THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université de Toulouse III – Paul Sabatier*
Spécialité *: Systèmes Informatiques Critiques*

Présentée et soutenue par *Minh Duc Nguyen*
Le *19 novembre 2009*

METHODOLOGIE DE TEST DE SYSTEMES MOBILES : UNE APPROCHE BASEE SUR LES SCENARIOS

Jury

Ana Rosa Cavalli, Professeur à TELECOM & Management SudParis (rapporteur)
Thierry Gayraud, Professeur à l'Université de Toulouse III-Paul Sabatier (président)
Karim Guennoun, Professeur Assistant à l'Ecole Hassania des Travaux Publics, Maroc
David Powell, Directeur de recherche CNRS
Hans-Peter Schwefel, Professeur à l'Université d'Åalborg, Danemark
Françoise Simonot-Lion, Professeur à l'Institut National Polytechnique de Lorraine (rapporteur)
Nicolas Rivière, Maître de conférences à l'Université de Toulouse III-Paul Sabatier
Hélène Waeselynck, Chargée de recherche CNRS

Ecole Doctorale : *Ecole Doctorale Systèmes de Toulouse*
Unité de recherche : *LAAS-CNRS*
Directeurs de thèse : *David Powell et Hélène Waeselynck*

Avant-propos

Les travaux de thèse présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je remercie les directeurs successifs du LAAS-CNRS, Messieurs Malik Ghallab et Raja Chatila, pour m'avoir accueilli dans ce laboratoire. Je remercie également Monsieur Jean Arlat et Madame Karama Kanoun, responsables successifs du groupe de recherche Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF), pour m'avoir permis de réaliser mes travaux au sein de ce groupe.

J'exprime ma profonde reconnaissance à mes encadrants, dont j'ai pu apprécier les qualités tant scientifiques qu'humaines : Monsieur David Powell, Directeur de recherche au LAAS-CNRS, qui a assumé la direction administrative de ma thèse ; Madame Hélène Waeselynck, Chargée de recherche au LAAS-CNRS et Monsieur Nicolas Rivière, Maître de conférences à l'Université de Toulouse III-Paul Sabatier, qui ont assuré la direction scientifique de mes travaux. Leur rigueur intellectuelle, leur très grande compétence, leurs encouragements et leur gentillesse ont été déterminants pour ces travaux.

Je remercie les personnes qui m'ont fait l'honneur de participer à mon jury de thèse :

- Madame Ana Rosa Cavalli, Professeur à TELECOM & Management SudParis
- Monsieur Thierry Gayraud, Professeur à l'Université de Toulouse III-Paul Sabatier
- Monsieur Karim Guennoun, Professeur Assistant à l'Ecole Hassania des Travaux Publics, Maroc
- Monsieur David Powell, Directeur de recherche CNRS
- Monsieur Hans-Peter Schwefel, Professeur à l'Université d'Åalborg, Danemark
- Madame Françoise Simonot-Lion, Professeur à l'Institut National Polytechnique de Lorraine
- Monsieur Nicolas Rivière, Maître de conférences à l'Université de Toulouse III-Paul Sabatier
- Madame Hélène Waeselynck, Chargée de recherche CNRS.

Mesdames Ana Rosa Cavalli et Françoise Simonot-Lion ont accepté de consacrer une partie de leur temps précieux pour juger mon travail en tant que rapporteurs. Je leur témoigne tout mon respect et ma reconnaissance.

Ces travaux ont été partiellement financés par le projet européen HIDENETS (*Highly Dependable IP-Based Networks and Services*) et le réseau d'excellence européen ReSIST (*Resilience for Survivability in IST*). J'exprime mes remerciements aux responsables de ces deux projets, Monsieur Hans-Peter Schwefel et Monsieur Jean-Claude Laprie.

Mes remerciements s'adressent également à Messieurs Khalil Drira, Karim Guennoun et Ismael Bouassida, qui nous ont rendu disponible leur outil et ont passé du temps à répondre à mes questions.

Je tiens à remercier Zoltan Micskei, doctorant à l'Université Technologique de Budapest (BUTE - *Budapest University of Technology and Economics*), pour nos collaborations et nos discussions enrichissantes pendant ses séjours au LAAS-CNRS.

Mes remerciements vont également à l'ensemble des membres du groupe TSF, permanents, doctorants et stagiaires, avec lesquels j'ai partagé ces années de travail. Ils m'ont encouragé et assisté de leur mieux dès mes premiers pas au LAAS-CNRS. J'ai eu l'occasion de me lancer dans une ambiance chaleureuse, et d'avoir des échanges avec des personnes que j'ai beaucoup appréciées, Mohammed Kaaniche, Yves Crouzet, Agnan De Bonneval, Jérémie Guiochet, Marc-Olivier Killijian, Vincent Nicomette, Matthieu Roy, Ludovic Courtès, Eric Alata, Thomas Robert, Thomas Pareaud, Etienne Baudin, Ossama Hamouda, Youssef Laarouchi, Amine Baina, Eric Lacombe, Géraldine Vache, Lu Caroline, Manel Sghairi, Chu Hoang Nam, ainsi que mes autres amis au LAAS.

