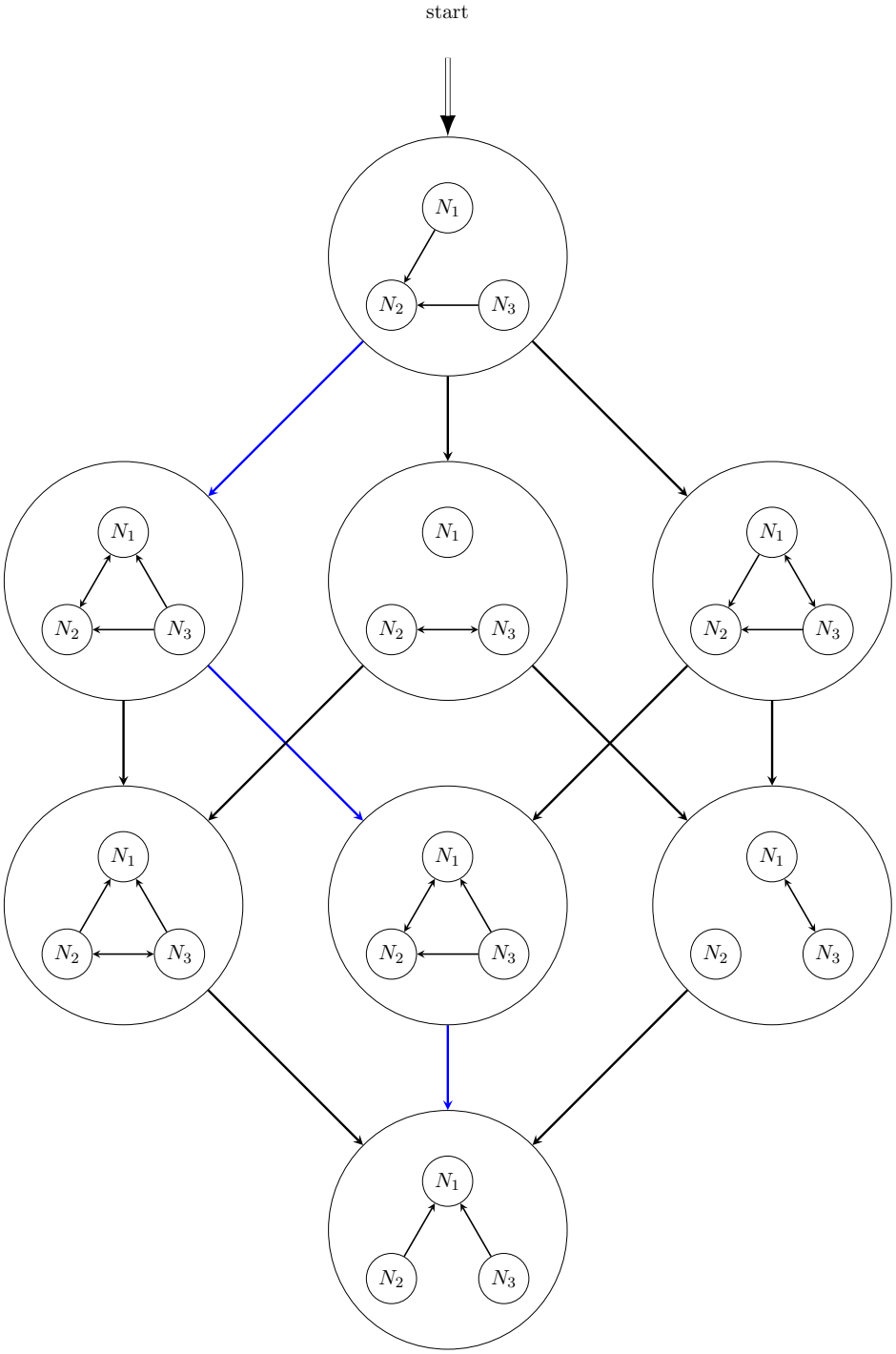


FIG. 3.22: Évolution potentielle du système en terme d'interactions locales directes entre les nœuds

Chapitre 4

Primitives, support d'exécution et applications de référence formellement validés

Sommaire

Le développement d'application pour les réseaux mobiles ad hoc est un challenge pour plusieurs raisons. Tout d'abord, l'hétérogénéité des dispositifs sur lesquels l'application est susceptible d'être déployée peut-être très importante (PCs portables, téléphones portables, PDAs, robots, etc.). Ensuite, les technologies de communication dont les nœuds disposent sont elles aussi très variées (Wi-Fi, Bluetooth, ZigBee, etc.). Enfin, la topologie du réseau évolue en permanence, ce dernier pouvant même être temporairement ou durablement déconnecté. Toutes ces caractéristiques rendent difficiles le développement et le déploiement d'applications dans ce contexte. Afin de simplifier ces opérations, des *middlewares* ont été développés.

Un *middleware* est une couche logicielle qui s'appuie sur un système d'exploitation et qui fournit un ensemble de primitives (voire de services) aux applications qui l'utilisent. Dans un contexte MANet, ses objectifs sont très variés. Il peut par exemple essayer d'apporter des solutions à la mobilité qui provoque des créations et des disparitions de liens de communication entre les nœuds [AMSSHS08]. Il peut aussi chercher à gérer l'hétérogénéité des dispositifs, que ce soit en terme de quantité d'énergie disponible [JR06, SCB+06], de technologie de communication [AHP02, Her03] ou de système d'exploitation [BCM05, GM07, MMH05]. Il peut enfin tenter de fournir des garanties en terme de sécurité des échanges [PAD+05, SR04, KSS+01]. Des *middlewares* sont aussi créés pour des contextes plus spécifiques, par exemple pour permettre la diffusion d'informations dans le cadre d'opérations de secours en montagne [PAD+05]. La diversité des difficultés a donné lieu à la création d'une multitude de *middlewares*. Nous avons précisé les objectifs et le fonctionnement de certains d'entre eux dans le chapitre 2 page 15.

Dans ce chapitre, nous modélisons le *middleware* que nous avons créé et les primitives originales pour le domaine des iMANets, (domaine introduit dans le chapitre 1) sur lesquelles il repose. Nous présentons ensuite deux applications de référence qui l'utilisent et la validation formelle de l'une d'entre elles grâce à CiMAN (cf. chapitre 3). Nous présentons enfin l'architecture de notre support d'exécution.

4.1 Définitions d'éléments de sécurité

Avant de présenter les primitives que nous avons conçues, nous introduisons quelques concepts fondamentaux de sécurité. Ils nous permettent de mettre en lumière des caractéristiques importantes de ces primitives.

4.1.1 Identité

Dans un dictionnaire tel que le Larousse, l'identité possède plusieurs définitions dont les deux suivantes :

1. *Identité : n.f. Caractère permanent et fondamental de quelqu'un, d'un groupe, qui*

fait son individualité, sa singularité : Personne qui cherche son identité. Identité nationale.

2. *Identité : n.f. Ensemble des données de fait et de droit qui permettent d'individualiser quelqu'un (date et lieu de naissance, nom, prénom, filiation, etc.) : Rechercher l'identité d'un noyé.*

La première de ces définitions se rapproche de notre définition de l'identité. La seconde se rapproche quant à elle plus de notre définition d'un identifiant, le terme d'identifiant n'ayant pas encore été validé par l'Académie française.

Définition 81. *L'**identité** est une notion abstraite qui définit une entité. Elle est unique pour chaque entité et est invariable au cours du temps.*

Exemple : Il est difficile de donner un exemple d'identité puisqu'il s'agit d'une notion abstraite. Tout exemple donné ici serait un exemple d'identifiant et non d'identité.

4.1.2 Identifiant

Définition 82. *L'**identifiant** est un ensemble de données/documents qu'une entité présente pour se faire connaître ou reconnaître. Le plus souvent, les identifiants sont uniques pour chaque entité et invariables au cours du temps.*

Exemples : Un *login* est un identifiant, tout comme un triplet (nom, prénom, date de naissance), un numéro INSEE, etc. Une pièce d'identité est un identifiant particulier car elle est aussi un authentifiant (cf. définition suivante).

4.1.3 Authentifiant et authentification

Définition 83. *L'**authentifiant** est un ensemble de données/documents qu'une entité peut fournir pour permettre son authentification.*

Définition 84. *L'**authentification** est un procédé qui permet à l'entité qui l'exécute de valider le lien entre l'identité de l'entité et l'identifiant qu'elle présente.*

Exemple : La photographie qui se trouve sur une pièce d'identité est considérée comme un authentifiant (bien que deux personnes puissent se ressembler suffisamment pour faire échouer le processus d'authentification). Elle est censée prouver que l'identifiant (ensemble des informations qui se trouvent sur la pièce d'identité) est bien lié à l'entité qui la présente. La procédure d'authentification est généralement réalisée par un individu, d'où les risques d'erreur.

4.1.4 Anonymat

Définition 85. *On dit que l'**anonymat** au sein d'un ensemble de nœuds (un groupe) N_i est **partiellement préservé** s'il n'existe pas de nœud N' hors de ce groupe capable de faire correspondre aux messages qu'il voit passer les nœuds qui les ont émis. On dit alors que l'anonymat de chacun des nœuds du groupe est partiellement préservé.*

Définition 86. *On dit que l'**anonymat** d'un nœud N est **totalement préservé** s'il n'existe pas de nœud N' soit capable de faire correspondre aux messages qu'il voit passer le nœud N.*

Les nœuds qui composent les réseaux mobiles ad hoc que nous modélisons peuvent être séparés en deux groupes ; ceux qui possèdent notre support d'exécution (éléments de $\mathcal{N}$), et ceux qui ne le possèdent pas (éléments de $\mathcal{N}'$). Nous disons donc que l'anonymat d'un nœud $N \in \mathcal{N}$ est partiellement préservé s'il n'existe pas de nœud $N' \in \mathcal{N}'$ capable de faire correspondre aux messages qu'il voit passer le nœud $N \in \mathcal{N}$ qui les a émis. De la même manière, nous définissons que l'anonymat d'un nœud N est totalement préservé s'il n'existe pas de nœud N' capable de faire correspondre aux messages qu'il reçoit le nœud N qui les a émis. Ce dernier type d'anonymat n'est pas lié à une notion d'appartenance à un groupe.

4.2 Primitives, *middleware* et *applet*

Le *middleware* que nous avons créé, et que nous supposons installé sur l'ensemble des nœuds du réseau, repose sur des primitives de communication (à base de *broadcast*) permettant d'assurer l'interaction entre ces nœuds. Il fournit un ensemble de primitives évoluées et originales pour le développement d'applications dans un réseau ad hoc fortement mobile.

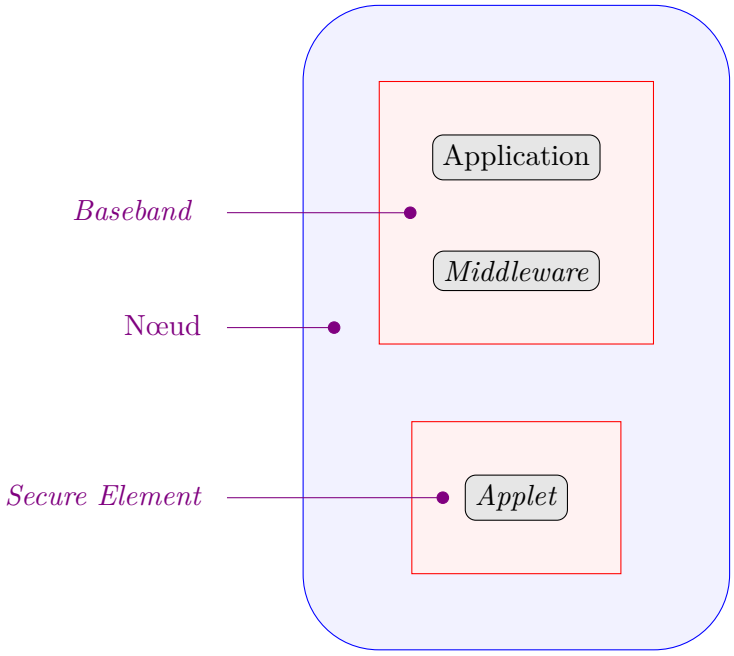


FIG. 4.23: Vision schématique des nœuds du réseau

Nous supposons dans la suite que les nœuds du réseau sont tels que représentés sur la figure 4.23. Ils sont donc composés de deux unités de calcul, la première nommée *baseband*

correspond par exemple au processeur d'un téléphone mobile, la seconde étant quant à elle un *Secure Element*, par exemple la carte SIM de ce même téléphone.

Le support d'exécution que nous avons créé (**MiMAN - *Middleware* for highly Mobile Ad hoc Networks**) propose un ensemble de primitives élémentaires que nous modélisons dans ce chapitre grâce à CiMAN. Au fur et à mesure de l'introduction de nouvelles primitives, nous les intégrons à l'expression formelle de MiMAN que nous appelons $\breve{\mathbb{M}}$. Par ailleurs, la sécurité du système repose sur la présence sur chacun des nœuds d'un *Secure Element*. Ce dispositif, contraint en terme de capacité de calcul et de communication, est capable d'exécuter un programme que l'on nomme *applet*. Dans la suite, nous appelons **AiMAN** l'*applet* que nous avons créée et $\breve{\mathbb{A}}$ sa modélisation formelle.

Pour chacune des primitives que nous introduisons et que nous modélisons, nous présentons un diagramme de séquence représentant les échanges entre les différentes entités impliquées. Nous proposons pour chacune l'expression formelle d'un exemple d'application $\breve{P}$ l'utilisant, du *middleware* $\breve{\mathbb{M}}$, du nœud $\breve{N}$ et lorsque nécessaire de l'*applet* $\breve{\mathbb{A}} \in \breve{\mathcal{P}}$ exécutée dans un *Secure Element*.

4.2.1 Primitives de communication *Over The Air*

Les nœuds qui forment un réseau mobile ad hoc peuvent être de nature très diverse. Une de leurs caractéristiques commune cependant est la présence d'une technologie de communication radio associée à une antenne omnidirectionnelle. Les primitives naturellement liées à ce type de communication sont une primitive d'émission en *broadcast* (`send`) et de réception (`receive`). Nous les décrivons dans cette section.

4.2.1.1 Primitive `send` $= \overline{s}$

Le *middleware* que nous avons créé propose une primitive d'émission de message en mode *broadcast*. Cette primitive, que nous avons nommée `send`, se contente de "déposer un message" dans les airs[1] (en mode non connecté donc).

Dans la suite du document APP est un *flag* permettant de différencier les types de message, i.e. de multiplexer les messages sur un canal. C'est une valeur dans CiMAN. Ici le type le message APP signifie que le message est de niveau applicatif. A sa réception, le *middleware* peut effectuer le traitement approprié, i.e. le faire suivre à l'application.

Le diagramme de séquence de la primitive `send` est présenté figure 4.24.

Modélisation. Comme le décrivent les définitions des processus, unités de calcul et nœuds de la figure 4.25, l'application P_s qui est exécutée au-dessus du *middleware* $\breve{M}_s$ au sein de la même unité de calcul dans le nœud N passe le message à envoyer au *middleware* $\breve{M}_s$ qui l'émet en *broadcast*. Une application désirant envoyer un message le fait donc *via* le *middleware*.

Trivialement,

[1] Dans la suite, on parlera de message *Over The Air* (OTA)

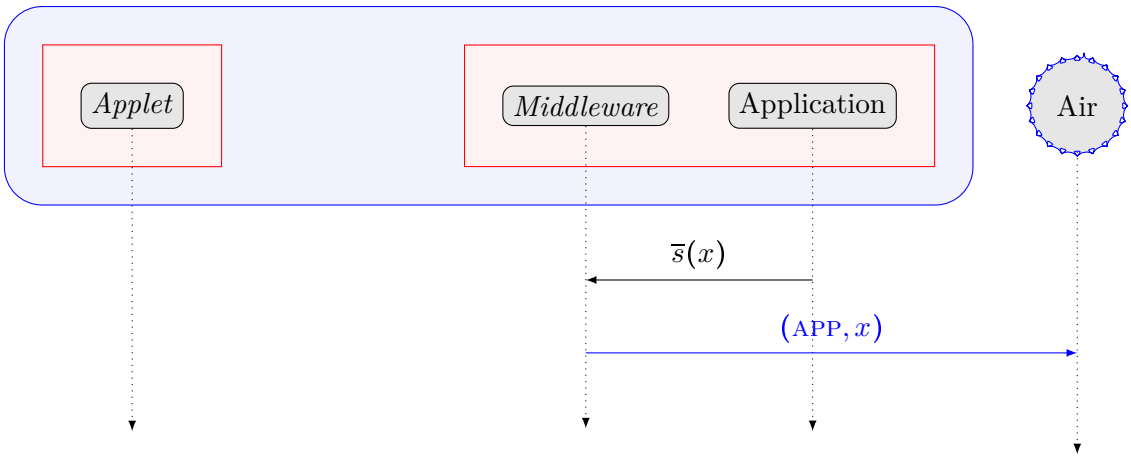


FIG. 4.24: Diagramme de séquence de la primitive `send`.

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[(\breve{M}_s \mid P_s)\setminus_s]]\!]$
Middleware	$\breve{M}_s$	$\stackrel{\text{def}}{=}$	$s(x).\ \overset{\circ}{a}\overset{\circ}{}\ (\text{APP}, x).\breve{M}_s$
Application	P_s	$\stackrel{\text{def}}{=}$	$\overline{s}(v).0$

FIG. 4.25: Modélisation de la primitive `send`.

$$
\begin{array}{lll}
N & \stackrel{\text{def}}{=} & [\![[(\breve{M}_s \mid P_s)\setminus_s]]\!] \\
 & \stackrel{\tau}{\rightarrow} & [\![[(\overset{\circ}{a}\overset{\circ}{}\ (\text{APP}, v).\breve{M}_s \mid 0)\setminus_s]]\!] \\
 & \stackrel{\overset{\circ}{a}\overset{\circ}{}(\text{APP},v)}{\rightarrow} & [\![[\breve{M}_s \mid 0]]\!]
\end{array}
$$

L'utilisation de la primitive `send` ($\overline{s}$) a donc bien pour effet l'envoi d'un message par la carte radio du nœud.

Ajout de la primitive `send` au *middleware* MiMAN. Au fur et à mesure de la définition des primitives, nous les ajoutons dans le processus $\breve{\mathbb{M}}$ qui modélise MiMAN. Sa première version est présenté figure 4.26.

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[(\breve{\mathbb{M}} \mid P)\setminus_s]]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$\breve{\mathbb{M}}_s$	
	$\breve{\mathbb{M}}_s$	$\stackrel{\text{def}}{=}$	$s(x).\ \overset{\circ}{a}\overset{\circ}{}\ (\text{APP}, x).\breve{\mathbb{M}}_s$	`send`

FIG. 4.26: Ajout de la primitive `send` au *middleware*

4.2.1.2 Primitive `receive` = r

Cette primitive permet à l'application de recevoir un message provenant de l'interface radio. Un message à destination d'une application est remonté par un mécanisme de *callback* depuis le *middleware* vers le niveau applicatif et est capturé par l'action $\overline{r}$ par la couche supérieure.

Le diagramme de séquence de la primitive `receive` est présenté figure 4.27.

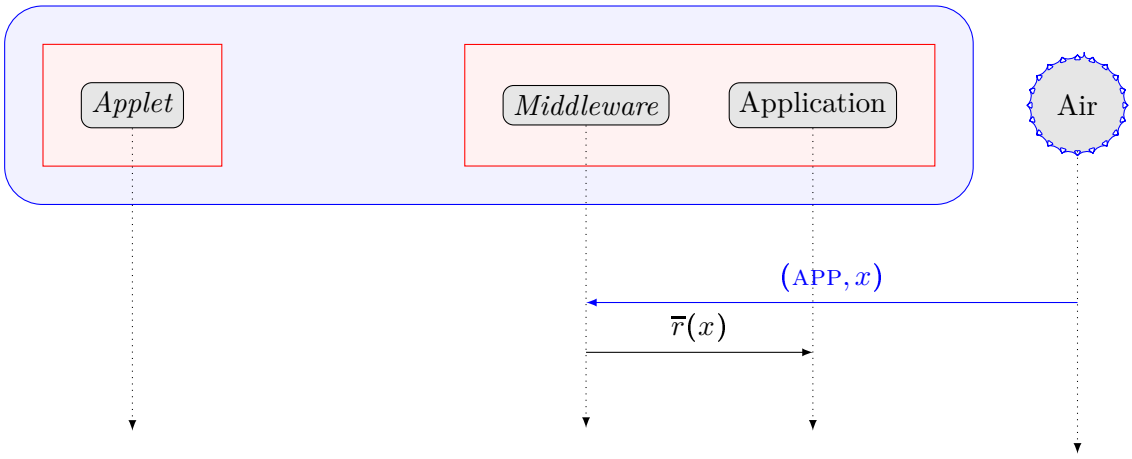


FIG. 4.27: Diagramme de séquence de la primitive `receive`.

Modélisation. Nous définissons formellement la primitive `receive` en figure 4.28. Le *middleware* $\breve{M}_r$ reçoit un message provenant de la radio et le fait suivre à l'application P_r grâce à l'action $\overline{r}$.

L'écriture d'une réception de message de la forme (APP, x) avec x variable est une facilité syntaxique. Une réception par cette transition n'est possible que si le premier élément est bien égal à APP. Afin de respecter les règles de réception du modèle, nous introduisons pour chaque réception d'un message avec *flag* un embranchement qui permet de recevoir tous les messages qui ne contiennent pas ce *flag* (cf. figure 4.28). La valeur OTHERS est donc une simplification syntaxique qui capte l'ensemble des valeurs non traitées explicitement dans les autres branches de l'opérateur +. Dans l'expression suivante, on a donc :

$$\mathring{a}\ (\text{OTHERS}) = \sum_{y \neq (\text{APP},x)} \mathring{a}\ (y)$$

On suppose ici que tous les nœuds utilisent la même bande de fréquence pour les communications radio et donc qu'ils communiquent tous sur le canal $\mathring{a}\ /\ \mathring{a}$.

Ajout de la primitive `receive` au *middleware* MiMAN. On notera que le *middleware* a pour l'instant pour rôle de permettre à l'application de réaliser des communications sans fil.

Nous intégrons maintenant la primitive `receive` au *middleware* MiMAN. Le *middleware* est à partir de maintenant composé de plusieurs processus en parallèle (cf. figure 4.29).

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![(\breve{M}_r \mid P_r)\setminus_r]\!]$
Middleware	$\breve{M}_r$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{APP}, x).\overline{r}(x).\breve{M}_r$
		$+$	$\mathring{a}\ (\text{OTHERS}).\breve{M}_r$
Application	P_r	$\stackrel{\text{def}}{=}$	$r(x).0$

FIG. 4.28: Modélisation de la primitive `receive`.

Dans un premier temps, il est composé de deux processus nommés $R\breve{I}M$ (*Radio Input Manager*) et $A\breve{I}M$ (*Application Input Manager*). $R\breve{I}M$ est le processus qui écoute les messages provenant de l'interface radio et $A\breve{I}M$ et le processus qui écoute les actions de l'application. Dans un second temps, un processus permettra de gérer les messages provenant de l'*applet*.

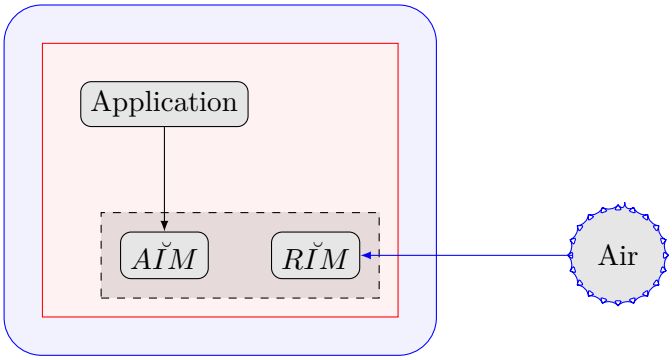


FIG. 4.29: Processus qui composent le *middleware*

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![(\breve{\mathbb{M}} \mid P)\setminus_{s,r}]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$R\breve{I}M \mid A\breve{I}M$	
avec	$R\breve{I}M$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{APP}, x).\overline{r}(x).R\breve{I}M$	`receive`
		$+$	$\mathring{a}\ (\text{OTHERS}).R\breve{I}M$	
et	$A\breve{I}M$	$\stackrel{\text{def}}{=}$	$s(x).\ \mathring{a}\ \mathring{}\ (\text{APP}, x).A\breve{I}M$	`send`

FIG. 4.30: Ajout de la primitive `receive` au *middleware*.

4.2.2 Primitives évoluées avec diffusion de l'identifiant OTA

Nous avons créé un ensemble de primitives évoluées spécifiques du contexte iMANet. Ces primitives utilisent le *Secure Element* dont les nœuds disposent afin de fournir de la sécurité et des garanties dans le réseau. Les garanties offertes par ces primitives sont discutées en section 4.3 page 131 au travers d'exemples d'algorithmes de comptage qui les utilisent.

Dans la suite du document, ID (Identifier), G_CI (Get Ciphered Identifier), P_CI (Provide Ciphered Identifier), B_CI (Broadcast Ciphered Identifier) et R_I (Receive Identifier) sont des *flags* (de la même manière que APP dans la section précédente).

4.2.2.1 Primitive `sendId` = $\overline{si}$

Cette primitive envoie par *broadcast* l'identifiant du nœud, chiffré par une clef symétrique. L'identifiant et la clef de chiffrement sont stockés dans le *Secure Element* embarqué sur le nœud et la clef de chiffrement n'est jamais accessible à l'extérieur du *Secure Element*. Nous supposons que tous les *Secure Elements* partagent cette clef symétrique.

Le diagramme de séquence de la primitive `sendId` est présenté figure 4.31.

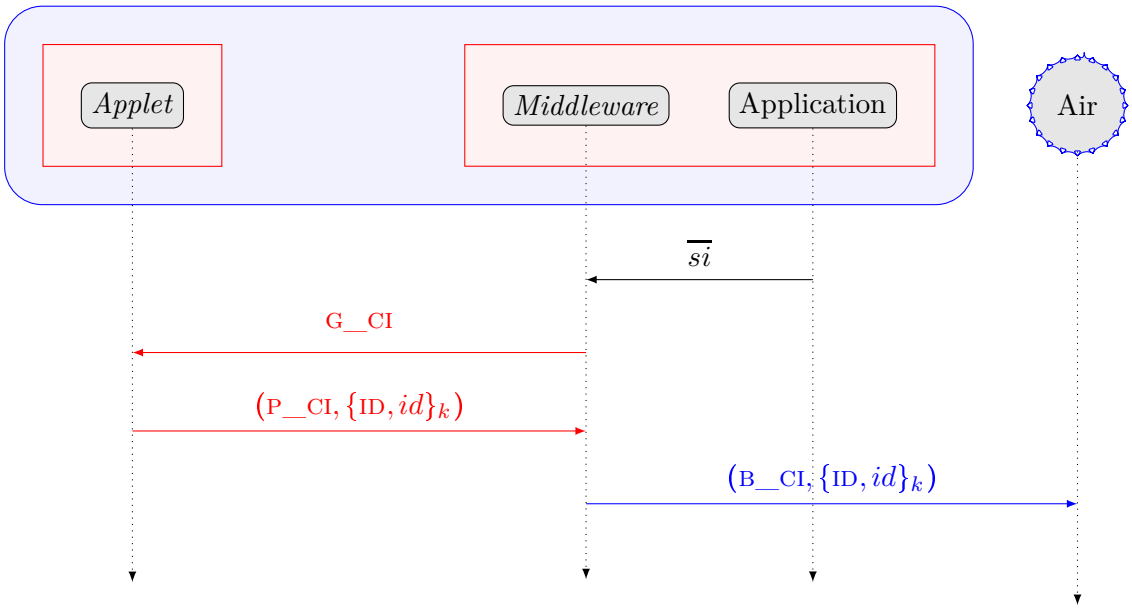


FIG. 4.31: Diagramme de séquence de la primitive `sendId`.

Modélisation. Comme le décrivent les définitions des processus, unités de calcul et nœuds de la figure 4.32, l'application P_{si} effectue l'action $\overline{si}$ qui est capturée par le processus $\breve{M}'_{si}$ du *middleware* $\breve{M}_{si}$. Il envoie alors à l'*applet* $\breve{A}_{id}$ une requête afin qu'elle construise le message correspondant, à savoir $\{\text{ID}, id\}_k$. Dans ce message, ID est un *flag* indiquant le type du message et id est la valeur de l'identifiant du nœud stocké dans le *Secure Element*. A sa réception, le *middleware* $\breve{M}_{si}$ le diffuse *Over The Air*.

Au niveau du *middleware*, une action $\overline{si}$ effectuée par l'application doit être capturée. Le *middleware* doit en outre être à l'écoute des messages provenant de l'*applet*. Dans toutes les modélisations de primitives qui suivent, nous découpons donc le processus qui modélise la partie *middleware* $\breve{M}_p$ d'une primitive p en deux processus $\breve{M}'_p$ et $\breve{M}''_p$ avec $\breve{M}_p = \breve{M}'_p \mid \breve{M}''_p$. Chacun d'eux est dédié à l'écoute exclusive soit de l'*applet*, soit de l'application, soit des messages OTA. Ce découpage correspond au découpage des processus dans le *middleware* $\breve{\mathbb{M}}$: lors de la modélisation des primitives dites actives (`send`, `sendId`, etc...), un processus est consacré à l'écoute des actions de l'application et l'autre à l'écoute des messages provenant de l'*applet* ; lors de la modélisation des primitives dites passives (ou *callback* au niveau de l'application), un processus est chargé de l'écoute de l'applet et l'autre de l'écoute des messages OTA. Ce découpage est directement lié aux interactions entre entités que font apparaître les primitives.

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id}] \parallel [(\breve{M}_{si} \mid P_{si})\backslash_{si}]]\!]$
Middleware	$\breve{M}_{si}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{si} \mid \breve{M}''_{si}$
avec	$\breve{M}'_{si}$	$\stackrel{\text{def}}{=}$	$si.\ddot{a}(\text{G\_\_CI}).\breve{M}'_{si}$
et	$\breve{M}''_{si}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CI}, x).\ \mathring{a}\ (\text{B\_\_CI}, x).\breve{M}''_{si}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{si}$
Applet	$\breve{A}_{id}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_CI}).\ddot{a}(\text{P\_\_CI}, \{\text{ID}, id\}_k).\breve{A}_{id}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id}$
Application	P_{si}	$\stackrel{\text{def}}{=}$	$\overline{si}.0$

FIG. 4.32: Modélisation de la primitive `sendId`.

On peut remarquer que des messages provenant de l'*applet* peuvent être perdus par le *middleware*. En effet, après une action de réception $\dot{a}(\text{P\_\_CI}, x)$, le processus $\breve{M}''_{si}$ doit effectuer une action $\mathring{a}\ (\text{B\_\_CI}, x)$ avant de pouvoir recevoir à nouveau sur le canal $\dot{a}$. Pour éviter ces pertes éventuelles, on utilise la réécriture définie dans le chapitre précédent qui permet l'utilisation du processus $\breve{CR}_{\dot{a},\overline{\alpha}}$ pour le *buffering* des messages reçus et leur redistribution par des actions de type $\overline{\alpha}$. On a aussi :

$$\begin{aligned}\Phi_{\dot{a},\overline{\alpha}}(\breve{M}''_{si}) &\stackrel{\text{def}}{=} (\breve{CR}_{\dot{a},\overline{\alpha}} \mid \phi_{\dot{a},\overline{\alpha}}(\breve{M}''_{si}))\backslash_{\alpha} \\ &= (\breve{CR}_{\dot{a},\overline{\alpha}} \mid (\alpha(\text{P\_\_CI}, x).\ \mathring{a}\ (\text{B\_\_CI}, x).\phi_{\dot{a},\overline{\alpha}}(\breve{M}''_{si}) + \alpha(\text{OTHERS}).\phi_{\dot{a},\overline{\alpha}}(\breve{M}''_{si})))\backslash_{\alpha}\end{aligned}$$

Les messages émis par un nœud ne divulguent pas son identifiant id, mais ils valent tous $\{\text{ID}, id\}_k$. Tous les nœuds (même ceux qui ne disposent pas d'un *Secure Element* qui exécute l'*applet* $\breve{A}_{id}$) peuvent donc reconnaître un nœud aux messages toujours identiques qu'il émet. En effet, le *flag* ID, l'identifiant id et la clef de chiffrement sont invariables. Les messages envoyés *Over The Air* sont donc tous identiques, devenant alors un identifiant

si on se réfère à la définition 82.

Ajout de la primitive `sendId` au *middleware* MiMAN et à l'*applet* AiMAN. Afin de gérer les messages provenant de l'*applet*, nous ajoutons le processus $Ap\breve{I}M$ (*Applet Input Manager*) au *middleware* $\breve{\mathbb{M}}$ (cf. figure 4.33).

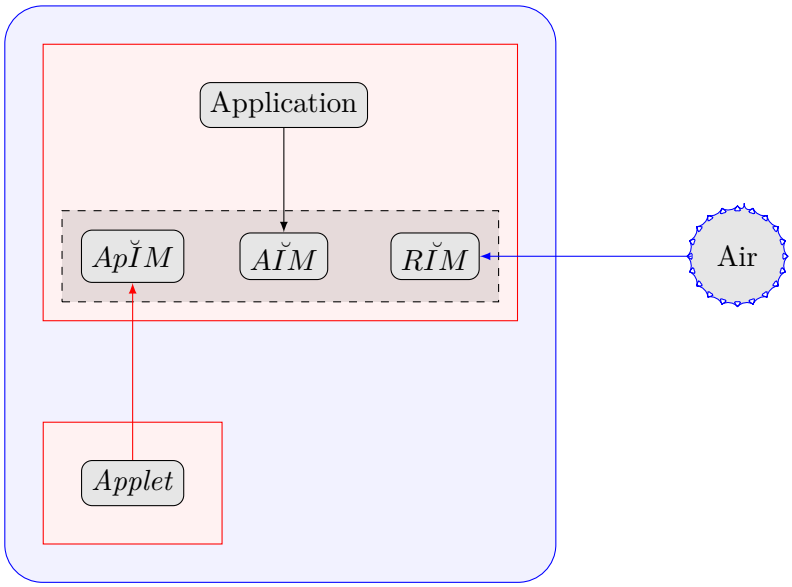


FIG. 4.33: Processus qui composent le *middleware*

Nous intégrons maintenant la primitive `sendId` au *middleware* MiMAN et nous introduisons la première version de l'*applet* AiMAN (cf. figure 4.34).

4.2.2.2 Primitive `receiveId` = ri

Cette primitive indique à l'application que le nœud a reçu un paquet d'identification construit par le *Secure Element* qui contient l'identifiant id' (cf. figure 4.35). Rien n'indique cependant si ce message provient directement du nœud sur lequel il a été généré ; il a pu être relayé par un nœud tierce. Cette primitive permet donc simplement de conclure (i) qu'il existe dans le réseau un nœud dont l'identifiant est id', et (ii) qu'il a existé un chemin entre ce nœud et le nœud récepteur du message.

Le diagramme de séquence de la primitive `receiveId` est présenté figure 4.35.

Modélisation. Comme le décrivent les définitions des processus, unités de calcul et nœuds de la figure 4.36, lors de la réception d'un message *Over The Air* de la forme $(\text{B\_CI}, x)$ (avec x variable) par la partie $\breve{M}'_{ri}$ du *middleware* $\breve{M}_{ri}$, ce dernier le fait suivre à l'*applet* $\breve{A}_{id}$ qui le déchiffre et retourne au *middleware* l'identifiant qu'il contient. Cet identifiant est remonté par le *middleware* à l'application P_{ri} par une action de type $\overline{ri}$. Cette action est capturée par l'application qui peut alors effectuer le traitement correspondant

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{\mathbb{A}}_{id}] \parallel [(\breve{\mathbb{M}} \mid P)\setminus_{s,r,si}]]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$R\breve{I}M \mid A\breve{I}M \mid \Phi_{\dot{a},\overline{\alpha}}(Ap\breve{I}M)$	
avec	$R\breve{I}M$	$\stackrel{\text{def}}{=}$	$\mathring{a}\,(\text{APP}, x).\overline{r}(x).R\breve{I}M$	`receive`
		$+$	$\mathring{a}\,(\text{OTHERS}).R\breve{I}M$	
et	$A\breve{I}M$	$\stackrel{\text{def}}{=}$	$s(x).\ \mathring{a}\ (\text{APP}, x).A\breve{I}M$	`send`
		$+$	$si.\ddot{a}(\text{G\_\_CI}).A\breve{I}M$	`sendId`
et	$Ap\breve{I}M$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CI}, x).\ \mathring{a}\ (\text{B\_\_CI}, x).Ap\breve{I}M$	`sendId`
		$+$	$\dot{a}(\text{OTHERS}).Ap\breve{I}M$	
Applet	$\breve{\mathbb{A}}_{id}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_CI}).\ddot{a}(\text{P\_\_CI}, \{\text{ID}, id\}_k).\breve{\mathbb{A}}_{id}$	`sendId`
		$+$	$\dot{a}(\text{OTHERS}).\breve{\mathbb{A}}_{id}$	

FIG. 4.34: Ajout de la primitive `sendId` au *middleware*.

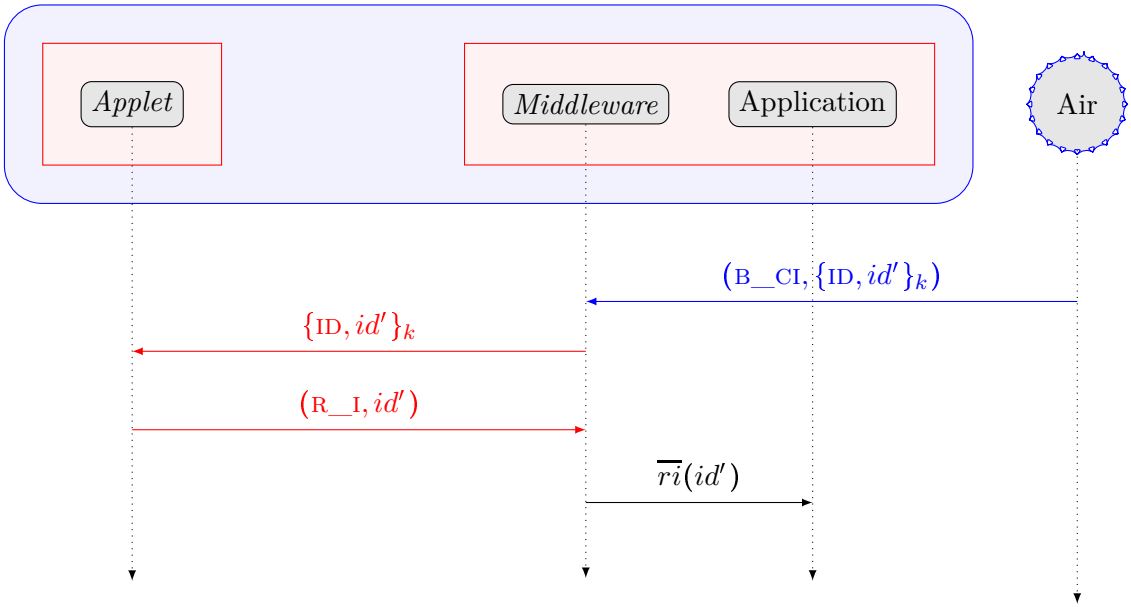


FIG. 4.35: Diagramme de séquence de la primitive `receiveId`.

en fonction du service qu'elle rend. La primitive `receiveId` est définie formellement ci-dessous figure 4.36.

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id}] \mid\mid [(\breve{M}_{ri} \mid P_{ri})\backslash_{ri}]]\!]$
Middleware	$\breve{M}_{ri}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{ri} \mid \breve{M}''_{ri}$
avec	$\breve{M}'_{ri}$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{B\_CI}, x).\ddot{a}(x).\breve{M}'_{ri}$
		$+$	$\mathring{a}\ (\text{OTHERS}).\breve{M}'_{ri}$
et	$\breve{M}''_{ri}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{R\_I}, x).\overline{ri}(x).\breve{M}''_{ri}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{ri}$
Applet	$\breve{A}_{id}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\{\text{ID}, x\}_k).\ddot{a}(\text{R\_I}, x).\breve{A}_{id}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id}$
Application	P_{ri}	$\stackrel{\text{def}}{=}$	$ri(x).0$

FIG. 4.36: Modélisation de la primitive `receiveId`.

Ajout de la primitive `receiveId` au *middleware* MiMAN et à l'*applet* AiMAN. Nous intégrons maintenant (cf. figure 4.37) la primitive `receiveId` au *middleware* MiMAN et à l'*applet* AiMAN existant.

4.2.3 Primitives évoluées sans diffusion OTA de l'identifiant

Nous présentons maintenant des primitives qui ne diffusent pas l'identifiant des nœuds. Dans la suite du document IDC (Identifier and Counter), G_CIC (Get Ciphered Identifier and Counter), P_CIC (Provide Ciphered Identifier and Counter), B_CIC (Broadcast Ciphered Identifier and Counter) et R_IC (*Receive Identifier and Counter*) sont des *flags* au même titre que APP page 109, i.e. qui permettent de différencier les types de messages.

A partir de ce point, nous ajoutons dans l'*applet* qui se trouve dans le *Secure Element* un compteur que nous appelons horloge logique. Le *Secure Element* n'a pas accès à une horloge réelle et nous simulons donc le temps qui avance grâce à ce compteur.

Cette horloge logique comme définie par Lamport [Lam78] permet d'ordonner partiellement les actions qui se produisent dans le système distribué qu'est le réseau mobile ad hoc malgré l'absence d'horloge globale et de synchronisation entre les nœuds. Elle est associée aux primitives `sendIdCounter` et `receiveIdCounter` et elle est incrémentée automatiquement à chaque appel de l'une de ces primitives.

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{\mathbb{A}}_{id}] \,\|\|\, [(\breve{\mathbb{M}} \mid P)\setminus_{s,r,si,ri}]]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$R\breve{I}M \mid A\breve{I}M \mid \Phi_{\dot{a},\overline{a}}(Ap\breve{I}M)$	
avec	$R\breve{I}M$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{APP}, x).\overline{r}(x).R\breve{I}M$	`receive`
		$+$	$\mathring{a}\ (\text{B\_\_CI}, x).\ddot{a}(x).R\breve{I}M$	`receiveId`
		$+$	$\mathring{a}\ (\text{OTHERS}).R\breve{I}M$	
et	$A\breve{I}M$	$\stackrel{\text{def}}{=}$	$s(x).\ \mathring{a}\mathring{}\ (\text{APP}, x).A\breve{I}M$	`send`
		$+$	$si.\ddot{a}(\text{G\_\_CI}).A\breve{I}M$	`sendId`
et	$Ap\breve{I}M$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CI}, x).\ \mathring{a}\mathring{}\ (\text{B\_\_CI}, x).Ap\breve{I}M$	`sendId`
		$+$	$\dot{a}(\text{R\_\_I}, x).\overline{ri}(x).Ap\breve{I}M$	`receiveId`
		$+$	$\dot{a}(\text{OTHERS}).Ap\breve{I}M$	
Applet	$\breve{\mathbb{A}}_{id}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_CI}).\ddot{a}(\text{P\_\_CI}, \{\text{ID}, id\}_k).\breve{\mathbb{A}}_{id}$	`sendId`
		$+$	$\dot{a}(\{\text{ID}, x\}_k).\ddot{a}(\text{R\_\_I}, x).\breve{\mathbb{A}}_{id}$	`receiveId`
		$+$	$\dot{a}(\text{OTHERS}).\breve{\mathbb{A}}_{id}$	

FIG. 4.37: Ajout de la primitive `receiveId` au *middleware*.

4.2.3.1 Primitive `sendIdCounter` = $\overline{\dot{sic}}$

Cette primitive envoie par *broadcast* l'identifiant du nœud et la valeur de l'horloge logique (qui est incrémentée à chaque appel à $\overline{\dot{sic}}$ et ric) chiffrés par la clef symétrique exclusivement stockée dans les *Secure Elements*. Comme expliqué précédemment, l'identifiant, la valeur de l'horloge logique et la clef de chiffrement sont stockés dans le même *Secure Element* et la clef de chiffrement n'est jamais accessible à l'extérieur de ce *Secure Element*. On peut remarquer que tous les messages construits par l'*applet* lors de l'appel à la primitive `sendIdCounter` sont différents car l'*applet* incrémente systématiquement son horloge logique lors de la création du message correspondant à cette primitive. Les messages envoyés *Over The Air* par les nœuds sont donc tous différents et ne représentent donc plus un identifiant pour l'ensemble nœuds, mais uniquement pour les nœuds capables de déchiffrer le message (si on se réfère à la définition 82 page 107).

Le diagramme de séquence de la primitive `sendIdCounter` est présenté figure 4.38.

Modélisation. La primitive `sendIdCounter` est modélisée figure 4.39. L'application P_{sic} effectue l'action $\overline{\dot{sic}}$ qui est capturée au niveau du *middleware* $\breve{M}_{sic}$. Ce dernier demande alors à l'*applet* $\breve{A}_{id,c}$ le message correspondant, à savoir $(\text{P\_\_CIC}, \{\text{IDC}, id, c\}_k)$. Dans ce message, P__CIC et IDC sont des *flag* indiquant le type du message, id est l'identifiant stocké dans le *Secure Element* et c est la valeur de l'horloge logique au moment de la

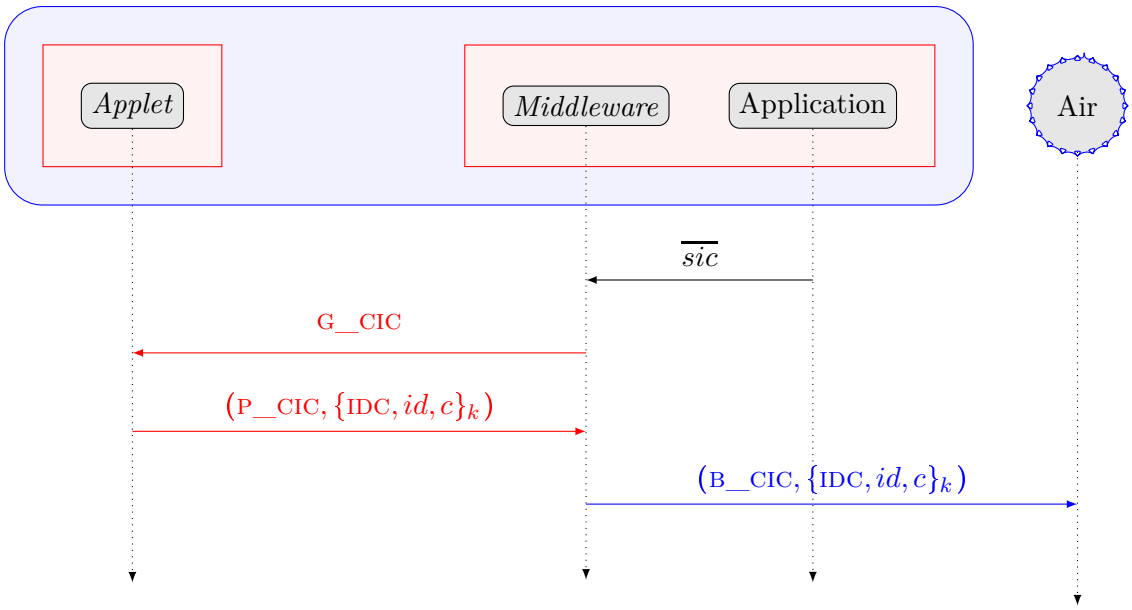


FIG. 4.38: Diagramme de séquence de la primitive `sendIdCounter`.

création du message. A sa réception, le *middleware* le diffuse *Over The Air*.

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id,0}] \parallel [(\breve{M}_{sic} \mid P_{sic}) \setminus_{sic}]]\!]$
Middleware	$\breve{M}_{sic}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{sic} \mid \breve{M}''_{sic}$
avec	$\breve{M}'_{sic}$	$\stackrel{\text{def}}{=}$	$sic.\ddot{a}(\text{G\_\_CIC}).\breve{M}'_{sic}$
et	$\breve{M}''_{sic}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CIC}, x).\ \mathring{a}\ (\text{B\_\_CIC}, x).\breve{M}''_{sic}$
		+	$\dot{a}(\text{OTHERS}).\breve{M}''_{sic}$
Applet	$\breve{A}_{id,c}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_CIC}).\ddot{a}(\text{P\_\_CIC}, \{\text{IDC}, id, c+1\}_k).\breve{A}_{id,c+1}$
		+	$\dot{a}(\text{OTHERS}).\breve{A}_{id,c}$
Application	P_{sic}	$\stackrel{\text{def}}{=}$	$\overline{sic}.0$

FIG. 4.39: Modélisation de la primitive `sendIdCounter`.

Ajout de la primitive `sendIdCounter` au *middleware* MiMAN et à l'*applet* AiMAN. Nous intégrons maintenant (cf. figure 4.40) la primitive `sendIdCounter` au *middleware* MiMAN et à l'*applet* AiMAN existant.

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{\mathbb{A}}_{id,0}] \,\|\|\, [(\breve{\mathbb{M}} \mid P)\setminus_{s,r,si,ri,sic}]]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$R\breve{I}M \mid A\breve{I}M \mid \Phi_{\dot{a},\overline{\ddot{a}}}(Ap\breve{I}M)$	
avec	$R\breve{I}M$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{APP}, x).\overline{r}(x).R\breve{I}M$	`receive`
		$+$	$\mathring{a}\ (\text{B\_\_CI}, x).\ddot{a}(x).R\breve{I}M$	`receiveId`
		$+$	$\mathring{a}\ (\text{OTHERS}).R\breve{I}M$	
et	$A\breve{I}M$	$\stackrel{\text{def}}{=}$	$s(x).\ \mathring{a}^{\circ}\ (\text{APP}, x).A\breve{I}M$	`send`
		$+$	$si.\ddot{a}(\text{G\_\_CI}).A\breve{I}M$	`sendId`
		$+$	$sic.\ddot{a}(\text{G\_\_CIC}).A\breve{I}M$	`sendIdCounter`
et	$Ap\breve{I}M$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CI}, x).\ \mathring{a}^{\circ}\ (\text{B\_\_CI}, x).Ap\breve{I}M$	`sendId`
		$+$	$\dot{a}(\text{R\_\_I}, x).\overline{ri}(x).Ap\breve{I}M$	`receiveId`
		$+$	$\dot{a}(\text{P\_\_CIC}, x).\ \mathring{a}^{\circ}\ (\text{B\_\_CIC}, x).Ap\breve{I}M$	`sendIdCounter`
		$+$	$\dot{a}(\text{OTHERS}).Ap\breve{I}M$	
Applet	$\breve{\mathbb{A}}_{id,c}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_CI}).\ddot{a}(\text{P\_\_CI}, \{\text{ID}, id\}_k).\breve{\mathbb{A}}_{id,c}$	`sendId`
		$+$	$\dot{a}(\{\text{ID}, x\}_k).\ddot{a}(\text{R\_\_I}, x).\breve{\mathbb{A}}_{id,c}$	`receiveId`
		$+$	$\dot{a}(\text{G\_\_CIC}).\ddot{a}(\text{P\_\_CIC}, \{\text{IDC}, id, c+1\}_k).\breve{\mathbb{A}}_{id,c+1}$	`sendIdCounter`
		$+$	$\dot{a}(\text{OTHERS}).\breve{\mathbb{A}}_{id,c}$	

FIG. 4.40: Ajout de la primitive `sendIdCounter` au *middleware*.

4.2.3.2 Primitive `receiveIdCounter` = *ric*

Cette primitive appelle une fonction de niveau applicatif par un mécanisme de *callback* depuis le *middleware* lorsque celui-ci a reçu un message contenant un identifiant et une horloge logique, message qui a été généré par un *Secure Element*.

Cette primitive (cf. figure 4.41) permet simplement de conclure (i) qu'il existe dans le réseau un nœud dont l'identifiant est id' et (ii) qu'il a existé un chemin entre ce nœud et le nœud récepteur du message. Comme expliqué précédemment, l'horloge logique est stockée et incrémentée dans le *Secure Element*. Elle est associée aux messages de type IDC et est incrémentée à chaque appel à $\overline{sic}$ ou *ric*. Elle permet ainsi d'ordonner localement les interactions entre les nœuds, comme décrit dans [Lam78].

Le diagramme de séquence de la primitive `receiveIdCounter` est présenté figure 4.41.