Je dois, aussi, beaucoup à mes parents, dont les bénédictions m'ont suivi tout au long. Je leur exprime toute mon admiration, mon affection et ma gratitude.

Je remercie Bui Thi Quynh qui, à sa manière, m'a énormément aidé. Je lui exprime mon admiration pour son courage et sa patience et ma gratitude pour son soutien indéfectible.

Je remercie profondément tous ceux qui ont pu m'apporter courage et gentillesse : ma famille et tous mes amis qui seraient trop nombreux à citer mais que je n'oublie pas.

Enfin, je dédie cette thèse à ma petite fille.

Table de matières

Liste des Figures

Liste des Tableaux

INTRODUCTION GENERALE

Contexte

Ces dernières années ont été marquées par des avancées majeures dans le domaine de l'informatique mobile et en particulier dans le cadre de la mobilité physique (en anglais, *mobile computing*). D'une manière générale, les systèmes informatiques mobiles incluent des dispositifs mobiles (assistants personnels, ordinateurs portables…) qui se déplacent tout en étant connectés aux réseaux de communication au moyen de liens sans fil (IEEE 802.11, Bluetooth…). La technologie sans fil a permis l'émergence de nouveaux types d'applications. D'un point de vue théorique et technologique, les applications mobiles sont plus complexes que leurs homologues fixes car elles impliquent non seulement des aspects d'applications distribuées (par exemple, le parallélisme, la communication, la latence) mais aussi des caractéristiques propres aux systèmes mobiles : les connexions et déconnexions fréquentes des nœuds, la gestion du réseau ad hoc et la dépendance vis-à-vis du contexte. Ces nouveautés des systèmes mobiles posent de nouveaux défis pour les méthodologies de validation et vérification d'applications.

L'objectif de notre travail est de développer des solutions pour le test d'applications et de services dans les systèmes mobiles. Ce travail est conduit dans le cadre du Réseau d'Excellence ReSIST (NoE IST 026764) et du projet HIDENETS (IST 26979) du sixième programme-cadre de la Communauté européenne, Technologies de la société de l'information.

Face aux nouvelles spécificités des systèmes mobiles, le test pose plusieurs nouveaux défis. Parmi ceux-ci, nous trouvons qu'un problème subtil est la manière de représenter les scénarios d'interactions entre les entités dans les systèmes mobiles. Dans les systèmes distribués classiques, les descriptions de scénarios jouent un rôle important dans le développement des logiciels. Elles peuvent être utilisées dans plusieurs phases. Pour le test, elles peuvent servir pour : la capture d'exigences, la spécification d'objectifs de test (c'est-à-dire, les interactions à couvrir par les cas de test), la conception de cas de test ou l'analyse de traces d'exécution. Mais les langages classiques de scénarios n'ont pas été conçus pour décrire des interactions dans un contexte mobile. Le transfert vers les nouveaux types de systèmes, et notamment les systèmes mobiles, nécessite d'étendre les descriptions de scénario pour couvrir leurs spécificités : le changement de topologie, la communication avec le voisinage, la dépendance vis-à-vis du contexte, etc.

La contribution de cette thèse porte sur la solution de ce problème. Il s'agira, dans un premier temps, d'appliquer une approche de test basée sur une description graphique des

scénarios pour les systèmes mobiles. Dans ce cadre, nous introduisons des extensions pour tenir compte des spécificités des systèmes mobiles. Ces extensions portent essentiellement sur les relations spatiales entre les nœuds et la communication par diffusion dans le voisinage. Nous discuterons des traitements automatiques des descriptions de scénarios. Nous nous focalisons sur les problèmes d'appariement de graphes du fait de la prise en compte explicite des relations spatiales entre nœuds.

La suite de notre contribution est l'introduction de *GraphSeq*, un outil que nous avons développé pour traiter l'aspect spatial dans les descriptions de scénarios. La nouveauté de *GraphSeq* se situe dans la déduction d'appariement sur les séquences de graphes pour traiter les configurations spatiales. L'outil prend en compte des caractéristiques importantes d'un système mobile : le changement de topologie du système, les créations et les suppressions des nœuds.

L'application de *GraphSeq* pour analyser des traces de mobilité de système contribue à vérifier si une trace couvre un objectif ou une exigence de test, ou encore à instancier les configurations spatiales pour un cas de test.

Organisation du document

Ce mémoire comporte cinq chapitres.