Modélisation. La primitive `receiveIdCounter` est modélisée figure 4.42. Lors de la réception d'un message *Over The Air* de la forme $(\text{B\_\_CIC}, x)$ (avec x variable) par la

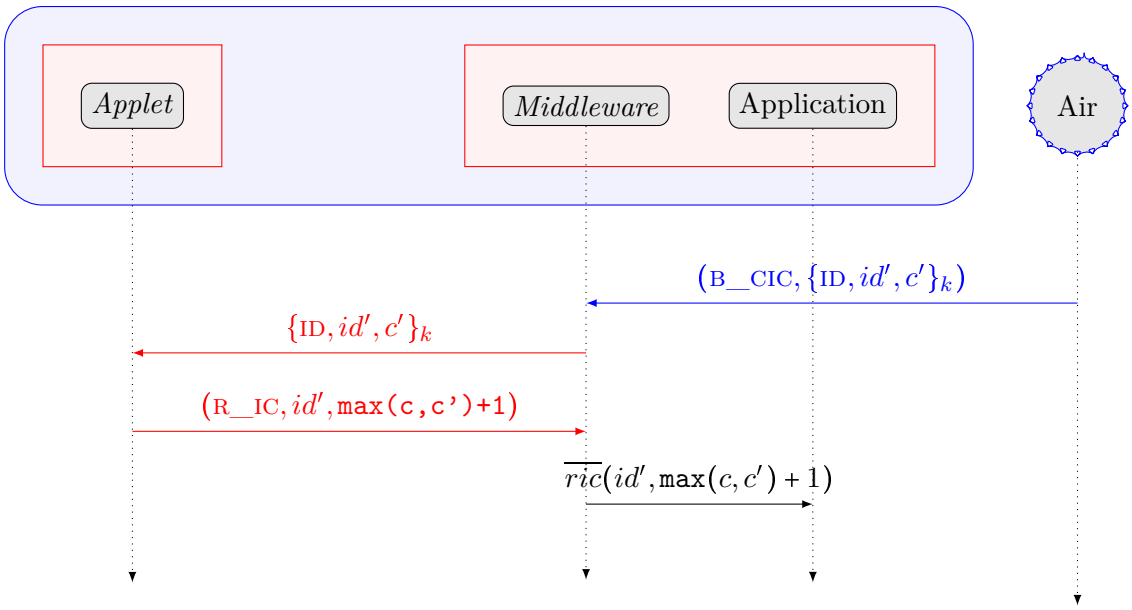


FIG. 4.41: Diagramme de séquence de la primitive `receiveIdCounter`.

partie $\breve{M}'_{ric}$ du *middleware* $\breve{M}_{ric}$, ce dernier fait suivre x à l'*applet* $\breve{A}_{id,c}$ qui le déchiffre et retourne au *middleware* $\breve{M}_{ric}$ l'identifiant stocké dans *Secure Element* qui a construit ce message ainsi que le maximum entre la valeur de l'horloge logique du *Secure Element* local (qui a déchiffré le message) et la valeur de l'horloge logique du *Secure Element* qui a construit le message au moment de sa construction. Ce couple (identifiant, horloge logique) est remonté par le *middleware* à l'application P_{ric} par une action de type $\overline{ric}$. Cette action est capturée par l'application qui peut alors effectuer le traitement approprié en fonction du service qu'elle rend.

Ajout de la primitive `receiveIdCounter` au *middleware* MiMAN et à l'*applet* AiMAN. Nous intégrons maintenant (cf. figure 4.43) la primitive `receiveIdCounter` au *middleware* MiMAN et à l'*applet* AiMAN existant.

Les primitives `sendIdCounter` et `receiveIdCounter` sont utilisées par l'application de collecte de traces de voisinage que nous présentons en section 4.4 page 138.

4.2.4 Primitives évoluées sans diffusion de l'identifiant au niveau *middleware*

Nous présentons maintenant un ensemble de primitives évoluées qui ne diffusent pas l'identifiant des nœuds. Ces primitives sont essentielles pour les applications de comptage sécurisées que nous présentons en section 4.3 page 131.

Dans la suite du document, PI (*ping*), PO (*pong*), SEEN, SEEING, LOST et NR (*next round*) sont des *flags* (au même titre que APP) et x, y et z désignent des variables.

Nous n'intégrons plus au fur et à mesure les primitives que nous définissons au *middleware*

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id,0}] \,\|\|\, [(\breve{M} \mid P)\backslash_{ric}]]\!]$
Middleware	$\breve{M}_{ric}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{ric} \mid \breve{M}''_{ric}$
avec	$\breve{M}'_{ric}$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{B\_\_CIC}, x).\ddot{a}(x).\breve{M}'_{ric}$
		$+$	$\mathring{a}\ (\text{OTHERS}).\breve{M}'_{ric}$
et	$\breve{M}''_{ric}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{R\_\_IC}, x, y).\overline{ric}(x, y).\breve{M}''_{ric}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{ric}$
Applet	$\breve{A}_{id,c}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\{\text{IDC}, x, y\}_k).\ddot{a}(\text{R\_\_IC}, x, \texttt{max}(y, c) + 1).\breve{A}_{id,\texttt{max}(y,c)+1}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id,c}$
Application	P_{ric}	$\stackrel{\text{def}}{=}$	$ric(x, y).0$

FIG. 4.42: Modélisation de la primitive `receiveIdCounter`.

car leurs modélisations, bien plus importantes en taille, rendraient la lecture plus difficile. Cette intégration au *middleware* est réalisée figures 4.54 et 4.55 en section 4.2.5 page 130.

Dans un réseau mobile ad hoc, la dynamique des nœuds induit un voisinage en évolution permanente. Nous présentons ici un mécanisme à base de ping/pong qui permet aux nœuds du réseau de sonder leur voisinage. La primitive `ping` provoque l'envoi d'un *beacon* (un ping) qui appelle une réponse (un pong) de la part des nœuds récepteurs de manière somme toute classique. Lors de l'arrivée d'un message de type PI (ping) sur un nœud, un message de type PO (pong) est retourné. La fonction `seen` de niveau applicatif, invoquée par le *middleware* grâce à un mécanisme de *callback*, signifie que le nœud a reçu un message un message de type PI, le premier avec cet identifiant. La primitive `seeing` est elle aussi une fonction de niveau applicatif mais elle est appelée par *middleware* lors de la réception d'un message de type PO.

Les primitives précédentes, `send` / `receive`, `sendId` / `receiveId` et `sendIdCounter` / `receiveIdCounter` fonctionnent par paire. Les cinq primitives que nous présentons dans cette section sont quant à elles liées les unes aux autres.

4.2.4.1 Primitive `ping` = *ping*

La primitive `ping` permet au nœud qui l'utilise de s'annoncer auprès de ses voisins de manière totalement sécurisée, c'est à dire en préservant son anonymat sur le réseau. De plus, comme son nom l'indique, elle provoque l'émission d'un message de type pong par les nœuds récepteurs. Comme nous le verrons plus loin, associée à `seen` elle permet à un nœud de découvrir l'existence d'autres nœuds et associée à `seeing` elle permet à un nœud d'évaluer son nombre de voisins.

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{\mathbb{A}}_{id,0}] \,\|\|\, [(\breve{\mathbb{M}} \mid P)\backslash_{s,r,si,ri,sic,ric}]]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$R\breve{I}M \mid A\breve{I}M \mid \Phi_{\dot{a},\overline{\alpha}}(Ap\breve{I}M)$	
avec	$R\breve{I}M$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{APP}, x).\overline{r}(x).R\breve{I}M$	`receive`
		+	$\mathring{a}\ (\text{B\_\_CI}, x).\ddot{a}(x).R\breve{I}M$	`receiveId`
		+	$\mathring{a}\ (\text{B\_\_CIC}, x).\ddot{a}(x).R\breve{I}M$	`receiveIdCounter`
		+	$\mathring{a}\ (\text{OTHERS}).R\breve{I}M$	
et	$A\breve{I}M$	$\stackrel{\text{def}}{=}$	$s(x).\ \mathring{a}\mathring{}\ (\text{APP}, x).A\breve{I}M$	`send`
		+	$si.\ddot{a}(\text{G\_\_CI}).A\breve{I}M$	`sendId`
		+	$sic.\ddot{a}(\text{G\_\_CIC}).A\breve{I}M$	`sendIdCounter`
et	$Ap\breve{I}M$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CI}, y).\ \mathring{a}\mathring{}\ (\text{B\_\_CI}, y).Ap\breve{I}M$	`sendId`
		+	$\dot{a}(\text{R\_\_I}, x).\overline{ri}(x).Ap\breve{I}M$	`receiveId`
		+	$\dot{a}(\text{P\_\_CIC}, x).\ \text{B\_\_}\mathring{\text{C}}\mathring{\text{I}}\text{C}, a\ (x).Ap\breve{I}M$	`sendIdCounter`
		+	$\dot{a}(\text{R\_\_IC}, x, y).\overline{ric}(x, y).Ap\breve{I}M$	`receiveIdCounter`
		+	$\dot{a}(\text{OTHERS}).Ap\breve{I}M$	
Applet	$\breve{\mathbb{A}}_{id,c}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_CI}).\ddot{a}(\text{P\_\_CI}, \{\text{ID}, id\}_k).\breve{\mathbb{A}}_{id,c}$	`sendId`
		+	$\dot{a}(\{\text{ID}, x\}_k).\ddot{a}(\text{R\_\_I}, x).\breve{\mathbb{A}}_{id,c}$	`receiveId`
		+	$\dot{a}(\text{G\_\_CIC}).\ddot{a}(\text{P\_\_CIC}, \{\text{IDC}, i, c+1\}_k).\breve{\mathbb{A}}_{id,c+1}$	`sendIdCounter`
		+	$\dot{a}(\{\text{IDC}, x, y\}_k).\ddot{a}(\text{R\_\_IC}, x, y).\breve{\mathbb{A}}_{id,c}$	`receiveIdCounter`
		+	$\dot{a}(\text{OTHERS}).\breve{\mathbb{A}}_{id,c})$	

FIG. 4.43: Ajout de la primitive `receiveIdCounter` au *middleware*.

Modélisation. La modélisation de la primitive `ping` est présentée figure 4.45. Nous supposons que l'*applet* peut accéder à une fonction `rand` qui retourne un nombre aléatoire. L'*applet* de la carte à puce ne disposant pas d'horloge, elle est incapable d'évaluer le temps. Nous utilisons donc un mécanisme à base de *round* pour le simuler. L'*applet* maintient un nombre entier r qui correspond au numéro de *round* courant, ce nombre étant totalement indépendant de l'horloge logique précédemment définie.

L'application P_{ping} effectue l'action $\overline{ping}$ qui est capturée au niveau du *middleware* $\breve{M}_{ping}$. Ce dernier demande alors à l'*applet* $\breve{A}_{id,r}$ le message de *ping* correspondant, à savoir $\{\text{PI}, id, r, rand\}_k$. Dans ce message, PI est un *flag* indiquant son type, id est l'identifiant du nœud stocké dans le *Secure Element*, r est le numéro de *round* de l'*applet* lors de la création du message et $rand$ est un nombre aléatoire. A sa réception, le *middleware* le diffuse *Over The Air*.

L'identifiant *id* et le flag PI sont constants, et le numéro de *round* n'est pas forcément différent lors de la création de deux messages de ping successifs (la création d'un tel message ne provoque pas de changement de *round* au niveau de l'*applet*, seul `nextRound` permet changer de *round*). Nous ajoutons donc un nombre aléatoire `rand` aux messages afin de garantir qu'ils sont tous différents les uns des autres, ce qui apporte une confidentialité partielle. En effet, les nœuds qui ne disposent pas de l'*applet* ne peuvent pas déchiffrer les messages, et les informations qu'ils peuvent alors recueillir ne donnent pas d'information sur les nœuds qui ont produits les messages interceptés (cf. définition 85 page 107).

Le diagramme de séquence de la primitive `ping` est présenté en figure 4.44.

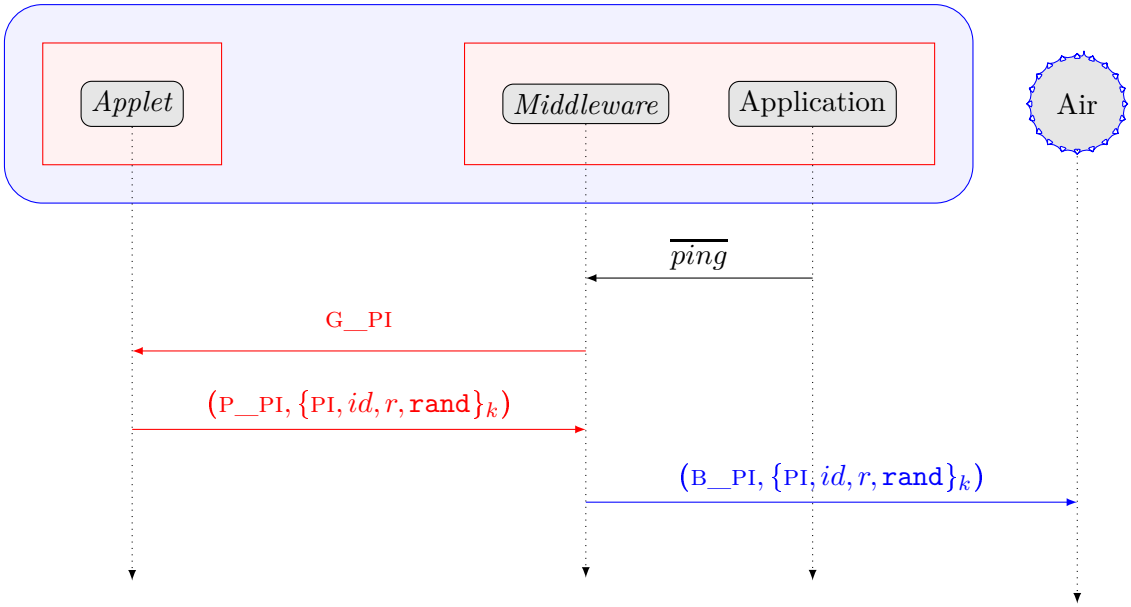


FIG. 4.44: Diagramme de séquence de la primitive `ping`.

4.2.4.2 Primitives `seen` = *seen*

Cette primitive, lorsqu'elle est invoquée dans l'application par le *middleware* grâce à un mécanisme de *callback*, signifie que le nœud vient de recevoir un message de type PI provenant d'un nœud duquel il n'en a pas encore reçu. Les informations contenues dans ce message (identifiant et numéro de *round*) et qui concernent le nœud qui l'a construit ne sont pas remontées par l'*applet* au *middleware* et ne sont donc pas transmises à l'application.

Modélisation. La modélisation de la primitive `seen` est présentée figure 4.47. La liste des identifiants des nœuds pour lesquels l'*applet* a déjà reçu un message de type PI est nommée *lpi* (liste ping). Cette liste permet à l'*applet* de déterminer si le nœud duquel on reçoit le message PI a déjà provoqué l'envoi d'un message de type SEEN vers le *middleware*. Lors de la réception d'un message *Over The Air* de type (PI, x) (avec x variable) par le *middleware* $\breve{M}_{seen}$, ce dernier fait suivre x à l'*applet* $\breve{A}_{id}(lpi)$ qui le déchiffre et retourne au *middleware* soit un message $(\text{P\_PO}, y)$ indiquant au *middleware* de faire suivre *Over The*

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id,0}] \parallel [(\breve{M}_{ping} \mid P_{ping}) \setminus_{ping}]]\!]$
Middleware	$\breve{M}_{ping}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{ping} \mid \breve{M}''_{ping}$
	$\breve{M}'_{ping}$	$\stackrel{\text{def}}{=}$	$ping.\ddot{a}(\text{G\_\_PI}).\breve{M}'_{ping}$
	$\breve{M}''_{ping}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_PI}, x).\ \mathring{a}\ (\text{B\_\_PI}, x).\breve{M}''_{ping}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{ping}$
Applet	$\breve{A}_{id,r}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_PI}).\ddot{a}(\text{P\_\_PI}, \{\text{PI}, id, r, \texttt{rand}\}_k).\breve{A}_{id,r}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id,r}$
Application	P_{ping}	$\stackrel{\text{def}}{=}$	$\overline{ping}.0$

FIG. 4.45: Modélisation de la primitive `ping`.

Air le message y contenant la réponse de type *pong* (si le nœud qui a émis le ping était déjà connu de l'*applet*), soit un message (SEEN, y) indiquant au *middleware* d'envoyer *Over The Air* le message y qui contient la réponse de type *pong* puis d'effectuer l'action $\overline{seen}$ (le message ping reçu provient d'un nœud qui était inconnu de l'*applet*). Cette dernière action est capturée par l'application qui peut alors effectuer le traitement approprié en fonction du service qu'elle rend. Comme expliqué précédemment, la primitive `seen` ne remonte aucune information sur le nœud qui a construit le message de *ping*. Il n'existe par ailleurs pas de primitive qui permette d'extraire d'autre information d'un message de type *ping*, garantissant ainsi un anonymat total (cf. définition 86 page 108).

Le diagramme de séquence de la primitive `seen` est présenté en figure 4.46.

4.2.4.3 Primitive `seeing` = *seeing*

Un message de type PI comprend notamment le numéro de *round* de l'*applet* au moment de la création du message. Lors de la réception d'un message de type PI par un nœud, celui-ci répond par un message de type PO dans lequel le numéro de *round* du message de type PI est recopié. Cela permet au nœud qui reçoit le message de type PO de savoir à quel *round* le message de type PO correspond. Un message de type PO est dit valide si le numéro de *round* qu'il contient correspond au numéro du *round* courant de l'*applet*.

Cette fonction de niveau applicatif, invoquée par le *middleware* grâce à un mécanisme de *callback*, signifie que le nœud a reçu un message de type PO (donc en réponse à un message de type PI) de la part d'un nœud qui ne lui avait pas répondu ni dans le *round* courant, ni lors du *round* précédent ; ce nœud était donc supposé perdu (cf. primitives `nextRound` et `lost` sections 4.2.4.4 et 4.2.4.5). L'information supplémentaire que remonte cette primitive est un compteur associé à l'identifiant du nœud distant. Ce compteur est

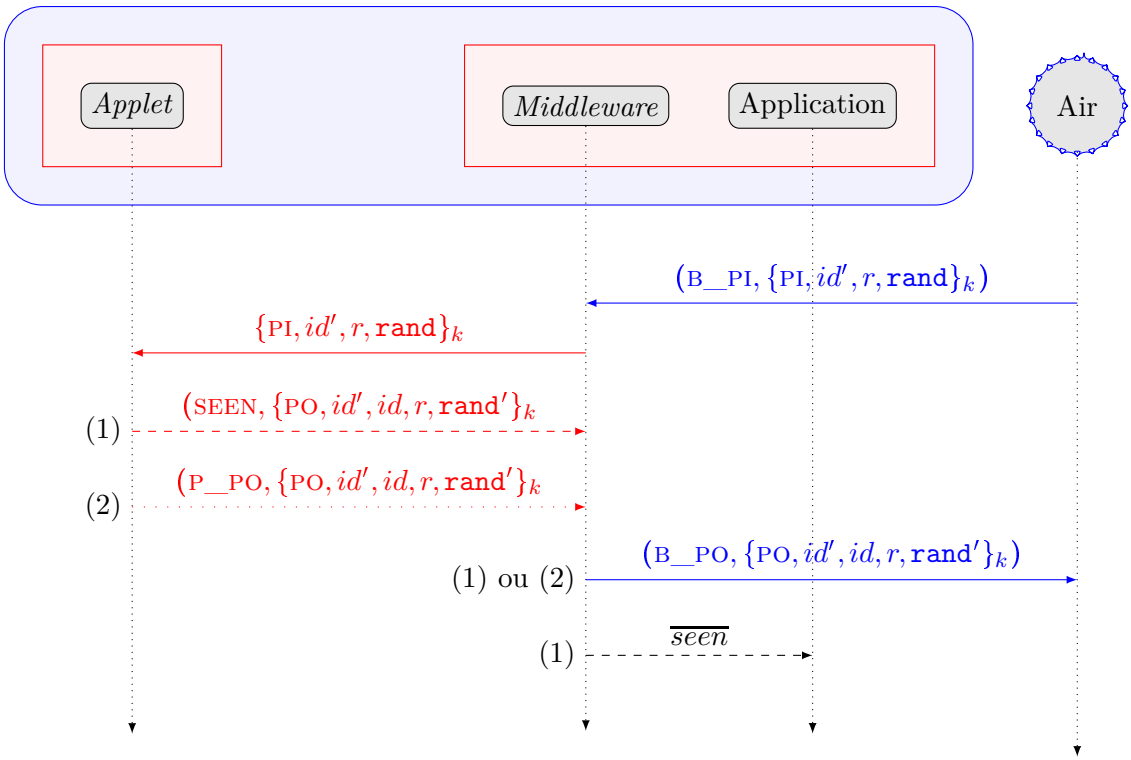


FIG. 4.46: Diagramme de séquence de la primitive `seen`.

incrémenté à chaque réception de message de type PO valide mais au plus une fois par *round*. Il indique donc le nombre de *rounds* pendant lesquels le nœud distant a pu retourner un message de type PO valide (c'est à dire qui possédait le même numéro de *round* que l'*applet*). Nous n'exploitons pour l'instant pas ce compteur au niveau des applications que nous présentons. Nous pensons cependant que celui-ci pourrait apporter des statistiques intéressantes concernant par exemple la fréquence moyenne d'apparition d'un nouveau voisin (quand la valeur remontée vaut 1). Il pourrait aussi permettre d'étudier la fréquence avec laquelle on croise les autres nœuds (montrant par exemple que si un *round* dure une journée, alors 20% des nœuds connus sont croisés plus de 50% des *rounds*).

Le diagramme de séquence de la primitive `seeing` est présenté en figure 4.48.

Modélisation. La primitive `seeing` est modélisée figure 4.49. La liste des identifiants des nœuds ayant transmis par retour un pong avec le numéro de *round* courant est nommée *cr* (*current round*) ; *pr* (*previous round*) est la liste des identifiants des nœuds ayant envoyé un pong valable dans le *round* précédent et *lpo* (liste pong) est la liste des identifiants des nœuds ayant déjà envoyé un pong valable (i.e. qui a provoqué un appel à la primitive `seeing`). Ces données permettent de garder à jour des informations sur les nœuds voisins, i.e. capables de répondre entre chaque changement de *round*. Dans la modélisation présentée figure 4.49, x, y et z désignent des variables.

Lors de la réception d'un message *Over The Air* de type $(\text{B\_PO}, x)$ (avec x variable) par la

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id}(\varnothing)] \,\|\|\, [(\breve{M}_{seen} \mid P_{seen}) \setminus seen]]\!]$
Middleware	$\breve{M}_{seen}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{seen} \mid \breve{M}''_{seen}$
	$\breve{M}'_{seen}$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{B\_\_PI}, x).\ddot{a}(x).\breve{M}'_{seen}$
		$+$	$\mathring{a}\ (\text{OTHERS}).\breve{M}'_{seen}$
	$\breve{M}''_{seen}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_PO}, x).\ \mathring{a}\mathring{a}\ (\text{B\_\_PO}, x).\breve{M}''_{seen}$
		$+$	$\dot{a}(\text{SEEN}, x).\ \mathring{a}\mathring{a}\ (\text{B\_\_PO}, x).\overline{seen}.\breve{M}''_{seen}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{seen}$
Applet	$\breve{A}_{id}(lpi)$	$\stackrel{\text{def}}{=}$	$\dot{a}(\{\text{PI}, x \in lpi, y, z\}_k).$ $\ddot{a}(\text{P\_\_PO}, \{\text{PO}, x, id, y, \texttt{rand}\}_k).\breve{A}_{id}(lpi)$
		$+$	$\dot{a}(\{\text{PI}, x \notin lpi, y, z\}_k).$ $\ddot{a}(\text{SEEN}, \{\text{PO}, x, id, y, \texttt{rand}\}_k).\breve{A}_{id}(lpi.x)$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id}(lpi)$
Application	P_{seen}	$\stackrel{\text{def}}{=}$	$seen.0$

FIG. 4.47: Modélisation de la primitive `seen`.

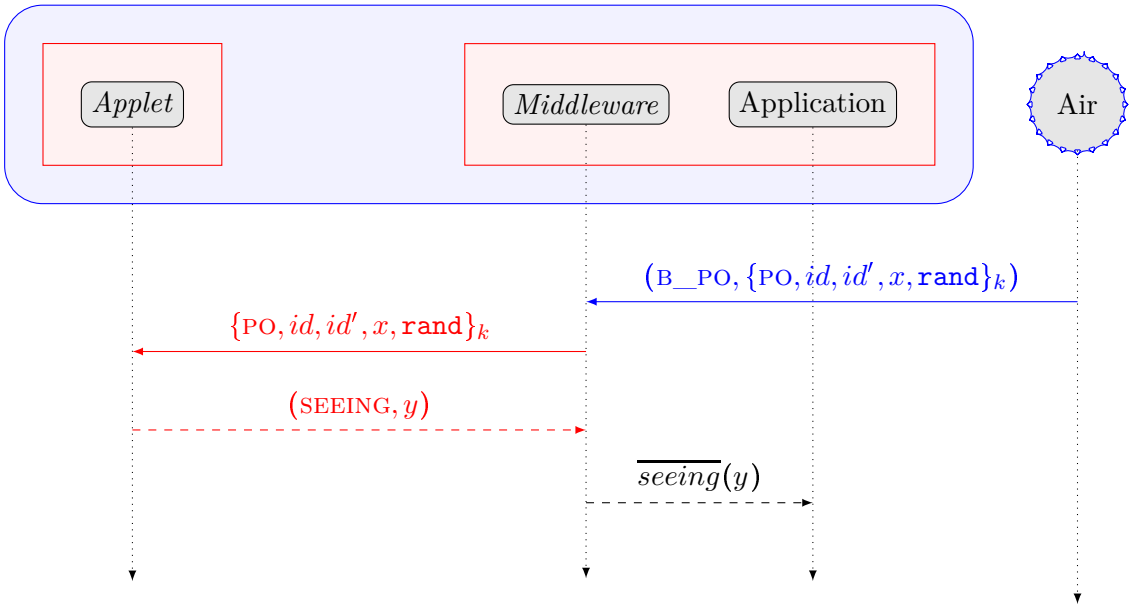


FIG. 4.48: Diagramme de séquence de la primitive `seeing`.

partie $\breve{M}'_{seeing}$ du *middleware* $\breve{M}_{seeing}$, ce dernier fait suivre x à l'*applet* $\breve{A}_{id,r,k}(cr, pr, lpo)$ qui le déchiffre. Si le numéro de *round* inscrit dans le message de type PO ne correspond pas au numéro de *round* courant alors le message est ignoré (action $\dot{a}$(OTHERS)), sinon trois traitements différents sont possibles.

Le premier de ces traitements est effectué si l'*applet* n'a jamais reçu de message de type PO du nœud qui l'a construit. Dans ce cas, un compteur correspondant à l'identifiant contenu dans la réponse est initialisé à 2 (processus K_2), l'applet retourne un message indiquant que la primitive `seeing` doit être appelée et enfin, l'identifiant est ajouté à la liste *lpo*.

Le second de ces traitements concerne le cas où l'identifiant du nœud qui répond appartient à la liste *pr* mais n'appartient pas à liste *cr*. Dans le cas, le compteur est incrémenté, et aucun message n'est retourné par l'*applet* au *middleware*.

Le troisième et dernier traitement possible concerne le cas où le nœud est connu de l'*applet* (car il appartient à la liste *lpo*) mais n'appartient pas aux listes *pr* ou *cr*. Dans ce cas, le compteur correspondant à ce nœud est incrémenté puis remonté au *middleware* en lui indiquant de lancer la primitive `seeing`.

L'action $\overline{seeing}$ est capturée par l'application qui peut alors effectuer le traitement approprié en fonction du service qu'elle rend. La primitive `seeing` ne remonte aucune information sur le nœud qui a construit le message de type PO. L'unique information remontée concerne le nombre de fois qu'un nœud a été capable de répondre "suffisamment rapidement" à une requête de type PI. Il n'existe par ailleurs pas de primitive qui permette d'extraire d'autre information d'un message de type PO ailleurs que dans l'*applet*, garantissant ainsi un anonymat total des nœuds du réseau tel que défini dans la définition 86 page 108.

4.2.4.4 Primitive `nextRound` = $\overline{nr}$

Cette primitive, appelée par l'application, provoque un changement de *round* de l'*applet* (le numéro de *round* courant est incrémenté). Un *round* est la période de temps écoulée entre deux appels à `nextRound` consécutifs. Lors d'un changement de *round*, la liste des identifiants des nœuds ayant retourné un pong valable dans le *round* courant (*cr*) devient la liste des nœuds ayant retourné un pong valable dans le *round* précédent (*pr*). Lors de cette opération, l'*applet* calcule le nombre de nœuds qui s'étaient annoncés dans le *round* précédent et qui ne se sont pas annoncés dans le *round* courant. Cette valeur peut-être récupérée par l'application grâce à la primitive `lost`.

Le diagramme de séquence de la primitive `nextRound` est présenté en figure 4.50.

Modélisation. La primitive `nextRound` est modélisée figure 4.51. Dans celle-ci, r représente la numéro du *round* courant et n le nombre de nœuds perdus. L'action $\overline{nr}$ effectuée par l'application P_{nr} est capturée par le *middleware* $\breve{M}_{nr}$ qui envoie une requête de changement de *round* à l'*applet* $\breve{A}_{r,n}(cr, pr)$. L'*applet* effectue une série d'actions afin de calculer le nombre de nœuds considérés perdus pendant le *round* courant (c'est le rôle du processus $\breve{K}_{r,n}(cr, pr)$).

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id,0,0}(\varnothing,\varnothing,\varnothing)] \parallel [(\breve{M}_{seeing} \mid P_{seeing})\setminus_{seeing}]]\!]$
Middleware	$\breve{M}_{seeing}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{seeing} \mid \breve{M}''_{seeing}$
	$\breve{M}'_{seeing}$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{B\_\_PO}, x).\ddot{a}(x).\breve{M}'_{seeing}$
		$+$	$\mathring{a}\ (\text{OTHERS}).\breve{M}'_{seeing}$
	$\breve{M}''_{seeing}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{SEEING}, x).\overline{seeing}(x).\breve{M}''_{seeing}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{seeing}$
Applet	$\breve{A}_{id,r,k}(cr, pr, lpo)$	$\stackrel{\text{def}}{=}$	$\dot{a}(\{\text{PO}, id, x \notin lpo, r, y\}_k).$
			$(K_2 \mid \ddot{a}(\text{SEEING}, 1).\breve{A}_{id,r,k}(cr.x, pr, lpo.x))\setminus_a$
		$+$	$\dot{a}(\{\text{PO}, id, x \notin cr \wedge x \in pr, r, y\}_k).$
			$\overline{a}.a(z).\breve{A}_{id,r,k}(cr.x, pr, lpo)$
		$+$	$\dot{a}(\{\text{PO}, id, x \notin cr \cup pr \wedge x \in lpo, r, y\}_k).$
			$\overline{a}.a(z).\ddot{a}(\text{SEEING}, z).\breve{A}_{id,r,k}(cr.x, pr, lpo)$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id,r,k}(cr, pr, lpo)$
	K_k	$\stackrel{\text{def}}{=}$	$a.\overline{a}(k).K_{k+1}$
Application	P_{seeing}	$\stackrel{\text{def}}{=}$	$seeing(x).P_{seeing}.0$

FIG. 4.49: Modélisation de la primitive `seeing`.

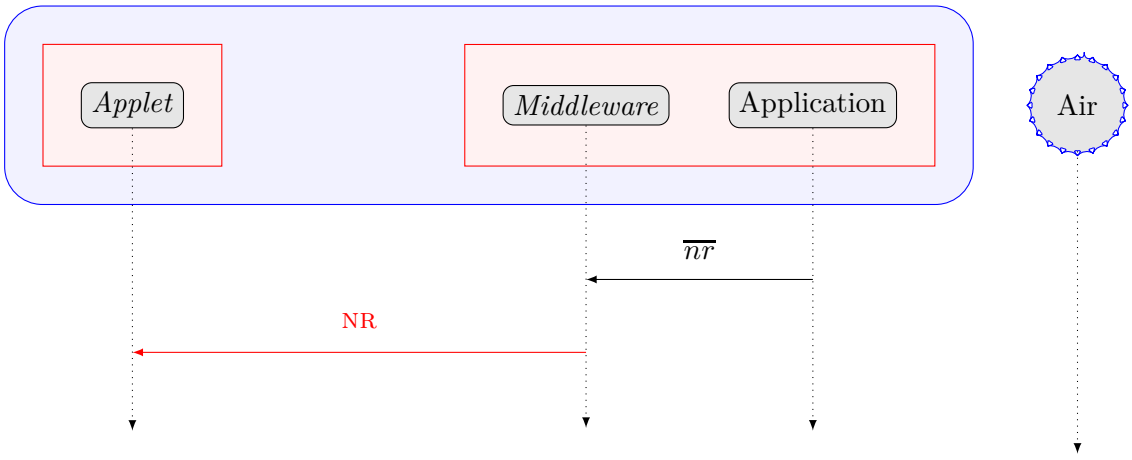


FIG. 4.50: Diagramme de séquence de la primitive `nextRound`.

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{0,0}(\varnothing,\varnothing)] \,\|\|\, [(\breve{M}_{nr} \mid P_{nr})\setminus_{nr}]]\!]$
Middleware	$\breve{M}_{nr}$	$\stackrel{\text{def}}{=}$	$nr.\ddot{a}(\text{NR}).\breve{M}_{nr}$
Applet	$\breve{A}_{r,n}(cr,pr)$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{NR}).\breve{K}_{r,n}(cr,pr)$ /*pr et cr sont des listes */
	$\breve{K}_{r,n}(cr,x.pr)$	$\stackrel{\text{def}}{=}$	$\begin{cases} \left.\begin{array}{l}\text{if } x \notin cr \text{ then} \\ \quad \breve{K}_{r,n+1}(cr,pr) \\ \text{else} \\ \quad \breve{K}_{r,n}(cr,pr)\end{array}\right\} \text{si } x.pr \neq \varnothing \\ \breve{A}_{r+1,n}(\varnothing,cr) \quad \text{si } x.pr = \varnothing \end{cases}$
Application	P_{nr}	$\stackrel{\text{def}}{=}$	$\overline{nr}.0$

FIG. 4.51: Modélisation de la primitive `nextRound`.

4.2.4.5 Primitive `lost` = *lost*

Cette primitive, appelée par l'application, permet de récupérer la valeur du compteur correspondant au nombre de nœuds considérés perdus par l'*applet*. Un nœud est considéré perdu lorsqu'aucun message de type pong provenant de ce nœud n'a été reçu entre deux appels consécutifs à la primitive `nextRound`.

Le diagramme de séquence de la primitive `lost` est présenté en figure 4.52.

Modélisation. La primitive `lost` est modélisée figure 4.53. L'action $\overline{lost}$ effectuée par l'application P_{lost} est capturée par le *middleware* $\breve{M}_{lost}$ qui envoie une requête pour récupérer la valeur du compteur du nombre de nœuds perdus n qui se trouve dans l'*applet* $\breve{A}_n$ et qui a été calculée lors du dernier appel à la primitive `nextRound`. L'*applet* retourne cette valeur au *middleware* puis la réinitialise à 0. Le *middleware* la fait suivre à l'application par l'intermédiaire d'une action $\overline{lost}$.

4.2.5 *Middleware* et *applet* complets

Nous présentons la modélisation complète du *middleware* MiMAN ainsi que de l'*applet* AiMAN figures 4.54 et 4.55.

Dans la suite, nous ne définissons plus $\breve{\mathbb{M}}$ et $\breve{\mathbb{A}}$ mais uniquement P.

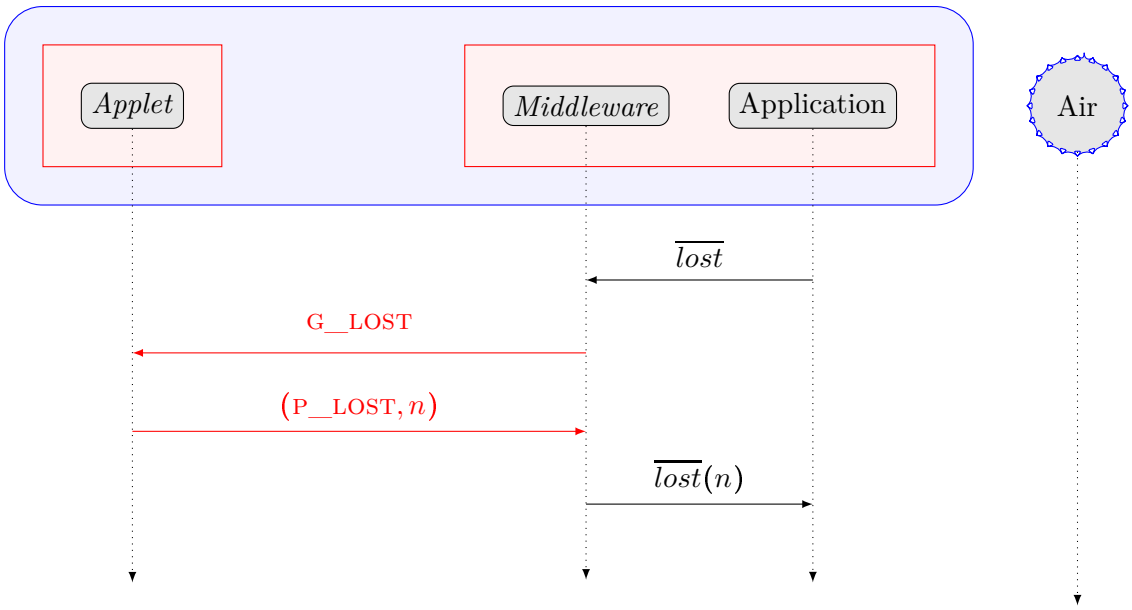


Fig. 4.52: Diagramme de séquence de la primitive `lost`.

Nœud	N	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_0] \parallel [(\breve{M}_{lost} \mid P_{lost})\backslash_{lost}]]\!]$
Middleware	$\breve{M}_{lost}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{lost} \mid \breve{M}''_{lost}$
	$\breve{M}'_{lost}$	$\stackrel{\text{def}}{=}$	$lost.\ddot{a}(\text{G\_\_LOST}).\breve{M}'_{lost}$
	$\breve{M}''_{lost}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_LOST}, x).\overline{lost}(x).\breve{M}''_{lost}$
Applet	$\breve{A}_n$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{G\_\_LOST}).\ddot{a}(\text{P\_\_LOST}, n).\breve{A}_0$
Application	P_{lost}	$\stackrel{\text{def}}{=}$	$\overline{lost}.lost(x).0$

Fig. 4.53: Modélisation de la primitive `lost`.

4.3 Application 1 : le comptage, ses garanties et la sécurité associée

L'objectif d'un algorithme de comptage est de permettre à un nœud de compter l'ensemble des membres du réseau auquel il appartient et qui répondent à certains critères. Dans un iMANet, la mobilité et donc la possibilité de constitution d'îlots influe directement sur la réussite d'un algorithme de comptage quel qu'il soit. Dans un tel contexte où chaque message peut être perdu, l'objectif de nos algorithmes de comptage est donc de fournir, non pas le nombre exact de nœuds qui composent le réseau (résultat impossible à garantir), mais une borne inférieure de ce nombre.

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{\mathbb{A}}_{id,0,0,0}(\varnothing,\varnothing,\varnothing,\varnothing)] \parallel [(\breve{\mathbb{M}} \mid P)\setminus_{s,r,si,ri,sic,ric,ping,seen,seeing,nr,lost}]\!]$	
Middleware	$\breve{\mathbb{M}}$	$\stackrel{\text{def}}{=}$	$R\breve{I}M \mid A\breve{I}M \mid \Phi_{\dot{a},\overline{\alpha}}(Ap\breve{I}M)$	
avec	$R\breve{I}M$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{APP}, x).\overline{r}(x).R\breve{I}M$	`receive`
		+	$\mathring{a}\ (\text{B\_\_CI}, x).\ddot{a}(x).R\breve{I}M$	`receiveId`
		+	$\mathring{a}\ (\text{B\_\_CIC}, x).\ddot{a}(x).R\breve{I}M$	`receiveIdCounter`
		+	$\mathring{a}\ (\text{B\_\_PI}, x).\ddot{a}(x).R\breve{I}M$	`seen`
		+	$\mathring{a}\ (\text{B\_\_PO}, x).\ddot{a}(x).R\breve{I}M$	`seeing`
		+	$\mathring{a}\ (\text{OTHERS}).R\breve{I}M$	
et	$A\breve{I}M$	$\stackrel{\text{def}}{=}$	$s(x).\ \mathring{a}\mathring{}\ (\text{APP}, x).A\breve{I}M$	`send`
		+	$si.\ddot{a}(\text{G\_\_CI}).A\breve{I}M$	`sendId`
		+	$sic.\ddot{a}(\text{G\_\_CIC}).A\breve{I}M$	`sendIdCounter`
		+	$ping.\ddot{a}(\text{G\_\_PI}).A\breve{I}M$	`ping`
		+	$nr.\ddot{a}(\text{NR}).A\breve{I}M$	`nextRound`
		+	$lost.\ddot{a}(\text{G\_\_LOST}).A\breve{I}M$	`lost`
et	$Ap\breve{I}M$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_CI}, x).\ \mathring{a}\mathring{}\ (\text{B\_\_CI}, x).Ap\breve{I}M$	`sendId`
		+	$\dot{a}(\text{R\_\_I}, x).\overline{ri}(x).Ap\breve{I}M$	`receiveId`
		+	$\dot{a}(\text{P\_\_CIC}, x).\ \mathring{a}\mathring{}\ (\text{B\_\_CIC}, x).Ap\breve{I}M$	`sendIdCounter`
		+	$\dot{a}(\text{R\_\_IC}, x, y).\overline{ric}(x, y).Ap\breve{I}M$	`receiveIdCounter`
		+	$\dot{a}(\text{P\_\_PI}, x).\ \mathring{a}\mathring{}\ (\text{B\_\_PI}, x).Ap\breve{I}M$	`ping`
		+	$\dot{a}(\text{P\_\_PO}, x).\ \mathring{a}\mathring{}\ (\text{B\_\_PO}, x).Ap\breve{I}M$	`seen`
		+	$\dot{a}(\text{SEEN}, x).\ \mathring{a}\mathring{}\ (\text{B\_\_PO}, x).\overline{seen}.Ap\breve{I}M$	`seen`
		+	$\dot{a}(\text{SEEING}, x).\overline{seeing}(x).Ap\breve{I}M$	`seeing`
		+	$\dot{a}(\text{P\_\_LOST}, x).\overline{lost}(x).Ap\breve{I}M$	`lost`
		+	$\dot{a}(\text{OTHERS}).Ap\breve{I}M$	

FIG. 4.54: Modélisation complète du *middleware* MiMAN.

Nous présentons ici cinq algorithmes de comptage qui possèdent des garanties de résultat et de sécurité (relative à l'anonymat) différentes. Nous résumons les caractéristiques de chacun de ces algorithmes dans le tableau 4.10. Le premier algorithme ne fournit aucune garantie en ce qui concerne le nombre de nœuds comptés et les nœuds ne sont pas anonymes. Cet algorithme est progressivement amélioré afin d'aboutir à un algorithme de comptage dans lequel une borne inférieure du nombre de nœuds du réseau est garantie, algorithme dans lequel les nœuds sont totalement anonymes et qui permet en outre d'estimer le nombre de nœuds voisins.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{Applet}\quad \breve{\mathbb{A}}_{id,c,r,n}(lpi,cr,pr,lpo) \stackrel{\text{def}}{=}} \\
 & \dot{a}(\text{G\_\_CI}).\ddot{a}(\text{P\_\_CI},\{\text{ID},id\}_k).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\{\text{ID},x\}_k).\ddot{a}(\text{R\_\_I},x).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\text{G\_\_CIC}).\ddot{a}(\text{P\_\_CIC},\{\text{IDC},i,c+1\}_k).\breve{\mathbb{A}}_{id,c+1,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\{\text{IDC},x,y\}_k).\ddot{a}(\text{R\_\_IC},x,\texttt{max}(y,c)+1).\breve{\mathbb{A}}_{id,\texttt{max}(y,c)+1,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\text{G\_\_PI}).\ddot{a}(\text{P\_\_PI},\{\text{PI},id,r,\texttt{rand}\}_k).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\{\text{PI},x\in lpi,y,z\}_k).\ddot{a}(\text{P\_\_PO},\{\text{PO},x,id,y,\texttt{rand}\}_k).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\{\text{PI},x\notin lpi,y,z\}_k).\ddot{a}(\text{SEEN},\{\text{PO},x,id,y,\texttt{rand}\}_k).\breve{\mathbb{A}}_{id,c,r,n}(lpi.x,cr,pr,lpo) \\
+ & \dot{a}(\{\text{PO},id,x\notin lpo,r,y\}_k).(K_2 \mid \ddot{a}(\text{SEEING},1).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr.x,pr,lpo.x))\setminus_a \\
+ & \dot{a}(\{\text{PO},id,x\notin cr\wedge x\in pr,r,y\}_k).\overline{a}.a(z).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr.x,pr,lpo) \\
+ & \dot{a}(\{\text{PO},id,x\notin cr\cup pr\wedge x\in lpo,r,y\}_k).\overline{a}.a(z).\ddot{a}(\text{SEEING},y).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr.x,pr,lpo) \\
+ & \dot{a}(\text{NR}).\breve{K}_{id,c,r,n}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\text{G\_\_LOST}).\ddot{a}(\text{P\_\_LOST},n).\breve{\mathbb{A}}_{id,c,r,0}(lpi,cr,pr,lpo) \\
+ & \dot{a}(\text{OTHERS}).\breve{\mathbb{A}}_{id,c,r,n}(lpi,cr,pr,lpo)
\end{array}
$$

$$K_k \stackrel{\text{def}}{=} a.\overline{a}(k).K_{k+1}$$

$$\breve{K}_{id,c,r,n}(lpi,cr,x.pr,lpo) \stackrel{\text{def}}{=} \begin{cases} \left.\begin{array}{l}\text{if } x\notin cr \text{ then}\\ \quad \breve{K}_{id,c,r,n+1}(lpi,cr,pr,lpo)\\ \text{else}\\ \quad \breve{K}_{id,c,r,n}(lpi,cr,pr,lpo)\end{array}\right\} & \text{si } x.pr\neq\varnothing \\ \breve{\mathbb{A}}_{id,c,r+1,n}(lpi,\varnothing,cr,lpo) & \text{si } x.pr=\varnothing \end{cases}$$

Fig. 4.55: Modélisation complète de l'*applet* AiMAN.

Les algorithmes de comptage que nous décrivons ci-après sont des algorithmes dans lesquels tous les nœuds sont à la fois compteurs et comptés, i.e. participent à l'activité coopérative, et exécutent le même algorithme. Cela ne signifie pas pour autant qu'il n'existe pas de nœud dans le réseau qui exécute un algorithme différent. Si (i) aucun nœud du réseau qui exécute l'algorithme ne peut être compté deux fois et (ii) si aucun nœud du réseau qui n'exécute pas l'algorithme peut-être compté alors l'algorithme de comptage fournit une borne inférieure du nombre de nœuds qui l'exécutent. Ce sont ces deux points que nous nous efforçons de garantir dans les versions les plus évoluées de nos algorithmes.

Les processus P_k que nous définissons ci-après décrivent chacun un algorithme de comptage dans lequel k représente le nombre de nœuds comptés. k vaut 0 à l'initialisation des nœuds.

4.3.1 Comptage non anonyme et sans garantie

Nous donnons la première version d'un algorithme de comptage qui permet à chaque nœud de compter les autres nœuds du réseau qui exécutent ce même algorithme, et ce quelle que

Algorithme (section)	Garantie	Anonymat	Mise à jour
4.3.1	Aucune	Non	Non
4.3.2	Borne inférieure	Non	Non
4.3.3	Borne inférieure	Partiel	Non
4.3.4	Borne inférieure	Total	Non
4.3.5	Borne inférieure	Total	Oui

TAB. 4.10: Résumé des caractéristiques de nos algorithmes de comptage.

soit leurs mobilités. Il utilise uniquement les primitives `send` et `receive` du *middleware* MiMAN.

Chaque nœud envoie régulièrement son identifiant et stocke les identifiants qu'il reçoit. Lors de la réception d'un nouvel identifiant (c'est à dire absent de la liste des identifiants déjà reçus), le processus l'ajoute à sa liste et incrémente le compteur.

La modélisation de cet algorithme est présentée en figure 4.56.

$P_{id,k}(l) \stackrel{\text{def}}{=} \overline{s}(id).P_{id,k}(l)$ *on envoie notre identifiant (pour être compté)*

$+ \quad r(x).$ *ou on reçoit un identifiant (on compte les autres)*

if $x \in l$ then *s'il est dans la liste*

$P_{id,k}(l)$ *on recommence*

else *sinon*

$P_{id,k+1}(l.x)$ *on incrémente le compteur des nœuds vus, on l'ajoute à la liste et on recommence*

FIG. 4.56: Application de comptage 1

La valeur du compteur correspond au nombre de nœuds du réseau qui exécutent l'algorithme s'il n'existe pas de nœud capable d'être compté à plusieurs reprises. On garantit donc une borne inférieure quelle que soit la mobilité des nœuds et des messages en l'absence de nœud malveillant, c'est à dire si on se limite à mettre en parallèle les nœuds qui exécutent ce processus, et uniquement ce processus. De manière évidente, un nœud qui enverrait comme identifiant un nombre aléatoire est un nœud malveillant : il pourrait être compté à plusieurs reprises.

Par ailleurs, on notera que les nœuds diffusent toujours le même message *Over The Air*. Il n'y a donc pas d'anonymat, même partiel dans ce système (cf. définition 85 page 107), puisqu'un nœud peut être reconnu aux messages qu'il envoie.

Nous améliorons progressivement cet algorithme en palliant certains des problèmes décrits précédemment lors de la définition de nouvelles applications de comptage qui sont basées sur les primitives plus évoluées que nous avons définies en section 4.2.

4.3.2 Comptage non anonyme avec borne inférieure garantie

Nous présentons maintenant la seconde version de l'algorithme de comptage qui utilise les primitives `sendId` ($\overline{si}$) et `receiveId` (ri) du *middleware* MiMAN. Nous supposons donc que les nœuds sont tels que définis en section 4.2.5 : ils disposent notamment d'un *Secure Element* qui exécute l'*applet* AiMAN. De manière classique, le *Secure Element* embarqué sur chacun des nœuds est supposé infaillible, c'est à dire qu'il est impossible de lire ou modifier son contenu ou de lui faire exécuter un autre processus que l'*applet* $\breve{\mathbb{A}}$.

La modélisation de cet algorithme est présentée en figure 4.57.

$$
\begin{array}{lcll}
P_k(l) & \stackrel{\text{def}}{=} & \overline{si}.P_k(l) & \textit{on envoie notre identifiant} \\
 & + & ri(x). & \textit{ou on reçoit un identifiant} \\
 & & \quad \text{if } x \in l \text{ then} & \quad \textit{s'il est dans la liste alors} \\
 & & \quad\quad P_k(l) & \quad\quad \textit{on recommence} \\
 & & \quad \text{else} & \quad \textit{sinon} \\
 & & \quad\quad P_{k+1}(l.x) & \quad\quad \textit{on incrémente le compteur des nœuds vus,} \\
 & & & \quad\quad \textit{on l'ajoute à la liste et on recommence}
\end{array}
$$

Fig. 4.57: Application de comptage 2

Grâce à l'utilisation des primitives `sendId` et `receiveId`, la borne inférieure sur le nombre de nœuds comptés est garantie. En effet, tout appel à la primitive ri de l'application par le *middleware* provient de la réception par le *middleware* d'un message chiffré par la carte à puce du nœud émetteur et qui par conséquent ne peut pas avoir été construit par un nœud qui ne possède pas ce *Secure Element*. De cette manière, les identifiants reçus au niveau applicatif sont nécessairement des identifiants de nœuds qui existent dans le réseau. L'application modélisée figure 4.57 ne compte donc que les nœuds qui possèdent notre support d'exécution. De plus, ces nœuds ne peuvent pas construire de message contenant un identifiant différent de leur identifiant réel car seul le *Secure Element* pourrait le faire, mais il est supposé de confiance. Aucun nœud ne peut donc être compté à plusieurs reprises en présentant un message forgé. Par conséquent, la valeur du compteur est une borne inférieure du nombre de nœuds dans le réseau. Ce sont ces mêmes arguments qui font que les algorithmes de comptage que nous présentons ensuite garantissent une borne inférieure sur le nombre de nœuds du réseau.

Par ailleurs, tous les messages envoyés *Over The Air* sont identiques pour un nœud donné. Ils sont tous égaux à $\{\text{ID}, id\}_k$. Ces messages sont donc des identifiants (cf. définition 82). Les nœuds qui ne possèdent pas le support d'exécution peuvent donc établir une correspondance entre ces identifiants et leurs émetteurs. L'algorithme ne permet donc pas aux nœuds d'être anonymes, même partiellement.

4.3.3 Comptage partiellement anonyme

L'algorithme précédent garantit une borne inférieure sur le nombre de nœuds qui composent le réseau. L'objectif de l'algorithme que nous présentons ci-dessous est de préserver cette borne inférieure tout en permettant aux nœuds d'être partiellement anonymes (cf. définition 85 page 107).

La modélisation de cet algorithme est présentée en figure 4.58.

$P_k(l) \stackrel{\text{def}}{=} \overline{sic}.P_k(l)$ *on envoie notre identifiant*
et notre horloge logique
$+ \quad ric(x,y).$ *ou on reçoit un identifiant x*
à l'instant (horloge logique) y
if $x \in l$ then *si x est dans la liste des déjà comptés alors*
$P_k(l)$ *on recommence*
else *sinon*
$P_{k+1}(l.x)$ *on incrémente le compteur des nœuds vus,*
on ajoute x à la liste et on recommence

FIG. 4.58: Application de comptage 3

Pour la même raison que l'algorithme précédent, la borne inférieure est garantie quelle que soient les nœuds qui composent le réseau.

L'utilisation de cet algorithme fait que tous les messages émis par un nœud sont différents. En effet, à chaque appel à la primitive `sendIdCounter` ($\overline{sic}$) ou `receiveIdCounter` (ric), l'horloge logique de l'*applet* est incrémentée. Or, les messages effectivement échangés sont de la forme $\{\text{IDC}, id, c\}_k$ (cf. section 4.2.3 page 117). Tous les messages construits par l'*applet* sont donc uniques et ne sont donc pas des identifiants pour les nœuds qui ne possèdent pas le support d'exécution que nous avons créé. Ils ne peuvent donc pas lier les messages qu'ils reçoivent aux nœuds qui les émettent. Les nœuds qui exécutent cet algorithme sont donc partiellement anonymes. En revanche, tous les nœuds qui possèdent le support d'exécution obtiennent, grâce à la primitive `receiveIdCounter` ($ric(x,y)$, l'identifiant du nœud émetteur pour chacun des messages reçus. Ils peuvent donc déterminer, à partir des messages qu'ils ont reçus, les nœuds qui les ont émis. Cet algorithme ne préserve donc pas totalement l'anonymat des nœuds sur le réseau.

4.3.4 Comptage totalement anonyme

Nous décrivons maintenant un algorithme de comptage totalement anonyme (cf. définition 86 page 108) qui utilise les primitives `ping` et `seen`. Sa modélisation est présentée en figure 4.59.

De la même manière que les deux algorithmes précédents, cet algorithme fournit une borne

P_k	$\stackrel{\text{def}}{=}$	$\overline{ping}.P_k$	*on envoie un ping ou*
	+	$seen.P_{k+1}$	*on détecte un nouveau voisin*
			et on incrémente le compteur

Fig. 4.59: Application de comptage 4

inférieure sur le nombre de nœuds qui possèdent le support d'exécution. De plus, les messages envoyés *Over The Air* par un nœud sont de la forme $\{\text{PI}, id, r, \texttt{rand}\}_k$ (cf. section 4.2.4 page 121) avec `rand` un nombre aléatoire. Les messages sont donc tous différents, permettant aux nœuds d'être au moins partiellement anonymes. Il est important de constater que l'utilisation du support d'exécution ne permet pas d'obtenir des informations sur les nœuds qui ont émis les messages. En effet, les identifiants des nœuds qui émettent les messages par l'intermédiaire de la primitive `ping` ne sont jamais accessible en dehors du *Secure Element*. Il est donc impossible, même pour un nœud qui exécute notre support d'exécution, d'établir un lien entre les messages qu'il reçoit et les nœuds qui les ont émis. Cet algorithme de comptage permet donc aux nœuds d'être totalement anonymes dans le réseau tout en garantissant une borne inférieure sur le nombre de nœuds comptés.

4.3.5 Comptage totalement anonyme des voisins

Aucune des applications précédentes ne permet d'obtenir une borne inférieure sur le nombre de voisins du nœud. En effet, lors de la réception d'un message par le *middleware*, rien n'indique que ce message provient directement du nœud dont l'identifiant est chiffré dans le message. En effet, il peut exister un nœud N' dont le rôle est de rejouer les messages qu'il reçoit. Ce nœud, qui favorise le comptage des autres nœuds dans le réseau en réémettant leurs messages empêche les nœuds d'évaluer leur nombre de voisins. D'un autre côté, la réémission de ces messages permet de favoriser voire d'accélérer le comptage des nœuds du réseau en propageant les messages. Afin de proposer une application capable de fournir une borne inférieure de ce nombre de voisins de manière totalement anonyme, nous proposons une nouvelle application. Pour être précis, nous définissons qu'un nœud N' est voisin d'un nœud N s'il est capable de faire en sorte que N reçoivent un message de type PO en réponse à son message de type PI avant qu'il (N) ne change de *round*. Un voisin est donc un nœud capable de répondre à un message de type PI suffisamment rapidement pour que l'émetteur de ce message reçoive la réponse dans le délai imparti. Cette notion de "suffisamment vite" est définie dans l'application par le délai entre deux appels successifs (un *round*) à la primitive `nextRound`. Néanmoins, il impossible de détecter une attaque de type *Man In The Middle*, et il est par conséquent impossible de créer une application qui garantisse que les nœuds comptés sont uniquement les voisins directs, c'est à dire les voisins à un saut.

La modélisation de cet algorithme est présentée en figure 4.60.

De la même façon que les algorithmes précédent, cette application fournit une borne inférieure du nombre de nœuds comptés. Les messages de type PI préservent totalement l'anonymat des nœuds. Les messages de type PO émis par les nœuds en réponse aux

P_k	$\stackrel{\text{def}}{=}$	$\overline{ping}.P_k$	*on envoie un ping*
	+	$seeing(x).P_{k+1}$	*ou on détecte un nouveau voisin*
			et on incrémente le compteur ou
	+	$\overline{nr}.P_k$	*ou change de round*
	+	$\overline{lost}.lost(x).P_{k-x}$	*ou on est informé que x nœud(s) a (ont) disparu*
			et on soustrait x de la valeur du compteur
			(lost retourne 0 s'il est appelé plus
			d'une fois durant le même round)

FIG. 4.60: Application de comptage 5

messages de type PI sont de la forme $(\text{B\_PO}, \{\text{PO}, x, id, y, \texttt{rand}\}_k)$ avec `rand` un nombre aléatoire. Ils sont donc tous uniques. Ces messages préservent donc l'anonymat des nœuds qui les émettent car il n'existe pas dans le support d'exécution de primitive permettant de récupérer, à partir d'un message de ce type, l'identifiant du nœud émetteur. Cette application garantit donc un anonymat total des nœuds du réseau et permet d'estimer le nombre de nœuds voisins.

Dans un contexte réel, le temps écoulé entre deux appels consécutifs à `nextRound` permettrait de minimiser les possibilités d'attaques de type *Man In The Middle*. C'est sur ce schéma que fonctionne la technologie NFC (*Near Field Communication*) [NFC] qui est progressivement déployée pour permettre le paiement sans contact.

4.4 Application 2 : collecte sécurisée de traces de voisinage

Nous présentons maintenant une seconde application qui utilise les primitives que nous avons précédemment définies. Elle permet la collecte sécurisée de traces de voisinage.

4.4.1 Objectif et intérêt

Le test d'une application ou d'un algorithme conçu pour les réseaux mobiles ad hoc en environnement réel est très difficile, les moyens humains par exemple devant être très importants. Les développeurs d'applications et les concepteurs d'algorithmes ont donc la plupart du temps recours à des simulateurs. Un grand nombre d'articles présentent des simulations dans lesquelles le modèle de mobilité utilisé par les nœuds n'est pas lié à une mobilité réelle. Il s'agit des modèles de mobilité tels que le *Random Walk* [CBD02] ou encore le *Random Waypoint* [CBD02] (cf. section 1.1.6 page 9). Certains simulateurs comme Madhoc [Hog05] permettent de définir un modèle de mobilité plus proche de la mobilité réelle. Ce dernier permet notamment de définir des points d'intérêt par lesquels les nœuds sont attirés. Ces points d'intérêt représentent par exemple les magasins à l'intérieur d'un centre commercial. Pour autant, les résultats obtenus grâce à l'utilisation de Madhoc,

bien que plus proches de ceux qui seraient obtenus en réalité, ne peuvent remplacer un test en environnement réel.

Un bon compromis est d'utiliser non pas un modèle de mobilité, mais une trace de mobilité collectée lors d'une exécution réelle. Concrètement, nous proposons de travailler sur des traces de voisinage l'important étant les voisins d'un nœud, plutôt que sa position géographique. En effet, la connaissance d'une proximité géographique entre des terminaux mobiles n'implique pas forcément qu'une communication est possible entre eux (cf. sections 1.1.2 page 7 et 1.1.3 page 7). Nous nous intéressons donc exclusivement aux traces de contact générées par communication réelle. Pour être précis, nous appelons traces de voisinage ou de contact les informations qui permettent de recréer, dans un simulateur par exemple, les interactions possibles (i.e. les rencontres intervenues) entre les nœuds au cours du temps dans un environnement réel.

Dans cette section, nous expliquons le fonctionnement de notre application de collecte de traces de voisinage et nous donnons son écriture en CiMAN. Nous présentons l'implémentation de celle-ci en section 4.5 page 145.

4.4.2 Horloges de Lamport

L'application que nous avons développée est exécutée sur des nœuds que nous supposons sans GPS et sans accès à une horloge globale. Chaque nœud collecte localement des informations, l'objectif étant que l'union de toutes ces informations permette d'ordonner certaines des rencontres qui ont conduites à l'acheminement d'un message d'un nœud à un autre, de sorte à en déduire des chemins dans le temps. Pour cela, nous utilisons un mécanisme proche des horloges logiques telles que Lamport les a définies dans l'article "*Time, clocks, and the ordering of events in a distributed system*" [Lam78]. Dans cet article Lamport décrit un mécanisme qui permet d'ordonner partiellement les évènements dans un système distribué en l'absence d'horloge globale. Ce mécanisme utilise une horloge logique propre à chacun des nœuds. Un nœud incrémente la valeur de son horloge logique locale lorsqu'une des actions qu'il souhaite ordonner (y compris les communications inter-processus) est effectuée. Nous nous sommes inspirés de ses travaux pour ordonner les communications entre les nœuds.

4.4.3 Modélisation

L'application de collecte de traces de voisinage utilise la primitive `sendIdCounter` qui envoie *Over The Air* un message contenant l'identifiant du nœud émetteur ainsi que son horloge logique à la date de création du message. La primitive associée à la réception d'un tel message est la primitive `receiveIdcounter`. Lors de sa réception, l'horloge logique du nœud récepteur est mise à jour. Elle prend la valeur de l'horloge logique associée au message incrémentée de un si et seulement si elle est supérieure à l'horloge logique locale et prend la valeur de l'horloge logique locale incrémentée de un sinon. Cette comparaison entre horloges logiques est effectuée dans l'*applet* et est donc cachée au *middleware*. Comme nous allons le voir, cette information qui n'est pas remontée par la primitive `receiveIdCounter` est pourtant essentielle pour notre application.

4.4.3.1 Algorithme de collecte de traces de voisinage

L'objectif de notre application est de permettre la reconstitution de trajets (chemins dans le temps) possibles entre les nœuds. Dans la suite, si le nœud N_2 reçoit un message M provenant directement ou indirectement de N_1 à la date t de son horloge logique, alors nous notons $N_1 \xrightarrow{t} N_2$. De même, si le nœud N_3 reçoit un message M de N_2 à l'instant t' alors nous notons $N_2 \xrightarrow{t'} N_3$. Nous souhaitons garantir que notre application de collecte de traces de voisinage soit telle que si nous collectons $N_1 \xrightarrow{t} N_2$ et $N_2 \xrightarrow{t'} N_3$ avec $t' > t$ alors il existe un trajet dans le temps entre N_1 et N_3. Ceci implique en particulier de ne pas collecter les traces de rencontre qui ne satisferaient pas cette propriété.

Pour cela, nous utilisons l'algorithme 5. Lors de la réception d'un message contenant l'identifiant et l'horloge logique du nœud émetteur, notre algorithme met à jour son horloge locale de la manière décrite par Lamport dans [Lam78], mais ne stocke l'identifiant du nœud émetteur et son horloge logique (trace) que si l'horloge logique contenue dans le message reçu est supérieure ou égale à l'horloge logique locale à la réception du message.

Cette approche ignore certains des échanges qui prennent place entre les nœuds du réseau, échanges que l'on ne sait pas ordonner tout en satisfaisant la propriété décrite ci-dessus.

Algorithme 5 Algorithme pour la collecte de traces de voisinage

```
{/* h est l'horloge logique locale */}
receive(id', h')
if h' ≥ h then
   h = h' + 1
   store(h, id')
else
   h = h + 1
end if
{/* En parallèle le nœud envoie périodiquement son identifiant et son horloge logique
locale en incrémentant à chaque fois son horloge logique */}
```

Nous démontrons maintenant que si l'algorithme 5 conduit au stockage de traces de voisinage $N_1 \xrightarrow{t} N_2$ et $N_2 \xrightarrow{t'} N_3$, en l'occurrence sur des nœuds différents, N_2 et N_3, avec $t' > t$, alors il existe un trajet dans le temps entre N_1 et N_3, c'est à dire qu'un message envoyé de N_1 à N_2 pourra être acheminé à une date ultérieure de N_2 à N_3. Ceci signifie que la date réelle (horloge globale physique) de réception du message de N_1 par N_2 est nécessairement inférieure à la date réelle d'envoi de ce message par N_2 à N_3.

Démonstration. Nous démontrons cette proposition par l'absurde. Nous considérons la configuration de la figure 4.61 constituée de traces de contact colletées par l'algorithme 5.

On souhaite donc montrer que $\alpha' > \alpha \Rightarrow H(send(M)) > H(receive(M))$ avec H la fonction théorique qui donne l'horloge globale du système.

Nous raisonnons par l'absurde. On suppose que les arcs (A_1) et (A_2) figure 4.61 existent et on cherche donc à montrer que $\alpha' > \alpha$ et $H(send(M)) \leq H(receive(M))$.

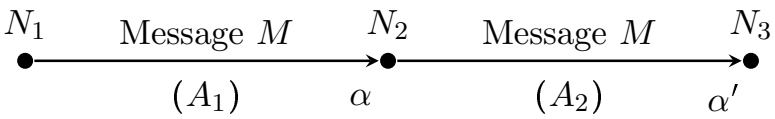


Fig. 4.61: Trajet avec horloge logique.

- Si $h_2 < h_3$ (cf. figure 4.62) : l'arc (A_2) de la figure 4.61 n'est pas conservé par l'algorithme 5 dans N_3 et n'existe donc pas. C'est donc impossible car en contradiction avec l'hypothèse.
- Si $h_2 \geq h_3$: $\alpha' = h_2 + 1$ (par utilisation de l'algorithme). Or $\alpha \geq h_2 + 1$ donc $\alpha \geq \alpha'$, ce qui contredit l'hypothèse $\alpha' > \alpha$.

□

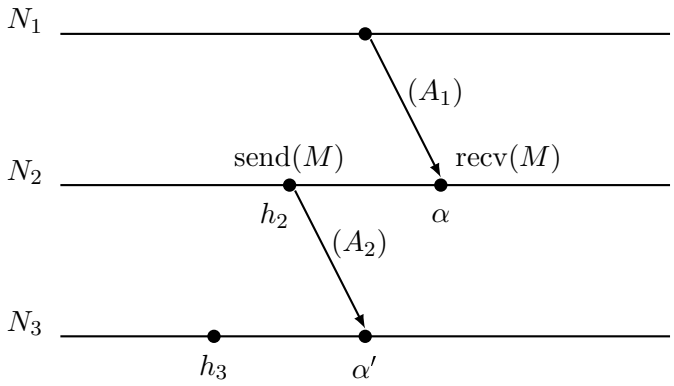


Fig. 4.62: Traces que l'on ne peut pas globalement ordonner.

L'algorithme 5 adapte son comportement en fonction de la valeur de l'horloge logique du message reçu, plus précisément en fonction de la réalisation ou non de la condition $h' \geq h$. Cette information, essentielle pour la collecte de traces de voisinage, n'est pas remontée au *middleware* par l'*applet* AiMAN. La condition $h' \geq h$ de l'algorithme 5 ne peut donc pas être vérifiée au niveau du *middleware* et donc au niveau de l'application. Pour palier ce manque, nous remplaçons[2] la primitive `receiveIdCounter` du support d'exécution par deux nouvelles primitives nommées `receiveIdLocalCounter` et `receiveIdRemoteCounter`.

4.4.3.2 Primitives `receiveIdLocalCounter` = *rilc* et `receiveIdRemoteCounter` = *rirc*

De la même manière que la primitive `receiveIdCounter`, les fonctions `receiveIdLocalCounter` et `receiveIdRemoteCounter` de niveau applicatif sont invoquées par le *middleware* grâce à un mécanisme de *callback*. La fonction `receiveIdLocalCounter` est appelée lorsque le nœud a reçu un message qui contenait un identifiant et une horloge logique

[2]Il s'agit bien de remplacer car sinon il y a interférence avec la primitive `receiveIdCounter` sur le traitement à effectuer. Dans une implémentation de ces primitives, l'appel de l'une ou l'autre des versions pourrait être géré par un *flag*.

strictement inférieure à l'horloge logique locale de l'*applet* au moment de la réception. La fonction `receiveIdRemoteCounter` est quant à elle appelée lorsque l'horloge logique du message reçu est supérieure ou égale à l'horloge logique locale de l'*applet* au moment de la réception.

Le diagramme de séquence de ces primitives est présenté figure 4.63.

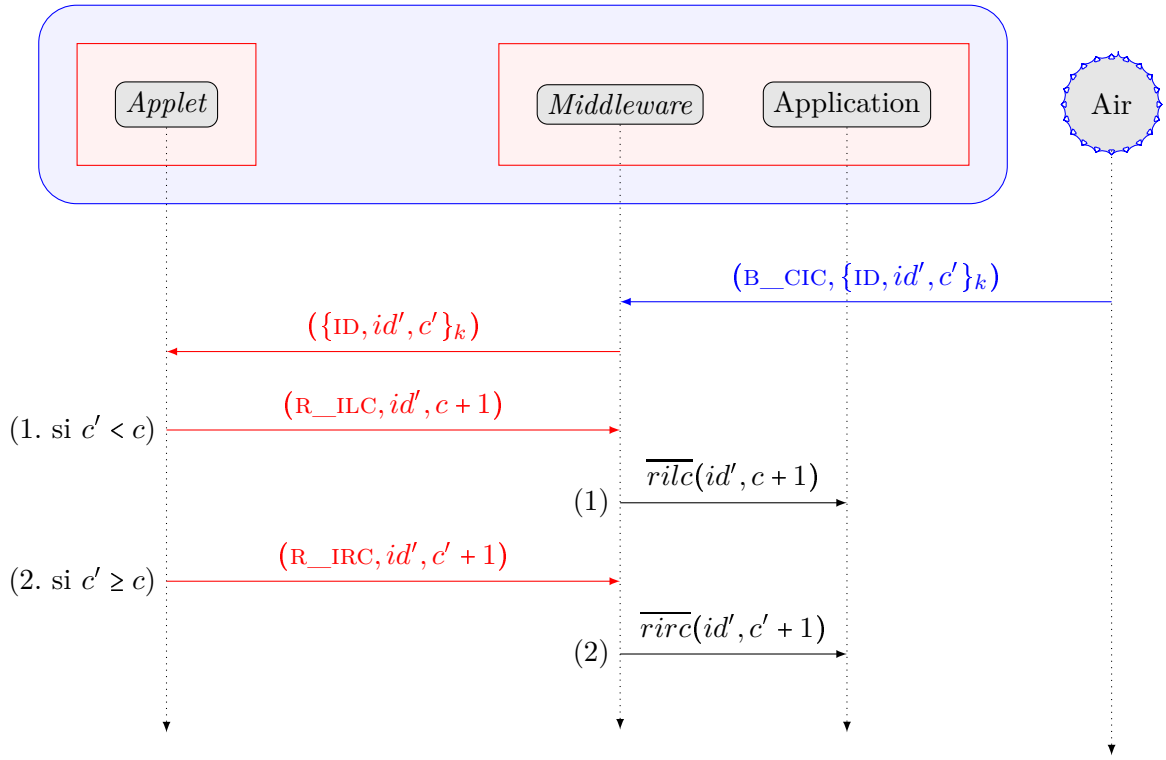


FIG. 4.63: Diagramme de séquence de la primitive `receiveIdRemoteCounter`.

Modélisation. Les primitives `receiveIdLocalCounter` et `receiveIdRemoteCounter` sont modélisées figure 4.64. Lors de la réception d'un message *Over The Air* de la forme $(\text{B\_CIC}, x)$ (avec x variable) par la partie $\breve{M}'_{rilrc}$ du *middleware* $\breve{M}_{rilrc}$, ce dernier fait suivre x à l'*applet* $\breve{A}_{id,c}$ qui le déchiffre et retourne au *middleware* $\breve{M}_{rilrc}$ l'identifiant provenant du *Secure Element* qui a construit ce message ainsi que la nouvelle valeur de l'horloge logique locale. Cette nouvelle valeur est le maximum entre la valeur de l'horloge logique contenue dans le message reçu et la valeur de l'horloge logique du *Secure Element* local, incrémenté de un. Si le maximum était l'horloge logique du message reçu alors l'*applet* retourne le message $(\text{R\_IRC}, x, y)$ au *middleware*, sinon elle retourne le message $(\text{R\_ILC}, x, y)$. Ce couple (identifiant, horloge logique) est remonté par le *middleware* à l'application P_{rilrc} par une action de type $\overline{rilc}$ ou $\overline{rirc}$. Cette action est capturée par l'application qui peut alors effectuer le traitement approprié en fonction du service qu'elle rend.

Nœud	N_{id}	$\stackrel{\text{def}}{=}$	$[\![[\breve{A}_{id,0}] \parallel [(\breve{M} \mid P) \setminus_{ric}]]\!]$
Middleware	$\breve{M}_{ric}$	$\stackrel{\text{def}}{=}$	$\breve{M}'_{ric} \mid \breve{M}''_{ric}$
avec	$\breve{M}'_{ric}$	$\stackrel{\text{def}}{=}$	$\mathring{a}\ (\text{B\_\_CIC}, x).\ddot{a}(x).\breve{M}'_{ric}$
		$+$	$\mathring{a}\ (\text{OTHERS}).\breve{M}'_{ric}$
et	$\breve{M}''_{ric}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\text{P\_\_IRC}, x, y).\overline{rirc}(x, y).\breve{M}''_{ric}$
			$\dot{a}(\text{P\_\_ILC}, x, y).\overline{rilc}(x, y).\breve{M}''_{ric}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{M}''_{ric}$
Applet	$\breve{A}_{id,c}$	$\stackrel{\text{def}}{=}$	$\dot{a}(\{\text{IDC}, x, y > c\}_k).\ddot{a}(\text{P\_\_IRC}, x, y+1).\breve{A}_{id,y+1}$
		$+$	$\dot{a}(\{\text{IDC}, x, y \leq c\}_k).\ddot{a}(\text{P\_\_ILC}, x, c+1).\breve{A}_{id,c+1}$
		$+$	$\dot{a}(\text{OTHERS}).\breve{A}_{id,c}$
Application	P_{ric}	$\stackrel{\text{def}}{=}$	$rilc(x, y).0 + rirc(x, y).0$

FIG. 4.64: Modélisation des la primitives `receiveIdLocalCounter` et `receiveIdRemoteCounter`.

4.4.3.3 Application de collecte de traces

L'application que nous avons développée est modélisée figure 4.65. Comme pour les applications de comptage, nous supposons que les nœuds disposent du support d'exécution que nous avons créé (MiMAN et AiMAN). Dans la figure 4.65, l représente les traces de voisinage collectées, c'est à dire la liste des couples (horloge logique, identifiant) que l'application stocke effectivement.

$P(l)$	$\stackrel{\text{def}}{=}$	$\overline{sic}.P_{c+1}(l)$	*on envoie le couple (id,horloge)*
	$+$	$rilc(x, y).P(l.(y, x))$	*ou on reçoit un couple (id,horloge)*
			et on le stocke

FIG. 4.65: Modélisation de l'application de collecte de traces de voisinage

4.4.4 Exemple de traces de voisinage

Nous présentons maintenant un exemple de collecte de traces de voisinage. Cet exemple ne découle pas de l'expérimentation réelle sur le point de débuter (cf. section 4.5) mais est un exemple créé pour illustrer notre propos. Nous supposons que le réseau est composé de trois nœuds N_1, N_2 et N_3 qui collectent des traces de voisinage. Chaque nœud stocke localement un ensemble de couples (horloge logique, émetteur). Nous représentons ces

N_1		N_2		N_3	
Horloge	Id	Horloge	Id	Horloge	Id
5	2	11	3	2	1
16	2	23	1	40	2
33	3	37	3	46	1
		50	3		

TAB. 4.11: Exemple de traces de voisinage

couples dans le tableau 4.11

Grâce à ces traces de voisinage, on peut conclure l'existence des trajets élémentaires suivants :

$N_2 \xrightarrow{5} N_1$ $N_3 \xrightarrow{11} N_2$ $N_1 \xrightarrow{2} N_3$
$N_2 \xrightarrow{16} N_1$ $N_1 \xrightarrow{23} N_2$ $N_2 \xrightarrow{40} N_3$
$N_3 \xrightarrow{33} N_1$ $N_3 \xrightarrow{37} N_2$ $N_1 \xrightarrow{46} N_3$
$N_3 \xrightarrow{50} N_2$

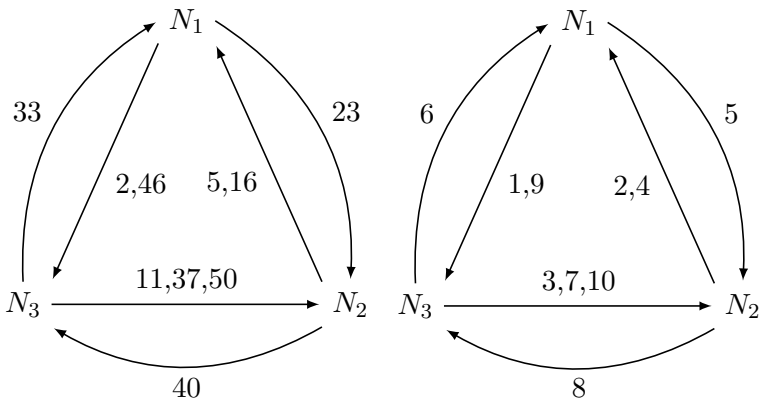
On peut en déduire l'existence de trajets construits par l'enchainement de ces trajets élémentaires, par exemple $N_2 \xrightarrow{5} N_1 \xrightarrow{23} N_2$

Rappelons que la représentation $N \xrightarrow{t} N'$ signifie que N' a reçu à la date t de son horloge locale, un message en provenance directe ou indirecte de N.

Les communications et donc les rencontres qui ont pris place dans cet exemple sont représentées par le graphe de la figure 4.66. Cette représentation sous forme de graphe temporel orienté permet de mettre en lumière certaines caractéristiques non triviales à la lecture des traces de voisinage. Par exemple, N_2 ne peut pas répondre à un message provenant directement de N_1 et cela, même si la requête de réponse est acheminée par les autres nœuds du réseau.

L'algorithme qui permet d'extraire les trajets possibles entre l'ensemble des nœuds à partir des traces de voisinage collectées sur chaque nœud est trivial, il consiste en l'union des trajets collectés sur chacun des nœuds. Le graphe temporel alors obtenu est présenté figure 4.66a. L'opération de renommage des dates dont le résultat est visible figure 4.66b a pour objectif de faciliter la lecture du graphe temporel. Elle se fait simplement par un tri croissant des dates existantes suivi d'un renommage par les premiers nombres entiers naturels.

De la même manière que l'application de comptage de la section 4.3.3, cette application permet aux nœuds d'être partiellement anonymes. En effet, ces deux applications ne communiquent *Over The Air* qu'en utilisant la primitive `sendIdCounter`.



(a) Graphe temporel (b) Graphe temporel simplifié

Fig. 4.66: Graphes temporels qui découlent de l'exemple de collecte de traces de voisinage

4.5 Mise en œuvre

Nous présentons maintenant la mise en œuvre des primitives, du support d'exécution et des applications que nous avons modélisés.

4.5.1 Caractéristiques du support d'exécution

Lors de nos expérimentations, nous avons utilisé une flotte de téléphones portables équipés du système d'exploitation Android [And] développé par Google. Les téléphones dont nous disposons sont des téléphones de développement de type HTC G1. Nous disposons aussi de téléphones grand public Samsung Galaxy et Motorola Milestone, qui ne sont pas des téléphones de développement, c'est à dire qui ne proposent pas d'accès *root* à l'utilisateur. Il est cependant souvent possible d'exploiter une faille de sécurité du système d'exploitation afin d'obtenir un accès *root* sur un téléphone qui n'est pas un téléphone de développement. Il existe même une application (nommée *One Click Root*) compatible avec de nombreux modèles et qui permet de *rooter* son téléphone très simplement.

Pour commencer, nous présentons le système d'exploitation Android sur lequel est basé notre *middleware*.

4.5.1.1 Système d'exploitation Android

Android est un système d'exploitation créé pour les *smartphones* et les PDAs. Il est en partie *open source* et il est distribué par Google sous licence Apache 2 [Apa]. Il a été initialement conçu par une startup nommée Android qui a été rachetée par Google le 15 novembre 2007. A l'heure de la rédaction du présent document, la version la plus récente du système d'exploitation est la 2.2 dénommée Froyo (Frost Yogourt). Comme toutes les versions d'Android existantes, elle est basée sur un noyau Linux et embarque une machine virtuelle Java *open source* sous licence Apache 2 nommée Dalvik. Cette machine virtuelle

a été spécialement conçue pour les environnements contraints en terme de capacité de calcul et de mémoire que sont les *smartphones* et les PDAs.

Caractéristiques techniques. FroYo est la sixième version d'Android. Elle suit les versions 0.9 (alpha), 1.0 (bêta), 1.5 (Cupcake), 1.6 (Donut) et 2.0/2.1 (Eclair). La prochaine version, nommée Gingerbread, est prévue pour le 4ème trimestre 2010. Toutes les versions d'Android existantes reposent sur un noyau Linux. La version sur laquelle nous avons développé notre bibliothèque et les applications qui l'utilisent est une version proche de la version 2.2. HTC ne fournit pas cette version d'Android pour les téléphones G1 dont nous disposons. La dernière version officiellement distribuée est la version 1.6 (Donut). Cependant, afin de travailler sur la version la plus récente d'Android, nous avons souhaité installer Android FroYo sur les HTC G1. Nous avons pour cela utilisé une version non officielle d'Android FroYo développée par un groupe indépendant nommé CyanoGen [Cya].

Diversification des supports. Aujourd'hui un grand nombre de téléphones grand public sont équipés du système d'exploitation Android. Selon le NPD Group [NPD] dont l'objectif est l'étude de la consommation (au sens achats des consommateurs), les ventes de téléphones Android ont été plus importantes que les ventes d'iPhones au premier trimestre 2010 aux États-Unis. La diversification des dispositifs qui supportent ce système d'exploitation a déjà commencé. Par exemple, des tablettes mobiles équipées d'Android sont d'ores et déjà commercialisées. La *Google TV*, une télévision reliée à Internet, équipée à la fois d'Android et de Chrome (le navigateur Internet de Google), est sur le point d'être commercialisée. Enfin, certains grands constructeurs de voitures américains ont signé des partenariats avec Google afin de fournir une version d'Android dans les ordinateurs de bord des véhicules à venir.

Intérêt. Android est basé sur un noyau Linux et il est en partie *open source*. Google propose des téléphones non bridés permettant d'obtenir des droits super-utilisateur, téléphones que l'on dit de développement. Ils permettent aux développeurs de modifier en profondeur le système d'exploitation, augmentant ainsi les possibilités offertes. Comme indiqué précédemment, HTC ne fournit pas de version ultérieure à la version 1.6 pour les téléphones de développement HTC G1. Cependant les développeurs qui ont réussi à modifier la version 2.2 (FroYo) pour l'adapter à ce support n'ont pu le faire que parce qu'une grande partie du code d'Android est *open source*. Cette ouverture a permis la création de projets innovants, notamment en matière de sécurité. Citons par exemple, *SecureFileManager* [SFA], un gestionnaire de fichiers sécurisé grâce au chiffrement prenant place dans un *Secure Element*, ou encore BTPCSC [SFA] (*Bluetooth PC/SC support for Android phones*) qui permet d'utiliser son téléphone Android comme un lecteur de cartes à puce depuis n'importe quelle application qui le nécessite. Ces logiciels sont hébergés sur le site web du projet *seek-for-android* [SFA], dont le but est de proposer pour Android des outils de sécurité basés sur un support matériel.

La sécurité dans un environnement mobile. Les nœuds qui constituent un réseau ad hoc fortement mobile sont contraints en terme de capacités de calcul et de communication.

Comme expliqué précédemment, nous supposons en particulier qu'ils ne disposent d'aucun lien de communication stable. Il ne peuvent donc pas communiquer avec une hypothétique autorité de certification qui permettrait d'utiliser des approches classiques de la sécurité. Nous distribuons donc cette sécurité sur les nœuds en supposant la présence sur chacun d'eux d'un *Secure Element*. La plupart des téléphones portables par exemple disposent d'un *Secure Element*, la carte SIM (Subscriber Identity Module). Elle semble donc être le dispositif idéal pour la sécurisation des échanges dans un réseau ad hoc fortement mobile. Il existe deux types de cartes SIM :

1. Carte SIM de l'opérateur. Elle permet d'authentifier l'utilisateur du téléphone sur le réseau de l'opérateur. Il peut alors utiliser le réseau de téléphonie mobile. Cependant, ces cartes SIM, qui sont vendues par les opérateurs, sont verrouillées et il est impossible d'y installer des applications (ou *applets* JavaCard).
2. Carte SIM de développement. Elle est directement vendue par les fabricants de cartes SIM, comme par exemple Gemalto. Elle est fournie avec l'ensemble des informations (clefs d'authentifications) qui permettent l'installation d'une *applet* de notre conception et elle n'est pas liée à un opérateur. Les téléphones dans lesquels elles sont placées ne permettent en revanche plus d'utiliser le réseau d'un opérateur de téléphonie mobile.

Aucune de ces deux solutions n'est donc satisfaisante et l'utilisation d'une unique carte SIM par téléphone ne semble donc pas être une solution pour la sécurisation des réseaux ad hoc fortement mobiles.

Il existe cependant des téléphones portables qui disposent de deux emplacements pour carte SIM. Ces téléphones pourraient donc embarquer à la fois une carte SIM pour l'authentification de l'utilisateur sur le réseau d'un opérateur de téléphonie mobile et une carte SIM de développement sur laquelle peuvent être installés des *applets* JavaCard. Ces téléphones dual-SIM sont très rares en Europe.

Une autre solution est l'utilisation d'un adaptateur dual SIM permettant de transformer la plupart des téléphones en téléphones dual SIM. Contrairement à la plupart des approches dual SIM native, l'utilisation d'un adaptateur empêche une utilisation simultanée des deux cartes SIM. Ils ne semblent donc pas adaptés non plus à notre utilisation.

4.5.1.2 Carte microSD

La société Giesecke & Devrient a récemment proposé un nouveau *token* embarquant un *Secure Element* ; il s'agit de cartes au format microSD. Tous les téléphones et *smartphones* de dernière génération permettent l'utilisation d'une carte microSD comme support de stockage externe. C'est le cas des téléphones qui forment la flotte de terminaux Android dont nous disposons. L'utilisation d'une carte microSD n'empêche pas l'utilisation d'une carte SIM ; le format microSD semble donc être une bonne approche pour l'utilisation d'un *Secure Element* sur des téléphones portables et c'est donc la solution que nous avons retenue.

Caractéristiques techniques. Une carte microSD sécurisée est une carte au format microSD standard. Elle embarque une quantité de mémoire importante (relativement à la mémoire présente sur les cartes à puce) pouvant atteindre plusieurs centaines de Méga-octets (voire plusieurs Giga-octets) et un élément de sécurité de type JavaCard. Le constructeur de cette carte microSD met à la disposition des développeurs un *driver* spécifique qui permet au téléphone d'utiliser la carte microSD à la fois comme support de stockage externe et comme élément de sécurisation. La communication entre le téléphone et la JavaCard embarquée dans la carte microSD utilise le protocole APDU standard. Encore une fois, le développement d'un pilote ad hoc sous Android est simplifié par le fait qu'Android est en grande partie *open-source*. De plus, un port microSD standard suffit à l'utilisation de cette carte, il n'est pas nécessaire que le port soit compatible SDIO, une JavaCard n'initialisant jamais une communication. La mise à disposition de ces cartes pour les développeurs non partenaires du constructeur est relativement récente ; elle date du 7 septembre 2010.

Intérêt. Comme expliqué précédemment, l'utilisation d'une carte microSD sécurisée apporte un grand nombre d'avantages par rapport à une solution de sécurisation utilisant la carte SIM. Tout d'abord, cette approche permet au téléphone mobile de continuer à exploiter le réseau sans fil fourni par un opérateur tout en utilisant l'outil de sécurisation qu'est la carte microSD. Ensuite, la plupart des téléphones sont éligibles à cette approche car ils proposent un port microSD. Enfin, un changement d'opérateur de téléphonie mobile entraîne un changement de carte SIM, et une remise en cause éventuelle des outils de sécurité que nous avons installés.

Nous pensons donc que l'utilisation d'un *Secure Element* embarqué dans une carte microSD est l'approche la plus adaptée au contexte dans lequel nous nous trouvons.

4.5.2 Développement

Nous présentons maintenant le développement effectué dans le cadre de cette thèse. Le support d'exécution développé l'a été pour des terminaux de type Android. Comme expliqué précédemment, Android repose sur un noyau Linux et une machine virtuelle Java nommée Dalvik. Le *middleware* que nous avons développé est donc principalement développé en Java, la configuration du *chipset* Wi-Fi en mode ad hoc ayant cependant due être réalisée par du code natif (cf. section 4.5.2.2).

4.5.2.1 Développement sous Android

Le développement sous Android se fait par l'intermédiaire de quatre processus de base que nous décrivons ci-après.

Activity. Les *Activity* sont des applications qui fournissent une interface graphique afin de permettre à l'utilisateur d'interagir avec elle. Dans la suite, nous parlerons simplement d'application.

Service. Un service est un processus qui s'exécute en arrière-plan. Il n'interagit pas avec l'utilisateur mais uniquement avec les autres processus (*Activity*, *Service*, *BroadcastReceiver* et *ContentProvider*).

Exemple : Une application de lecture de musique utilise ces deux types de processus. Le processus de type *Activity* permet à l'utilisateur de choisir les morceaux à écouter, de changer de piste, de régler le volume. Le processus de type *Service* réalise la lecture de la piste de musique. De cette manière, l'utilisation d'une autre application ne stoppe par la lecture en cours, le service pouvant continuer à fonctionner en arrière-plan.

BroadcastReceiver. Un composant de type *BroadcastReceiver* est un composant qui est à l'écoute des messages que peut envoyer le système d'exploitation, les *Activity*, les *BroadcastReceiver* ou encore les *Services*. Ces messages peuvent concerner les évènements qui se produisent tels que la réception d'un appel par le téléphone. Il permet alors aux applications de fournir un traitement adapté. Le lecteur de musique de l'exemple précédent pourrait automatiquement mettre la lecture en pause pour la durée de l'appel.

ContentProvider. Le *ContentProvider* est un composant qui permet la mutualisation de contenus qu'il peut ainsi fournir à plusieurs applications ou service, évitant la duplication des données. Par exemple, l'application de répertoire téléphonique fournie avec Android est un *ContentProvider*.

4.5.2.2 Communication entre les nœuds et sécurité

Les nœuds qui constituent les réseaux mobiles ad hoc sont pour la plupart équipés des technologies Bluetooth et Wi-Fi. Ces deux technologies de communication ont des caractéristiques très différentes. Bluetooth est une technologie de communication conçue pour l'interconnexion de dispositifs qui ne partagent aucune information préalablement à la communication. Les nœuds peuvent effectuer une découverte de voisinage standardisée par la norme 802.15.1 [IEE] qui leur permet ensuite de communiquer. Les versions de Bluetooth les plus répandues sont les versions 2.0 et 2.1 qui permettent des débits de l'ordre de 3 Mbit/s. La version 3.0 dont la norme a été adoptée le 21 avril 2009, permet des débits de l'ordre de 24 Mbit/s [Blua]. Les premiers équipements utilisant cette norme commencent à apparaître (c'est le cas par exemple du téléphone Samsung Galaxy S).

La technologie Wi-Fi quant à elle n'a pas été pensée pour permettre une communication entre des dispositifs sans configuration préalable. Pour communiquer, les nœuds doivent appartenir à la même cellule dont l'identifiant est nommé SSID (*Service Set Identifier*) [WIF] et posséder la même plage d'adressage IP. Ces informations sont indispensables pour permettre aux nœuds de communiquer.

Convergence Wi-Fi / Bluetooth. Il est intéressant de remarquer que l'on assiste à une convergence de ces deux technologies de communication. D'un côté, Bluetooth utilise depuis la version 3.0, une technologie proche de la norme Wi-Fi afin de gagner en débit. De l'autre côté, la Wi-Fi Alliance vient de se doter d'une spécification nommée *Wi-Fi*

Direct qui permet l'interconnexion des nœuds en Wi-Fi sans que ceux-ci n'aient à être préalablement configurés. Les technologies de communication semblent en cela converger.

Lors de nos développements, nous avons privilégié l'utilisation de la technologie Wi-Fi car la technologie Bluetooth permet uniquement des communications deux à deux entre les nœuds. Le Wi-Fi quant à lui, permet une communication de type *broadcast*, dont les applications pour réseaux mobiles ad hoc doivent pouvoir tirer profit.

Configuration du *chipset* Wi-Fi en mode Ad Hoc. Le SDK (*Software Development Kit*) d'Android, en version 2.2 à l'heure de la rédaction du présent document, ne permet la communication entre les nœuds qu'en la présence de points d'accès. Nous souhaitons au contraire que les nœuds puissent communiquer directement les uns avec les autres, sans la présence d'une infrastructure, i.e. en mode ad hoc.

Pour le développement d'applications natives, Google propose un NDK (*Native Development Kit*). Celui-ci permet de développer des applications qui ne sont pas exécutées par la machine virtuelle Dalvik mais directement par le noyau Linux. La configuration du *chipset* en mode ad hoc n'est cependant par supportée non plus par le NDK.

Android utilise le logiciel libre `wpa_supplicant` [WPA] pour configurer de l'interface Wi-Fi, logiciel qui autorise la configuration en mode ad hoc. Par ailleurs, le module noyau associé au *chipset* Wi-Fi des téléphones Android supportant (en ce qui concerne nos téléphones) ce mode, nous avons pu forcer le passage en mode ad hoc de l'interface Wi-Fi sur les téléphones HTC G1.

ZeroConf. Afin de faciliter nos expérimentations, les communications que nous avons mises en place utilisent la pile protocolaire UDP/IP. Nous utilisons exclusivement des communications par *broadcast* et, dans nos applications, les nœuds ne possèdent pas forcément d'identifiant. Ces couches protocolaires non nécessaires à la mise en place de nos applications ont cependant facilité le développement de notre support d'exécution. A terme, nous envisageons de nous en affranchir.

C'est la couche IP que nous utilisons actuellement pour l'adressage des nœuds et chaque nœud doit donc posséder une adresse IP unique. Pour cela, nous utilisons la spécification ZeroConf [Zer]. Cette spécification permet à chaque nœud de déterminer s'il peut ou non utiliser une adresse IP donnée. Son fonctionnement, relativement simple, est le suivant. Les nœuds choisissent une adresse IP de manière aléatoire. Une fois choisie, ils interrogent les nœuds présents dans le voisinage pour vérifier qu'aucun nœud ne possède déjà cette adresse. Si un nœud possédant déjà cette adresse IP est détecté, alors le processus est répété, jusqu'à ce qu'aucun conflit ne soit plus détecté. Dans notre cas, la dynamique du réseau nous oblige à effectuer ces opérations de manière périodique.

Utilisation d'une carte microSD sécurisée. Comme expliqué précédemment, la sécurisation du réseau passe par l'utilisation d'un *Secure Element*. Celui que nous utilisons est embarqué dans une carte microSD. Nous devons donc ajouter au noyau Linux sur lequel repose Android un module adapté au dialogue avec ce périphérique ainsi qu'un ensemble de bibliothèques et d'outils qui permettent de dialoguer avec une JavaCard et qui sont

compatibles PC/SC [PCS]. Les outils et les *patchs* à appliquer au système d'exploitation pour l'utilisation de ce *Secure Element* sont disponibles sur la page web du projet *seek-for-android* [SFA]. Depuis peu, une bibliothèque est proposée par le fabriquant de la carte microSD que nous utilisons afin de faciliter la communication avec la carte à puce qu'elle embarque. Après application des *patchs*, recompilation complète du système et installation sur le HTC G1, le *Secure Element* peut être utilisé.

4.5.3 Architecture de notre support d'exécution

Nous présentons maintenant l'architecture de notre support d'exécution, les interfaces en lien avec les primitives que nous avons précédemment présentées ainsi que le service et les applications que nous avons développées. L'architecture est schématisée figure 4.67.

4.5.3.1 Primitives

AIDL (*Android Interface Definition Language*) est un langage de description d'interface (IDL). L'utilisation d'une interface de type AIDL permet à une application d'utiliser une primitive fournie par le *middleware* grâce à un mécanisme de type RPC (*Remote Procedure Call*). Ce mécanisme est essentiel pour permettre à un ensemble d'applications d'utiliser de manière dynamique un service en cours d'exécution. L'ensemble des fonctions que peut appeler une application est présentée ci-dessous.

```
package com.miman.service;

interface IActivePrimitives {
  /* Register and unregister primitive callbacks */
  void registerCallbacks(in ICallbackPrimitives cp);
  void unregisterCallbacks(in ICallbackPrimitives cp);

  void send(byte[] data);
  void sendId();
  void sendIdCounter();
  void ping();
  void nextRound();
  void getLost();
}
```

La méthode `registerCallbacks`, que l'application appelle lui permet de s'enregistrer auprès du *middleware* afin que ce dernier puisse appeller les méthodes de type callback (par exemple `receive`, `receiveId`, etc.) implémentées dans l'application. De manière symétrique, la méthode unregisterCallback permet à l'application de se désenregistrer auprès du *middleware* afin que ce dernier n'appelle plus les méthodes de *callback*.

Pour que le *middleware* puisse appeler, par un mécanisme de type RPC les primitives précédentes, l'application doit donc implémenter l'interface ICallbackPrimitives décrite ci-dessous. La classe qui implémente cette interface est donnée en paramètre à la méthode registerCallbacks lors de l'enregistrement de l'application auprès du *middleware*.

```
package com.miman.service;

interface ICallbackPrimitives {
  void receive(in byte[] buffer);
  void receiveId(int id);
  void receiveIdLocalCounter(int id, int c);
  void receiveIdRemoteCounter(int id, int c);
  void seen();
  void seeing(int n);
  void lost(int n);
}
```

4.5.3.2 Service MiMAN

Le service MiMAN que nous avons développé est le cœur du *middleware*. Lancé automatiquement au démarrage du système Android, il est en lien direct avec les applications qui souhaitent l'utiliser et il gère l'interface de communication Wi-Fi du terminal ainsi que la communication avec le *Secure Element* qu'il embarque.

Démarrage automatique Lors du démarrage d'un terminal Android, lorsque l'initialisation du système d'exploitation est terminée, ce dernier diffuse un message de type *BOOT_COMPLETED* qui peut être reçu par une application de type *BroadcastReceiver* (cf. section 4.5.2.1 page 148). Une fois réveillée, cette application peut, par exemple, démarrer un service. C'est précisément cette approche que nous utilisons pour lancer automatiquement notre support d'exécution au démarrage du système. Nous avons développé un *BroadcastReceiver* nommé BootService dont le rôle est de lancer le *middleware* à la réception du message *BOOT_COMPLETED*. Lors de la réception de ce message, la méthode onReceive() de BootService est appelée. Cette méthode lance le *middleware*. Le contenu de cette méthode est donné ci-dessous.

```
package com.miman.service;

public class BootService extends BroadcastReceiver {

  public void onReceive(final Context context,
                        final Intent intent) {
    Intent mServiceIntent = new Intent();
    String action = intent.getAction();
    if (Intent.ACTION_BOOT_COMPLETED.equals(action)) {
      mServiceIntent.setAction("com.miman.service.MiMANService");
      context.startService(mServiceIntent);
    }
  }
}
```

Lors de la réception d'un message de type *BOOT_COMPLETED*, le *BroadcastReceiver* BootService démarre le service MiMANService qui correspond au cœur du support d'exécution. MiMANService initialise alors le *chipset* Wi-Fi en mode ad hoc et attend la connexion de clients, c'est à dire d'applications locales souhaitant l'utiliser.

Les messages que BootService a le droit de recevoir sont spécifiés dans le fichier XML (eXtensible Markup Langage) nommé AndroidManifest.xml qui est indispensable à l'installation d'une application ou d'un service. Le fichier AndroidManifest.xml associé à notre projet précise que le message *BOOT_COMPLETED* peut être reçu par l'application BootService.

```
<receiver android:enabled="true" android:name="BootService">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED">
    </action>
  </intent-filter>
</receiver>
```

Nous présentons maintenant la manière dont le service MiMAN configure le *chipset* Wi-Fi en mode ad hoc.

Communication OTA Avant de pouvoir communiquer *Over The Air*, le *middleware* doit configurer l'interface Wi-Fi afin (i) de passer le *chipset* en mode ad hoc, et (ii) d'obtenir une adresse IP qui n'est pas déjà utilisée par un voisin. Le code permettant ces opérations peut être différent d'un téléphone à l'autre. Nous avons donc défini l'interface AdhocManager.

```
public interface AdhocManager {
  public void startAdhoc();
  public void stopAdhoc();

  public boolean isAdhocEnabled();
  public InetAddress getIpAddress();
  public InetAddress getBroadcastAddress();
}
```

Cette interface est implémentée par les classes HtcG1AdhocManager.java et SamsungGalaxyAdhocManager.java qui effectuent les opérations nécessaire au passage de l'interface Wi-Fi en mode ad hoc en fonction du type de téléphone sur lequel est exécuté le *middleware*. Dans un travail futur, nous pourrons ajouter une classe MotorolaMilestoneAdhocManager qui permettra de passer les interface Wi-Fi des téléphones Motorola Milestone dont nous disposons en mode ad hoc.

Afin de compléter la démarche, il serait intéressant de voir si Android sait tirer parti du *Wi-Fi Direct* [WIF] sur le Samsung Galaxy S. Le succès commercial de ce téléphone et sa certification *Wi-Fi Direct* peuvent laisser penser qu'une prochaine version d'Android et de son SDK permettront d'utiliser le Wi-Fi des terminaux en mode ad hoc sans modification complexe de l'OS et donc sans que l'utilisateur n'ait besoin des droits administrateur sur son mobile.

SecureElement et SecureProcess Notre middleware fait aussi le lien entre les applications et le *Secure Element* embarqué sur le nœud. Afin de permettre au *middleware* d'utiliser un autre *Secure Element* que la carte microSD que nous avons présentée précédemment, nous proposons l'interface SecureElement qui abstrait la nature réelle du *Secure Element*.

```
public interface SecureElement {
  public void init();
  public byte[] sendAPDU(byte[] apdu);
  public void halt();
}
```

Cette interface est implémentée par les classes MicroSD et SecureProcess que nous avons développées. La classe MicroSD a pour objectif de faire le lien avec l'*applet* qui est exécutée dans le *Secure Element* (dans la carte microSD) tandis que la classe SecureProcess permet quant à elle de simuler l'exécution d'une *applet* sur un Secure Element. Nous avons implanté cette double approche en particulier car les tests que nous allons mettre en place utilisent plusieurs dizaines de téléphones Android, alors que nous ne disposons pas de ce nombre de cartes microSD. Dans un premier temps, nous simulons donc leur présence grâce à la classe SecureProcess. Cette classe n'existe que pour permettre la mise en place de l'application sous forme de *proof-of-concept* car elle compromet par nature la sécurité de l'ensemble du système. Cette solution n'est donc que provisoire.

4.5.4 Architecture des applications

Nous présentons maintenant l'architecture des applications que nous avons développées : une application de comptage qui préserve l'anonymat des nœuds du réseau en section 4.5.4.1 ainsi qu'une application de collecte de traces de voisinage en section 4.5.4.2.

4.5.4.1 Comptage

Nous présentons une version allégée de l'application qui permet le comptage des nœuds dans le réseau. Il s'agit de l'application modélisée section 4.3.4. Les éléments qui ne sont pas présentés ici concernent la gestion de l'interface graphique qui permet, outre l'affichage de la valeur du compteur, l'envoi de messages de type ping à la demande de l'utilisateur, ces éléments ne nous semblant pas pertinents pour la description de l'implémentation de l'application modélisée section 4.3.4.

```
public class MiMANCounter extends Activity implements
  ICallbackPrimitives, ...{

    /* Variable qui correspond au compteur */
    int counter = 0;

    /* Le middleware appelle la primitive seen */
    public void seen() {
      counter++;
```

```
    }

    /* Construction de l'application */
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      /* La classe MiMANClient fournit des methodes
       * relatives au lien application/middleware */
      MiMANClient.register(this);
    }

    /* Destruction de l'application */
    public void onDestroy() {
      MiMANClient.unregister(this);
      super.onDestroy();
    }
  }
```

L'application de comptage des voisins repose quant à elle sur les primitives `seeing`, `nextRound` et `lost` décrites en section 4.3.5 page 137. La gestion des communications avec l'*applet*, de la configuration de l'interface Wi-Fi et des communications *Over The Air* sont de la même manière déléguées au *middleware*.

4.5.4.2 Collecte de traces de voisinage

L'application de collecte de traces de voisinage fonctionne sur le même modèle, et grâce à l'utilisation des primitives `sendIdCounter` et `receiveIdRemoteCounter`.

```
public class Collector extends Activity implements
  ICallbackPrimitives, ...{

    /* Traces de voisinage */
    Vector<Trace> traces = new Vector<Trace>();

    /* Le middleware appelle la primitive receiveIdRemoteCounter */
    public void receiveIdRemoteCounter(int id, int c) {
      traces.add(new Trace(id,c));
    }

    /* Construction de l'application */
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      /* La classe MiMANClient fournit des methodes
       * relatives au lien application/middleware */
      MiMANClient.register(this);
    }
```

```
    /* Destruction de l'application */
    public void onDestroy() {
      MiMANClient.unregister(this);
      super.onDestroy();
    }
  }
```

Cependant, afin de simplifier l'expérimentation que nous allons mettre en place, nous avons créé une version spécifique de cette application de traces de voisinage comme partie intégrante du service MiMAN. De cette manière, dès le démarrage du terminal mobile, le service de collecte de traces de voisinage est automatiquement lancé.

Nous allons distribuer aux étudiants du Master SR (Systèmes et Réseaux) de l'université Bordeaux 1 adossé à l'équipe Muse du LaBRI dans laquelle se déroulent les travaux décrits dans cette thèse, des téléphones Android sur lesquels nous avons installé le *middleware* MiMAN et l'application de collecte de traces de voisinage. Ces traces nous permettrons de construire le graphe des communications entre les nœuds à travers le temps. Nous espérons en dégager des propriétés topologiques originales.

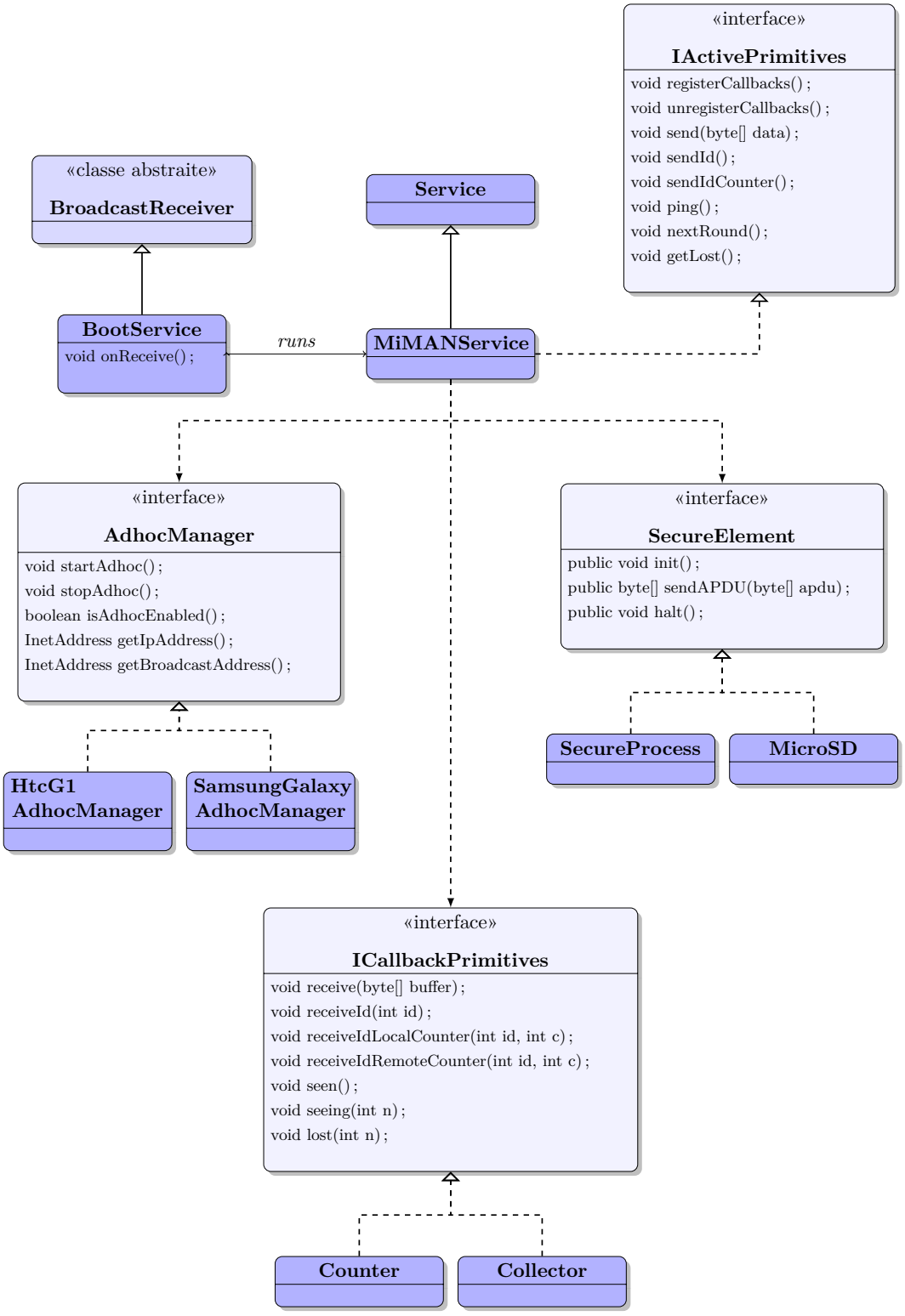


Fig. 4.67: Architecture simplifiée du *middleware*

Conclusion

De la multiplication et de la diversification des terminaux mobiles communicants ont émergé de nouveaux problèmes théoriques et un nouveau domaine de recherche qu'est l'étude des réseaux mobiles ad hoc. La possible interconnexion de tous les dispositifs sans fil fait apparaître de nouveaux services et de nouveaux problèmes à la fois théoriques et pratiques. La communauté scientifique a été très active dans ce domaine en proposant (i) des outils logiciels pour l'exploitation de ces nouveaux réseaux et (ii) des algèbres de processus pour leur modélisation.

Dans cette thèse, nous avons cherché à ne pas dissocier la modélisation des interactions entre les différentes entités qui composent un tel réseau des lois physiques qui régissent les interactions qui peuvent réellement exister. Le chapitre 1 nous a permis d'introduire le contexte précis de recherche dans lequel nous nous plaçons et que nous avons appelé iMANet (highly Mobile Ad hoc Network). Dans ce contexte, un grand nombre des suppositions classiquement présentes dans la littérature ne sont pas valables plus précisément, celles développées dans le chapitre 1), invalidées par les considérations physiques qui régissent les interactions entre les dispositifs sans fil. En effet, lors de communication par ondes radio, il est par exemple impossible de garantir que tous les nœuds possèdent la même portée. Le contexte d'étude iMANet est donc plus général que celui des réseaux mobiles ad hoc tels qu'on les considère dans le sens où nous faisons moins de suppositions sur les nœuds qui le composent. Autrement dit, les résultats obtenus dans un iMANet restent vraie dans un MANet classique, mais pas la réciproque. L'ensemble de ces travaux de thèse (à l'exception de l'annexe A) se place dans ce contexte de iMANet.

L'état de l'art des travaux logiciels et théoriques relatifs aux réseaux mobiles ad hoc réalisé dans le chapitre 2 met en évidence (i) la diversité des problèmes que les *middlewares* existants cherchent à résoudre et (ii) l'absence de formalisme permettant de modéliser les iMANets tels que définis dans le chapitre 1. Par ailleurs, nous attachons une grande importance à la sécurité dans le réseau et les *middlewares* étudiés, pour la plupart, ne s'y intéressent pas.

La contribution principale de cette thèse est la définition dans le chapitre 3 d'une algèbre de processus nommé CiMAN qui permet en particulier la modélisation des communications qui peuvent intervenir dans les iMANets. CiMAN est une algèbre de processus à plusieurs niveaux qui permet de définir des processus, des unités de calcul, des nœuds et des messages et qui supporte les interactions entre ces entités par échange de messages sous diverses formes. En CiMAN, les processus peuvent communiquer deux à deux de

manière synchrone et atomique (comme en CCS [Mil89]). Hébergés sur des unités de calcul différentes, il peuvent alors communiquer par une opération de broadcast inspirée de CBS [Pra95] qui modélise une communication sur un bus. Enfin, hébergés sur des unités de calcul elles mêmes placées sur des nœuds différents, ils peuvent alors communiquer grâce à une opération de *broadcast* non atomique, l'action d'émission provoquant la création d'une nouvelle entité de type message autonome en terme de mobilité.

Dans le chapitre 4, nous avons présenté un ensemble de primitives évoluées que nous avons modélisées avec CiMAN et qui composent le support d'exécution que nous avons développé. Il est constitué d'un *middleware* nommé MiMAN (*Middleware for highly Mobile Ad hoc Network*) et d'une *applet* nommée AiMAN (*Applet for highly Mobile Ad hoc Network*). Nous supposons que l'*applet* est exécutée dans un *Secure Element* et que celui-ci est infaillible, c'est à dire qu'il est impossible de lire ou d'écrire son contenu ou de lui faire exécuter des processus autres que l'*applet* que nous avons définie et qu'il embarque. Après avoir défini l'anonymat partiel et l'anonymat total dans un réseau, nous avons précisé les conséquences de l'utilisation de nos primitives sur la préservation de ces deux types d'anonymat. Enfin, nous avons présenté les choix technologiques que nous avons faits, le développement logiciel effectué et l'architecture de notre support d'exécution.

Par ces travaux, nous pensons avoir contribué à une meilleure compréhension des iMANets et des problématiques que soulève leur dynamique.

Directions de recherche

Perspectives issues du chapitre 1

La remise en question des hypothèses classiques du domaine des MANets dans le chapitre 1 a fait apparaître des caractéristiques intéressantes sur les liens de communication qui peuvent exister dans le réseau. En nous plaçant dans un environnement à deux dimensions et en supposant que les nœuds possèdent des portées radio uniformément réparties, nous avons pu montrer (cf. annexe A) que le ratio du nombre de liens de communication asymétriques sur le nombre de liens de communication symétriques vaut 3. Des résultats récents de simulation semblent indiquer que ce ratio vaut 4 dans un environnement en trois dimensions, et qu'il vaut plus généralement $n + 1$ dans un environnement à n dimensions. Il serait intéressant de prouver analytiquement ce résultat, et d'étudier les conséquences de la méthode de distribution des portées de nœuds, actuellement linéaire, sur le résultat obtenu.

Perspectives issues du chapitre 3

Concernant la mobilité des nœuds dans le chapitre 3, nous avons abouti à la construction d'un méta-graphe évolutif consistant en un graphe décrivant les évolutions possibles de l'ensemble des entités du réseau en fonction de la mobilité des nœuds et des messages qu'ils sont susceptibles d'émettre. Il serait intéressant d'étudier quelles propriétés sur la mobilité individuelle de chacun des nœuds de départ induisent des propriétés sur la mobilité des nœuds du système dans leur globalité. En d'autres termes, existe-t-il des propriétés

exprimables sur la mobilité des nœuds qui impliquent des propriétés sur le méta-graphe évolutif ?

Perspectives issues du chapitre 4

Dans le chapitre 4, nous présentons une application de collecte de traces de voisinage qui garantit l'anonymat partiel des nœuds du réseau : les identifiants des nœuds sont dévoilés au niveau du *middleware* et de l'application. Il pourrait être intéressant de faire en sorte que cette application préserve l'anonymat total des nœuds qui l'utilisent en ne dévoilant jamais l'identité des nœuds du réseau et en ne remontant les traces de voisinage que sous la forme chiffrée. Le déchiffrement des traces pourraient alors avoir lieu après leur collecte. La collecte de ces traces de voisinage amène la question de l'exploitation de celles-ci, de la création d'algorithmes qui permettent d'extraire des propriétés topologiques du graphe orienté qu'elles ont permis de créer, et qui représente la vie d'un vrai réseau, captée sur le terrain.

Annexe A

Proportion de liens asymétriques

Nous considérons que les liens de communication entre les nœuds sont asymétriques. Il est donc possible qu'un nœud A soit capable d'envoyer un message à un nœud B sans que le nœud B ne puisse répondre à A. Nous considérons donc que l'hypothèse H_4 (cf. section 1.1.4 page 8), très fréquemment adoptée dans la littérature, n'est pas réaliste. Il en découle en particulier qu'il est impossible pour un nœud de connaître la liste exacte de ses voisins, même si ceux-ci s'annoncent régulièrement. En effet, la mobilité des nœuds est supposée très importante et la communication entre les nœuds asynchrone. Ces deux caractéristiques impliquent que lors de la réception d'un message d'annonce d'un nœud, l'information peut déjà ne plus être valide. Nous faisons l'hypothèse qu'aucun lien de communication entre les nœuds n'est fiable ou persistant pendant une durée ϵ, aussi petit soit ϵ. Une conséquence importante est par exemple que le problème de consensus n'a pas de solution, et qu'il est donc impossible pour plusieurs nœuds de se synchroniser [FLP85].

Nous supposons donc qu'il est possible qu'un nœud ne soit capable que de communiquer avec un autre de manière unidirectionnelle, i.e. sans possibilité pour le nœud récepteur de répondre directement (mais nous n'interdisons pas la possibilité d'une communication bidirectionnelle). En travaillant sur cette hypothèse, nous avons constaté que le nombre de liaisons asymétriques d'un réseau est très important relativement au nombre de liens symétriques. Dans cette annexe, nous étudions mathématiquement cette proportion en supposant que les portées des nœuds sont différentes d'un nœud à l'autre. Nous montrons qu'elle n'est pas négligeable et qu'il est donc très intéressant de considérer les liens asymétriques lors de la création d'un algorithme ou d'une application pour les MANets. Ceci valide la pertinence de la suppression de l'hypothèse habituelle H_4 (cf. section 1.1.4 page 8) : la relation de voisinage est symétrique.

Afin de calculer formellement le nombre de liens asymétriques, nous supposons dans cette annexe que les hypothèses H_1 (le monde est plat) et H_2 (une zone de transmission radio est circulaire) présentées respectivement dans les sections 1.1.1 et 1.1.2 page 7 sont vérifiées. Ces suppositions sont contradictoires avec le chapitre 1, mais dans une première approche du problème elles nous permettent de simplifier le calcul.

Dans un travail futur, nous montrerons que ne pas faire ces hypothèses entraîne une augmentation de la proportion de liens asymétriques stricts. Bien que nous ne le prouvions pas mathématiquement ici, voici une explication intuitive concernant l'hypothèse H_2 (cf.

section 1.1.2 page 7). Le fait qu'il existe des trous dans la couverture d'un nœud entraîne la disparition de certains liens de communication. Si l'on considère que cette disparition est totalement aléatoire, alors elle provoque soit la disparition d'un lien asymétrique, soit la disparition d'un lien symétrique. Or, la suppression d'un lien asymétrique ne fait que diminuer le nombre de liens asymétriques alors que la suppression d'un lien symétrique a pour effet d'ajouter un lien asymétrique. Le nombre de liens asymétriques peut donc croître alors que le nombre de liens asymétriques ne peut que décroître lors de la suppression aléatoire d'un lien de communication. C'est pourquoi la proportion de liens asymétriques pourrait être encore plus importante sans la présence de l'hypothèse H_2 (cf. section 1.1.2 page 7).

De la même manière, de récents travaux [ANGP09] montrent l'impact théorique de la présence d'obstacles sur les portées radio des nœuds de l'environnement. Contrairement à ces travaux, nous considérons ici un environnement sans obstacle et en dimension 2. Nous nous intéresserons aux conséquences de la présence d'obstacles et d'une troisième dimension dans un travail futur.

Nous considérons une surface torique de superficie donnée N^2. Nous plaçons sur cette surface K dispositifs ($K > 0$) de manière aléatoire (chaque point de la surface ayant la même probabilité d'abriter un dispositif). Chacun de ces dispositifs possède une portée P choisi aléatoirement et uniformément dans l'intervalle $[A\,;B]$; $A \geq 0$. Nous sommes conscient que cette répartition des nœuds (uniforme dans un espace torique) a un impact important sur le résultat que nous obtenons. Dans un travail futur, nous nous intéresserons à des répartitions probabilistes plus en lien avec la réalité (même si le résultat alors obtenu ne pourra toujours être qu'une approximation de la réalité).

Nous calculons alors le pourcentage de liaisons unilatérales strictes parmi l'ensemble des liaisons existantes en fonction du minorant A et du majorant B de la portée.

Nombre de liaisons unilatérales strictes. Nous calculons tout d'abord le nombre moyen de liaisons unilatérales strictes de chaque nœud. Pour chaque couple de nœuds N et N' de portées respectives x et y ($x \leq y$) et séparés par une distance d, il existe une liaison unilatérale stricte entre N et N' si et seulement si N est à portée de N' sans que N' soit à portée de N ; c'est-à-dire lorsque $x < d \leq y$. En d'autres termes, la liaison unilatérale n'existe que lorsque le dispositif N se trouve sur la surface grise foncée de la figure A.69 (b). Nous appelons N_u le nombre moyen de liaisons unilatérales strictes par nœud (le détail concernant le calcul de N_u est donné page 167).

$$\begin{aligned} N_u &= \frac{\pi(K-1)}{N^2}\frac{1}{B-A}\int_A^B \frac{1}{(B-\alpha)(\alpha-A)}\int_\alpha^B\int_A^\alpha y^2 - x^2\,dx\,dy\,d\alpha \\ &= \frac{\pi(K-1)}{2N^2}(B^2-A^2) \end{aligned}$$

Nombre de liaisons bilatérales. Nous calculons maintenant le nombre moyen de liaisons bilatérales de chaque nœud. Pour chaque couple de dispositifs N, N' de portées respectives x et y ($x \leq y$), il existe une liaison bilatérale entre N et N' si et seulement si N

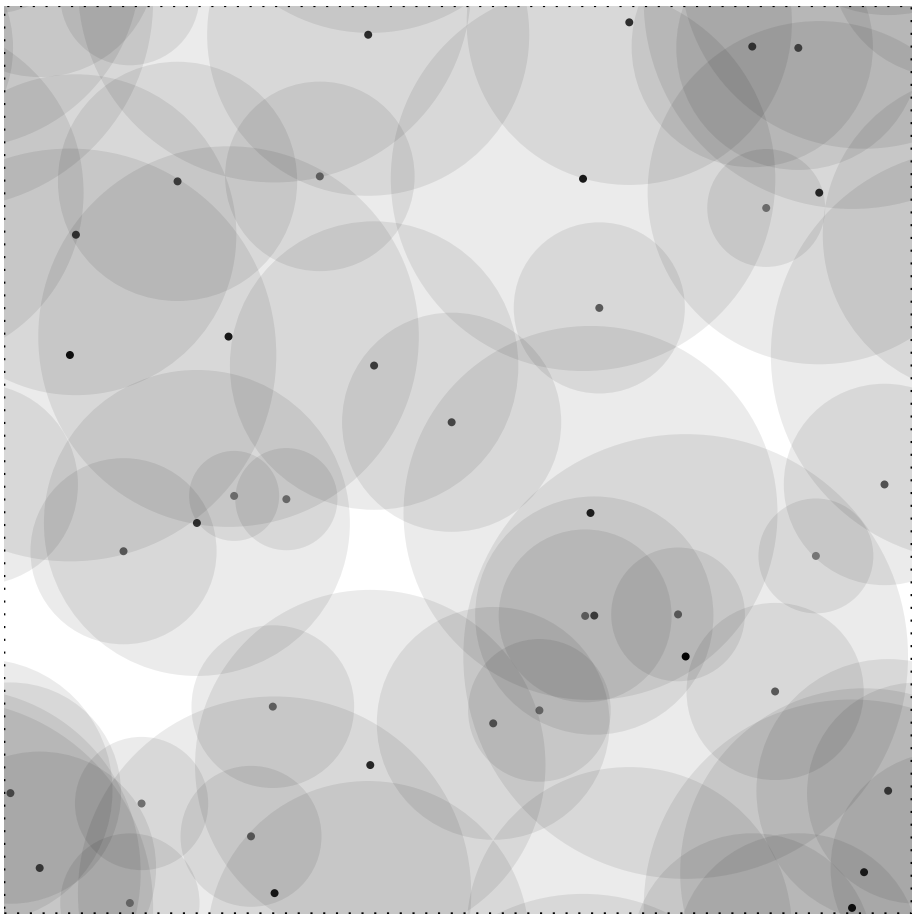
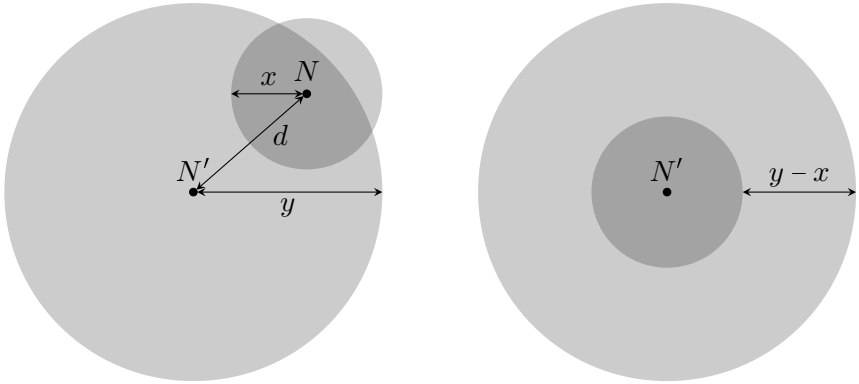


FIG. A.68: Le monde en 2D

est à portée de N' et N' est à portée de N. En d'autres termes la liaison bilatérale n'existe que lorsque le dispositif N se trouve sur la surface grise foncée de la figure A.70 (b). Nous appelons N_b le nombre moyen de liaisons bilatérales de chaque nœud (le détail concernant le calcul de N_b est page 168).

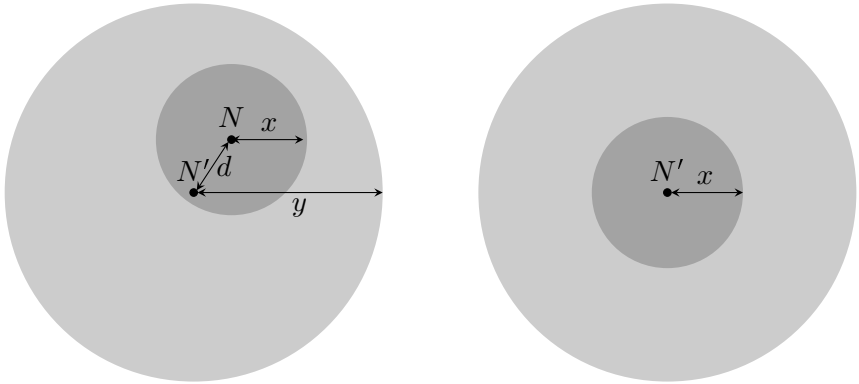
$$\begin{aligned} N_b &= \frac{\pi(K-1)}{2N^2}\frac{1}{B-A}\int_A^B x^2\,dx \\ &= \frac{\pi(K-1)}{6N^2}(A^2+AB+B^2) \end{aligned}$$



(a) N voisin unilatéral strict de N'

(b) Zone dans laquelle N doit se trouver pour être voisin unilatéral strict de N'

FIG. A.69: Liaison unilatérale stricte



(a) N voisin réciproque de N'

(b) Zone dans laquelle N doit se trouver pour que N et N' soient voisins réciproques

FIG. A.70: Liaison bilatérale

Pourcentage de liaisons unilatérales strictes. Nous calculons maintenant le pourcentage P de liaisons unilatérales strictes.

$$\begin{aligned} P &= 100\frac{N_u}{N_u + N_b} \\ &= 100\frac{B^2 - A^2}{B^2 - A^2 + \frac{1}{3}(A^2 + AB + B^2)} \end{aligned}$$

Nous pouvons remarquer que si $A = 0$ (la borne inférieure de la portée est nulle) alors P est constant quel que soit B et $P = 75\%$. Cela signifie que la densité n'a pas d'influence sur le pourcentage de liaisons asymétriques et que, dans ce contexte, 3/4 des liens de communication sont à sens unique.

En conclusion, il nous semble très important de ne pas supposer que tous les liens sont symétriques, autrement dit de ne pas utiliser que les liens symétriques. Prenons l'exemple d'un algorithme de diffusion d'information dans un réseau mobile ad hoc. Si l'algorithme de diffusion ne considère que les liens symétriques, c'est-à-dire n'envoie l'information qu'aux nœuds s'étant préalablement fait connaître comme voisin, alors 3/4 des liens de communication ne sont pas utilisés.

Nous donnons maintenant le détail du calcul concernant N_u.

$$
\begin{aligned}
N_u &= \frac{\pi.(K-1)}{N^2}\frac{1}{B-A}\int_A^B \frac{1}{(B-\alpha)(\alpha-A)}\int_\alpha^B\int_A^\alpha y^2-x^2\,dx\,dy\,d\alpha \\
&= \frac{\pi(K-1)}{N^2}\frac{1}{B-A}\int_A^B \frac{1}{(B-\alpha)(\alpha-A)}\int_\alpha^B [y^2x-\frac{x^3}{3}]_A^\alpha\,dy\,d\alpha \\
&= \frac{\pi(K-1)}{N^2}\frac{1}{B-A}\int_A^B \frac{1}{(B-\alpha)(\alpha-A)}\int_\alpha^B y^2\alpha-\frac{\alpha^3}{3}-y^2A+\frac{A^3}{3}\,dy\,d\alpha \\
&= \frac{\pi(K-1)}{N^2}\frac{1}{B-A}\int_A^B \frac{1}{(B-\alpha)(\alpha-A)}\int_\alpha^B (\alpha-A)y^2-\frac{1}{3}(\alpha^3-A^3)\,dy\,d\alpha \\
&= \frac{\pi(K-1)}{N^2}\frac{1}{B-A}\int_A^B \frac{1}{(B-\alpha)(\alpha-A)}[\frac{1}{3}(\alpha-A)y^3-\frac{1}{3}(\alpha^3-A^3)y]_\alpha^B\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B (\alpha-A)B^3-(\alpha^3-A^3)B-(\alpha-A)\alpha^3+(\alpha^3-A)\alpha\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B (\alpha-A)(B^3-\alpha^3)-(\alpha^3-A^3)(B-\alpha)\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B (\alpha-A)(B-\alpha)(B^2+B\alpha+\alpha^2)-(\alpha-A)(B-\alpha)(\alpha^2+A\alpha+A^2)\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B B^2+B\alpha+\alpha^2-\alpha^2-A\alpha-A^2\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B B^2-A^2+\alpha(B-A)\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B (B-A)(B+A)+\alpha(B-A)\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\frac{1}{B-A}\int_A^B (B-A)(B+A+\alpha)\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}\int_A^B \alpha+B+A\,d\alpha \\
&= \frac{\pi(K-1)}{3N^2}[\frac{\alpha^2}{2}+\alpha(B+A)]_A^B \\
&= \frac{\pi(K-1)}{3N^2}(\frac{B^2}{2}+B(B+A)-\frac{A^2}{2}-A(B+A))
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{\pi(K-1)}{3N^2}((B+A)\frac{B-A}{2}+(B+A)(B-A)) \\
&= \frac{\pi(K-1)}{3N^2}((B+A)(\frac{B-A}{2}+B-A)) \\
&= \frac{\pi(K-1)}{3N^2}((B+A)\frac{3}{2}(B-A)) \\
&= \frac{\pi(K-1)}{2N^2}(B^2-A^2)
\end{aligned}
$$

Nous présentons aussi les détails concernant le calcul de N_b.

$$
\begin{aligned}
N_b &= \frac{\pi(K-1)}{2N^2}\frac{1}{B-A}\int_A^B x^2\,dx \\
&= \frac{\pi(K-1)}{2N^2}\frac{1}{B-A}[\frac{x^3}{3}]_A^B \\
&= \frac{\pi(K-1)}{2N^2}\frac{1}{B-A}(\frac{B^3}{3}-\frac{A^3}{3}) \\
&= \frac{\pi(K-1)}{2N^2}\frac{1}{B-A}((B-A)(A^2+AB+B^2)) \\
&= \frac{\pi(K-1)}{6N^2}(A^2+AB+B^2)
\end{aligned}
$$

Bibliographie

[AHP02] Akhil Arora, Carl Haywood, and Kuldip Singh Pabla. JXTA for J2ME - Extending the Reach of Wireless With JXTA Technology. Technical report, Sun Microsystem, March 2002.

[AMSSHS08] Arezu Moghadam, Suman Srinivasan, and Henning Schulzrinne. 7DS - A Modular Platform to Develop Mobile Disruption-tolerant Applications. In *Second IEEE Conference and Exhibition on Next Generation Mobile Applications, Services, and Technologies (NGMAST 2008)*, September 2008.

[Ang80] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *STOC '80 : Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 82–93, New York, NY, USA, 1980. ACM.

[ANGP09] C. Aboue-Nze, F. Guinand, and Y. Pigné. Impact of obstacles on the degree of mobile ad hoc connection graphs. In *Proceedings of the International Conference on Networking and Services (ICNS'2009). 21-25 April, 2009. Valencia, Spain*, pages 332–337, 2009.

[Ash62] W. Ross Ashby. Principles of the self-organizing system. In Heinz v. Foerster and George W. Zopf, editors, *Principles of Self-Organization : Transactions of the University of Illinois Symposium*, pages 255–278. Pergamon, London, 1962.

[AVW+06] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, and J. Vuori. Chedar : peer-to-peer middleware. *Parallel and Distributed Processing Symposium, International*, 0 :252, 2006.

[Bar96] Thomas James Barber. *BodyLAN : A low power communications system.* PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, USA, 1996.

[BCM03] Dario Bottazzi, Antonio Corradi, and Rebecca Montanari. AGAPE : a Location-aware Group Membership Middleware for Pervasive Computing Environments. In *ISCC '03 : Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, page 1185, Washington, DC, USA, 2003. IEEE Computer Society.

[BCM04] Dario Bottazzi, Antonio Corradi, and Rebecca Montanari. Context-Aware Group Communication in Mobile Ad-Hoc Networks. In *WWIC*, pages 38–47, 2004.

[BCM05] Dario Bottazzi, Antonio Corradi, and Rebecca Montanari. Context-Awareness for Impromptu Collaboration in MANETs. In *NCA '04 : Proceedings of the Network Computing and Applications, Third IEEE International Symposium*, pages 339–342, Washington, DC, USA, 2005. IEEE Computer Society.

[BCMT04] Mario Bisignano, Andrea Calvagna, Giuseppe Di Modica, and Orazio Tomarchio. Design and development of a Jxta middleware for mobile ad-hoc networks, 2004.

[BD07] Eleonora Borgia and Franca Delmastro. Effects of unstable links on aodv performance in real testbeds. *EURASIP J. Wirel. Commun. Netw.*, 2007(1) :32–32, 2007.

[BMJ+98] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98 : Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM.

[BMT04] Mario Bisignano, Giuseppe Di Modica, and Orazio Tomarchio. An infrastructure-less peer-to-peer framework for mobile handheld devices, 2004.

[BMT05] Mario Bisignano, Giuseppe Di Modica, and Orazio Tomarchio. JMobiPeer : A Middleware for Mobile Peer-to-Peer Computing in MANETs. In *International Conference on Distributed Computing Systems Workshops*, volume 08, pages 785–791, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[BS99] Mark Billinghurst and Thad Starner. Wearable devices : New ways to manage information. *Computer*, 32(1) :57–64, 1999.

[CAMCM04] Shiva Chetan, Jalal Al-Muhtadi, Roy Campbell, and M. Dennis Mickunas. A Middleware for Enabling Personal Ubiquitous Spaces. In *UbiSys '04 : System Support for Ubiquitous Computing Workshop at Sixth Annual Conference on Ubiquitous Computing (UbiComp 2004)*, Nottingham, England, September 2004.

[CAMCM05] Shiva Chetan, Jalah Al-Muhtadi, Roy Campbell, and M Dennis Mickunas. Mobile Gaia : A Middleware for Ad-hoc Pervasive Computing. In *IEEE Consumer Communications and Networking Conference*, Las Vegas, 2005.

[Cas02] Claude Castelluccia. Statistically unique and cryptographically verifiable (SUCV) identifiers and addresses. In *In Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS*, 2002.

[Cas07] A. Casteigts. *Contribution à l'Algorithmique Distribuée dans les Réseaux Mobiles Ad Hoc - Calculs Locaux et Réétiquetages de Graphes Dynamiques*. PhD thesis, University of Bordeaux, 2007.

[CBD02] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile*

Computing (WCMC) : Special issue on Mobile Ad Hoc Networking : Research, Trends and Applications, 2(5) :483–502, 2002.

[CC06] Arnaud Casteigts and Serge Chaumette. Dynamicity aware graph relabeling systems and the constraint based synchronization : A unifying approach to deal with dynamic networks. In Xiuzhen Cheng, Wei Li, and Taieb Znati, editors, *Wireless Algorithms, Systems, and Applications*, volume 4138 of *Lecture Notes in Computer Science*, pages 688–697. Springer Berlin / Heidelberg, 2006.

[CCF09] A. Casteigts, S. Chaumette, and A. Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *Proc. of 16th Intl. Conference on Structural Information and Communication Complexity (SIROCCO'09)*, volume 5869 of *Lecture Notes in Computer Science*, pages 126–140, Piran, Slovenia, May 2009. Springer-Verlag.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo. CARISMA : Context-Aware Reflective Middleware System for Mobile Applications. In *IEEE Transactions on Software Engineering*, 2003.

[CFS+01] Scott Camazine, Nigel R. Franks, James Sneyd, Eric Bonabeau, Jean-Louis Deneubourg, and Guy Theraula. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA, 2001.

[CJ03] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[CP01] Gianpaolo Cugola and Gian Pietro Picco. PeerWare : Core Middleware Support for Peer-to-Peer and Mobile Systems. Technical report, Politecnico di Milano, 2001.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[DCY00] Samir R. Das, Robert Castaneda, and Jiangtao Yan. Simulation-based performance evaluation of routing protocols for mobile ad hoc networks. *ACM/Baltzer Mobile Networks and Applications (MONET) Journal*, 5(3) :179–189, 2000.

[DPHU+08] Isabelle Demeure, Guilhem Paroux, Javier Hernando-Ureta, Amir R. Khakpour, and Julien Nowalczyk. A power-aware middleware for mobile ad-hoc networks. In *NOTERE '08 : Proceedings of the 8th international conference on New technologies in distributed systems*, pages 1–7, New York, NY, USA, 2008. ACM.

[DPS+94] A. Demers, K. Petersen, M. Spreitzer, D. Ferry, M. Theimer, and B. Welch. The Bayou Architecture : Support for Data Sharing among Mobile Users. In K. Petersen, editor, *Proc. Workshop on Mobile Computing Systems and Applications*, pages 2–7, 1994.

[EM01] Cristian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *IPDPS'01 : Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 149, Washington, DC, USA, 2001. IEEE Computer Society.

[FC06] Stephen Farrell and V. Cahill. *Delay- and Disruption-Tolerant Networking*. Artech House, Inc., Norwood, MA, USA, 2006.

[FGLS04] H. Frey, D. Görgen, J. Lehnert, and P. Sturm. SELMA : A middleware platform for self-organizing distributed applications in mobile multihop ad-hoc networks. In *PERCOM*, 2004.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2) :374–382, 1985.

[Fri29] Karinthy Frigyes. *Everything is different*. 1929.

[GC04] Javier Gomez and Andrew T. Campell. A case for variable-range transmission power control in wireless multihop networks. In *in Proceedings of INFOCOM'04*, pages 1425–1436, 2004.

[Gel85] David Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1) :80–112, 1985.

[GM07] Frédéric Guidec and Yves Mahéo. Opportunistic Content-Based Dissemination in Disconnected Mobile Ad Hoc Networks. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM '07*, volume 0, pages 49–54, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[God07] Jens Chr. Godskesen. A calculus for mobile ad hoc networks. Technical report, IT University, May 2007.

[GR04] F. Guidec and H. Roussain. Asynchronous Document Dissemination in Dynamic Ad Hoc Networks. In *Second International Symposium on Parallel and Distributed Processing and Applications (ISPA'04)*, December 2004.

[HBSF] M. Hapner, R. Burridge, R. Sharma, and J. Fialli. Java Message Sservice specfification version 1.1 Sun Microsystem, inc. [Online]. Available : `http://java.sun.com/products/jms/`.

[Her03] K. Herrmann. MESHMdl - A Middleware for Self-Organization in Ad hoc Networks. In *Proceedings of the 1st International Workshop on Mobile Distributed Computing (MDC'03)*, May 2003.

[Hog05] Luc Hogie. The Madhoc Simulator. Technical report, Le Havre University, 2005.

[HPJR03] Radu Handorean, Jamie Payton, Christine Julien, and Gruia-Catalin Roman. Coordination Middleware Supporting Rapid Deployment of Ad Hoc Mobile Systems. In *ICDCSW '03 : Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 362, Washington, DC, USA, 2003. IEEE Computer Society.

[HR02] Radu Handorean and Gruia-Catalin Roman. Service Provision in Ad Hoc Networks. In *COORDINATION '02 : Proceedings of the 5th International Conference on Coordination Models and Languages*, pages 207–219, London, UK, 2002. Springer-Verlag.

[IHB+10a] Makoto Ikeda, Masahiro Hiyama, Leonard Barolli, Fatos Xhafa, and Arjan Durresi. Mobility effects on the performance of mobile ad hoc networks. *Complex, Intelligent and Software Intensive Systems, International Conference*, 0 :230–237, 2010.

[IHB+10b] Makoto Ikeda, Masahiro Hiyama, Leonard Barolli, Fatos Xhafa, Arjan Durresi, and Makoto Takizawa. Mobility effects of wireless multi-hop networks in indoor scenarios. *Advanced Information Networking and Applications, International Conference on*, 0 :495–502, 2010.

[JR02] Christine Julien and Gruia-Catalin Roman. Egocentric context-aware programming in ad hoc mobile environments. In *SIGSOFT '02/FSE-10 : Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, pages 21–30, New York, NY, USA, 2002. ACM.

[JR04] Christine Julien and Gruia-Catalin Roman. Active Coordination in Ad Hoc Networks. In *Proceedings of the 6th International Conference on Coordination Models and Languages*, pages 199–215, February 2004.

[JR06] Christine Julien and Gruia-Catalin Roman. EgoSpaces : Facilitating Rapid Development of Context-Aware Mobile Applications. *IEEE Trans. Softw. Eng.*, 32(5) :281–298, 2006.

[KNE03] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. Technical report, Dartmouth College, July 2003.

[KNG+04a] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. Technical Report TR2004-507, Dept. of Computer Science, Dartmouth College, June 2004.

[KNG+04b] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 78–82. ACM Press, October 2004.

[KSS+01] Gerd Kortuem, Jay Schneider, Jay Schneider, Dustin Preuitt, Thaddeus G. C. Thompson, Stephen Fickas, and Zary Segall. When Peer-to-Peer comes Face-to-Face : Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *P2P '01 : Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.

[KWVV05] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile Chedar - A Peer-to-Peer Middleware for Mobile Devices. In *PERCOMW '05 :*

Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, pages 86–90, Washington, DC, USA, 2005. IEEE Computer Society.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565, 1978.

[lFcRH04] Chien liang Fok, Gruia catalin Roman, and Gregory Hackmann. A Lightweight Coordination Middleware for Mobile Computing. In *In Proceedings of the 6th international conference on coordination models and languages*, pages 135–151. Springer-Verlag, 2004.

[LSG02] Nicolas Le Sommer and Frédéric Guidec. JAMUS : Java Accommodation of Mobile Untrusted Software. In *4th Nord EurOpen/Usenix Conference (NordU 2002)*, pages 38–48, Helsinki, Finland, February 2002. Multiprint, Helsinki.

[MC02] René Meier and Vinny Cahill. STEAM : Event-Based Middleware for Wireless Ad Hoc Network. In *ICDCSW '02 : Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 639–644, Washington, DC, USA, 2002. IEEE Computer Society.

[MC03a] R. Meier and V. Cahill. Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications. In *Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03)*. Springer-Verlag Heidelberg, 2003.

[MC03b] R. Meier and V. Cahill. Location-Aware Event-Based Middleware : A Paradigm for Collaborative Applications ?, 2003.

[MCE01] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. An XML-based Middleware for Peer-to-Peer computing. In *P2P '01 : Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, page 69, Washington, DC, USA, 2001. IEEE Computer Society.

[MCZE02] Cecilia Mascolo, Licia Capra, Stefanos Zachariadis, and Wolfgang Emmerich. XMIDDLE : A Data-Sharing Middleware for Mobile Computing. *Wirel. Pers. Commun.*, 21(1) :77–103, 2002.

[Mer07] Massimo Merro. An observational theory for mobile ad hoc networks. *Electron. Notes Theor. Comput. Sci.*, 173 :275–293, 2007.

[MGS09] N. Monmarché, F. Guinand, and P. Siarry, editors. *Fourmis artificielles. Tome 2 : nouvelles directions vers une intelligence collective.* Traités IC2. Hermès, octobre 2009. ISBN : 978-2-7462-2349-3.

[Mil82] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989. ISBN 0-13-114984-9 (Hard) 0-13-115007-3 (Pbk).

[Mil99] Robin Milner. *Communicating and mobile systems : the π-calculus.* Cambridge University Press, New York, NY, USA, 1999.

[MMH05] Mirco Musolesi, Cecilia Mascolo, and Stephen Hailes. EMMA : Epidemic Messaging Middleware for Ad hoc networks. *Personal Ubiquitous Comput.*, 10(1) :28–36, 2005.

[MP04] Amy L. Murphy and Gian Pietro Picco. Using Coordination Middleware for Location-Aware Computing : A Lime Case Study. In *Proceedings of the 6th International Conference on Coordination Models and Languages (COORD04)*, pages 263–278. Springer Lecture Notes on Computer Science vol. 2949, February 2004.

[MPR03] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME : A Coordination Middleware Supporting Mobility of Hosts and Agents. Technical report, Dept. of Computer Science, University of Rochester, NY, USA, April 2003.

[MPR06] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME : A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3) :279–328, 2006.

[MR98] Peter J. McCann and Gruia-Catalin Roman. Compositional programming abstractions for mobile computing. *IEEE Trans. Softw. Eng.*, 24(2) :97–110, 1998.

[MS06] Nicola Mezzetti and Davide Sangiorgi. Towards a calculus for wireless systems. *Electr. Notes Theor. Comput. Sci.*, 158 :331–353, 2006.

[Mur00] Amy Lynn Murphy. *Enabling the rapid development of dependable applications in the mobile environment.* PhD thesis, Washington University, St. Louis, MO, USA, 2000. Director-Gruia-Catalin Roman.

[MVZ05] Marco Mamei, Matteo Vasirani, and Franco Zambonelli. Self-Organizing Spatial Shapes in Mobile Particles : The TOTA Approach. In *Engineering Self-Organising Systems*, pages 138–153. Springer Berlin / Heidelberg, May 2005.

[MZ03] Marco Mamei and Franco Zambonelli. Location-Based and Content-Based Information Access in Mobile Peer-to-Peer Computing : The TOTA Approach. In *AP2PC*, pages 162–173, 2003.

[MZ04a] Marco Mamei and Franco Zambonelli. Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In *PERCOM '04 : Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 263, Washington, DC, USA, 2004. IEEE Computer Society.

[MZ04b] Marco Mamei and Franco Zambonelli. Self-Maintained Distributed Tuples for Field-based Coordination in Dynamic Networks. In *SAC '04 : Proceedings of the 2004 ACM symposium on Applied computing*, pages 479–486, New York, NY, USA, 2004. ACM.

[MZ05] Marco Mamei and Franco Zambonelli. Programming stigmergic coordination with the TOTA middleware. In *AAMAS '05 : Proceedings of the fourth international joint conference on Autonomous agents*

and multiagent systems, pages 415–422, New York, NY, USA, 2005. ACM.

[MZ09] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications : The tota approach. *ACM Trans. Softw. Eng. Methodol.*, 18(4) :1–56, 2009.

[MZL03] Marco Mamei, Franco Zambonelli, and Letizia Leonardi. Tuples on the air : A middleware for context-aware computing in dynamic networks. In *ICDCSW '03 : Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 342, Washington, DC, USA, 2003. IEEE Computer Society.

[NG09] C. Aboue Nze and F. Guinand. A survey of connectivity in mobile ad hoc networks. In *Proceedings of the 3rd International Conference on Complex Systems and Applications*, pages xx–yy, Le Havre, june 29th - july 2nd 2009.

[NH06] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2) :203 – 227, 2006. Automated Reasoning for Security Protocol Analysis, Automated Reasoning for Security Protocol Analysis.

[OG02] Scott Oaks and Li Gong. *Jxta in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[PAD+05] Thomas Plagemann, Jon Andersson, Ovidiu Drugan, Vera Goebel, Carsten Griwodz, Paal Halvorsen, Ellen Halvorsen, Ellen Munthe-Kaas, Matija Puzar, Norun Sanderson, and Katrine Stemland Skjelsvik. Middleware Services For Information Sharing In Mobile Ad-Hoc Networks, 2005.

[PB02] Gian Pietro Picco and Marco L. Buschini. Exploiting Transiently Shared Tuple Spaces for Location Transparent Code Mobility. In *COORDINATION '02 : Proceedings of the 5th International Conference on Coordination Models and Languages*, pages 258–273, London, UK, 2002. Springer-Verlag.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[PDR08] Guilhem Paroux, Isabelle Demeure, and Laurent Reynaud. An energy-aware middleware for collaboration on small scale manets. In *ASNS'08 : Proceedings of the Autonomous and Spontaneous Networks Symposium*, Paris, France, 2008.

[PGGH03] Thomas Plagemann, Vera Goebel, Carsten Griwodz, and Paal Halvorsen. Towards Middleware Services for Mobile Ad-Hoc Network Applications. In *FTDCS '03 : Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, page 249, Washington, DC, USA, 2003. IEEE Computer Society.

[PMR99] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME : Linda Meets Mobility. In *ICSE '99 : Proceedings of the 21st*

international conference on Software engineering, pages 368–377, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[PMR00] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Developing Mobile Computing Applications with LIME. In *ICSE '00 : Proceedings of the 22nd international conference on Software engineering*, pages 766–769, New York, NY, USA, 2000. ACM.

[PMR01] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME : A Middleware for Physical and Logical Mobility. In *ICDCS '01 : Proceedings of the The 21st International Conference on Distributed Computing Systems*, page 524, Washington, DC, USA, 2001. IEEE Computer Society.

[PR99] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[Pra93] K. V. S. Prasad. A calculus of value broadcasts. In *PARLE '93 : Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, pages 391–402, London, UK, 1993. Springer-Verlag.

[Pra95] K. V. S. Prasad. A calculus of broadcasting systems. In *ESOP '94 : Selected papers of ESOP '94, the 5th European symposium on Programming*, pages 285–327, Amsterdam, The Netherlands, The Netherlands, 1995. Elsevier Science Publishers B. V.

[Rap01] Theodore Rappaport. *Wireless Communications : Principles and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[RG06] Ashikur Rahman and Pawel Gburzynski. Hidden problems with the hidden node problem. In *in Proceedings of 23rd Biennial Symposium on Communications*, pages 270–273, 2006.

[SAAPL08] Marc Sanchez-Artigas, Marcel Arrufat, Gerard Paris, and Pedro Garcia Lopez. SCOMET : A Middleware for Collaborative Working Environments in MANETs. *Ubiquitous Computing and Communication Journal*, 3, March 2008.

[SCB+06] Aline Senart, Raymond Cunningham, Mélanie Bouroche, Neil O'Connor, Vinny Reynolds, and Vinny Cahill. MoCoA : Customisable Middleware for Context-Aware Mobile Applications. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (2)*, volume 4276 of *Lecture Notes in Computer Science*, pages 1722–1738. Springer, 2006.

[SDL+02] Kimaya Sanzgiri, Bridget Dahill, Brian N. Levine, Clay Shields, and Elizabeth M. Belding Royer. A secure routing protocol for ad hoc networks. *Network Protocols, IEEE International Conference on*, 0 :78+, 2002.

[SFFS04] Peter Sturm, Daniel Fischer, Volker Fusenig, and Thomas Scherer. The .NET CF Implementation of GecGo - A middleware for multihop ad-hoc networks -, 2004.

[SFGL04] Peter Sturm, Hannes Frey, Daniel Görgen, and Johannes Lehnert. Supporting Smart Applications in Multihop Ad-Hoc Networks - The GecGo Middleware. In *in Proc. of 8th International Conference on Knowledge-Based Intelligent Information & Engineering Systems KES 04, ser. LNCS*, pages 718–726, 2004.

[SMLB06] Miguel S. Silva, Francisco Martins, Luis Lopes, and Joao Barros. A calculus for sensor networks. *CoRR*, abs/cs/0612093, 2006. informal publication.

[SR04] Nicolas Le Sommer and Hervé Roussain. JASON : an Open Platform for Discovering, Delivering and Hosting Applications in Mobile Ad Hoc Networks, 2004.

[SRS08] Anu Singh, C. Ramakrishnan, and Scott Smolka. A process calculus for mobile ad hoc networks. *Coordination Models and Languages*, pages 296–314, 2008.

[SSAMHS09] Suman Srinivasan, Arezu Moghadam, and Henning Schulzrinne. BonAHA : Service Discovery Framework for Mobile Ad-Hoc Applications. In *IEEE Consumer Communications & Networking Conference 2009 (CCNC'09)*, January 2009.

[SSAMSGHHGS07] Suman Srinivasan, Arezu Moghadam, Se Gi Hong, and Henning G Schulzrinne. 7DS - Node Cooperation and Information Exchange in Mostly Disconnected Networks. In *IEEE International Conference on Communications (ICC)*, June 2007.

[TBCM03] O. Tomarchio, M. Bisignano, A. Calvagna, and G. Di Modica. Expeerience : A JXTA Middleware for Mobile Ad-Hoc Networks. In *In Third Int. Conference on Peer-to-Peer Computing (P2P2003)*, 2003.

[TTP+95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15)*, Copper Mountain Resort, Colorado, 1995.

[Var92] Hal R. Varian. *Microeconomic Analysis, Third Edition*. W. W. Norton & Company, 3rd edition, February 1992.

[Var01] Andras Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, pages 319–324, Prague, Czech Republic, June 2001. SCS – European Publishing House.

[VPP06] Maria Grazia Vigliotti, Iain Phillips, and Catuscia Palamidessi. Expressiveness via Leader Election Problems. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *4th International Symposium on Formal Methods for Components and Objects (FMCO) Postproceedings of the 4th International Symposium on Formal Methods for Components and Objects*, volume 4111 of *Lecture Notes in Computer Science*, pages 172–194, Amsterdam Pays-Bas, 2006. Springer.

[ZCME02] Stefanos Zachariadis, Licia Capra, Cecilia Mascolo, and Wolfgang Emmerich. XMIDDLE : information sharing middleware for a mobile environment. In *ICSE '02 : Proceedings of the 24th International Conference on Software Engineering*, pages 712–712, New York, NY, USA, 2002. ACM.

[Zim96] T. G. Zimmerman. Personal area networks : near-field intrabody communication. *IBM Syst. J.*, 35(3-4) :609–617, 1996.

Webographie

[Ait] Aithne Project. http://www.dsg.cs.tcd.ie/aithne.

[And] Android. http://www.android.com/.

[Apa] Apache. http://www.apache.org/licenses/LICENSE-2.0.html.

[App] Apple computer's Bonjour. [Online]. Available : http://developer.apple.com/networking/bonjour/.

[Blua] Bluetooth Special Interest Group. http://www.bluetooth.com/.

[Blub] Bluetooth tracking. http://www.bluetoothtracking.com.

[Cya] cyanogen$^{(mod)}$. http://www.cyanogenmod.com/.

[FF02] Kevin Fall and Stephen Farrell, 2002. Internet Research Task Force, http://www.dtnrg.org/.

[IEE] IEEE 802.15. http://www.ieee802.org/15/.

[IPN] InterPlanetary Internet Special Interest Group. http://www.ipnsig.org/.

[JXT01] SUN Microsystems Project JXTA, 2001. http://www.jxta.org.

[Mul] Multicast DNS. [Online]. Available : http://files.multicastdns.org/.

[NFC] NFC Forum. http://www.nfc-forum.org/.

[NPD] Consumer market recsearch - npd. http://www.npd.com/.

[PCS] PCSC Workgroup. http://www.pcscworkgroup.com/.

[SFA] Secure Element Evaluation Kit for the Android platform. http://code.google.com/p/seek-for-android.

[WIF] Wi-Fi Alliance. http://www.wi-fi.org/.

[WPA] Linux WPA/WPA2/IEEE 802.1X Supplicant. http://hostap.epitest.fi/wpa_supplicant/.

[Zer] Zeroconf working group. [Online]. Available : http://www.zeroconf.org/.

Modèle de calcul, primitives, et applications de référence, pour le domaine des réseaux ad hoc fortement mobiles

Mots-clés : réseaux ad hoc fortement mobiles, volatilité, iMANet, algèbre de processus, CiMAN, graphes dynamiques, sécurité

Résumé : Les réseaux ad hoc dynamiques qui évoluent de manière non planifiée et imprévisible sont souvent étudiés en faisant l'hypothèse d'une composition et d'une topologie qui évoluent peu et relativement lentement. Il est alors possible de proposer dans ce contexte faiblement mobile des mécanismes (comme par exemple du routage, des infrastructures PKI, *etc.*) qui permettent aux applications conçues pour les réseaux statiques de continuer à fonctionner. Les travaux présentés dans cette thèse sont au contraire centrés sur les réseaux ad hoc fortement dynamiques (iMANets). Les nœuds qui les constituent sont extrêmement mobiles et volatils, ce qui engendre des modifications incessantes et rapides de topologie. Les contributions principales de cette thèse sont (i) la définition d'une algèbre nommée CiMAN (*Calculus for highly Mobile Ad hoc Networks*) qui permet de modéliser les processus communicants dans ces réseaux ad hoc fortement mobiles, (ii) l'utilisation de cette algèbre pour prouver la correction d'algorithmes dédiés à ces réseaux, et (iii) un *middleware* et des applications de référence adaptés à ce contexte.

Process calculus, programming interface and reference applications, for highly mobile ad hoc networks

Keywords : highly mobile ad hoc networks, volatility, iMANet, process calculus, CiMAN, dynamic graphs, security

Abstract : Mobile ad hoc networks that evolve in an unplanned and unpredictable manner are often studied assuming that their composition and their topology evolve relatively slowly. In this context of weak mobility, it is then possible to propose mechanisms (such as routing, Public Key Infrastructure, *etc.*) which make the application designed for a static context still operational. At the opposite, the work presented in this thesis focuses on highly mobile ad hoc networks (iMANets). The nodes of these networks are extremely mobile, bringing ceaseless and fast changes in the network topology. The main contributions of this thesis are (i) the definition of an algebra called CiMAN (Calculus for highly Mobile Ad hoc Networks) which makes it possible to model communicating processes in these highly mobile ad hoc networks, (ii) the use of this algebra to prove the correctness of algorithms dedicated to these networks, and (iii) a *middleware* and reference applications specifically designed for this context.