Le premier chapitre introduit la définition des systèmes informatiques mobiles et les concepts de base du test des logiciels dédiés à ce type de systèmes. Nous allons aborder les systèmes mobiles et les travaux de test associés. Nous présentons des problèmes potentiels rencontrés en essayant de résoudre les questions fondamentales du test dans les systèmes mobiles : la détermination du niveau de test, la sélection de test, le problème de l'oracle, la plate-forme de test.

Dans le chapitre 2, nous nous sommes penchés sur une étude de cas trouvée dans la littérature : un protocole d'appartenance de groupe dans le domaine des réseaux ad hoc. Nous l'avons analysée et testée. L'étude de cas nous aide à illustrer et à mieux comprendre les défis posés par le test des systèmes mobiles. Ensuite, les enseignements tirés nous orientent vers la définition d'un cadre de test basé sur des scénarios et des traitements automatisés pour analyser et implémenter des scénarios sur une plate-forme de test.

Le troisième chapitre présente les principes généraux d'une approche de test basée sur des descriptions de scénario. Dans une telle approche, les descriptions de scénario sont utilisées pour spécifier des exigences fonctionnelles, des objectifs de test, des cas de test et des traces d'exécution. Afin d'appliquer cette approche dans le cadre de systèmes mobiles, il est nécessaire d'étendre les descriptions de scénario pour couvrir les spécificités des systèmes mobiles. Nous introduisons des extensions qui portent essentiellement sur les relations spatiales entre les nœuds et la communication par diffusion dans le voisinage. L'utilité de ces extensions sera illustrée par plusieurs exemples de scénarios dans un contexte mobile. Ensuite, nous discuterons des traitements qui peuvent être effectués à partir des descriptions de scénarios. Nous mettrons l'accent sur les problèmes d'appariement de graphes qui apparaissent dans ces traitements, du fait de la prise en compte explicite des relations spatiales entre les nœuds.

Le quatrième chapitre présente les algorithmes d'appariement de graphes de *GraphSeq*, l'outil développé pour traiter la vue spatiale des scénarios d'interaction dans les systèmes

mobiles. La vue spatiale représente explicitement la dynamicité de la topologie des nœuds dans le système avec des connexions et déconnexions induites par la mobilité, ainsi que des créations et des disparitions dynamiques des nœuds, qui sont les caractéristiques importantes des systèmes mobiles.

Le cinquième chapitre présente la validation et les expérimentations de *GraphSeq*. *GraphSeq* est validé par des exemples illustratifs et par un outil automatisé. Ensuite, nous montrons l'utilité de *GraphSeq* pour extraire des motifs de mobilité au travers des résultats des expérimentations.

Enfin, nous concluons ce mémoire et présentons les perspectives principales de nos travaux.

CHAPITRE 1

CADRE DES TRAVAUX ET ETAT DE L'ART

1. Introduction

Ce premier chapitre introduit la définition des systèmes informatiques mobiles et les concepts de base du test des logiciels dédiés à ce type de systèmes. Nous allons aborder ici les systèmes mobiles et les travaux de test associés.

Dans un premier temps (paragraphe 2), nous allons introduire les systèmes informatiques mobiles, ainsi que leurs nouvelles caractéristiques par rapport aux systèmes répartis classiques. Ces caractéristiques peuvent avoir une influence sur les techniques traditionnelles du test. Ensuite, nous allons présenter les travaux sur la modélisation des systèmes mobiles (paragraphe 3), qui ont des influences sur les méthodes de test.

Nous poursuivons, dans le paragraphe 4, par une présentation sur le test. Nous allons nous intéresser plus particulièrement à sa mise en œuvre dans le cadre des systèmes mobiles : la détermination de niveau de test (§4.1), la sélection de test (§4.2) et l'oracle (§4.3). Nous mettrons en parallèle ce qui existe dans le cadre des applications traditionnelles, et les travaux correspondants pour les systèmes mobiles. Ensuite, nous aborderons les langages de scénarios d'interactions (§4.4), qui sont souvent utilisés dans les activités liées au test des systèmes répartis.

Dans le paragraphe 5, nous discuterons d'un problème technologique : la plate-forme expérimentale qui permet de réaliser les tests. Nous allons argumenter sur l'intérêt d'utiliser des moyens de simulation/émulation pour la plate-forme expérimentale et l'illustrer par des exemples existants dans la littérature.

Les travaux de ce chapitre ont fait l'objet d'un rapport [Nguyen 06] publié dans le cadre du réseau d'excellence européen ReSIST (*Resilience for Survivability in IST*).

2. Introduction aux systèmes mobiles

2.1. Définition et classes de systèmes mobiles

Les avancées du domaine des sciences et technologies de l'information et de la communication "sans fil" ont conduit au développement de nouveaux types de systèmes, qui

sont dédiés à l'informatique mobile. Conformément à la classification dans [Lyytinen et Yoo 02], nous avons quatre catégories de systèmes, qui dépendent des niveaux de mobilité et d'instrumentation de l'environnement, comme cela est montré dans la Figure 1.

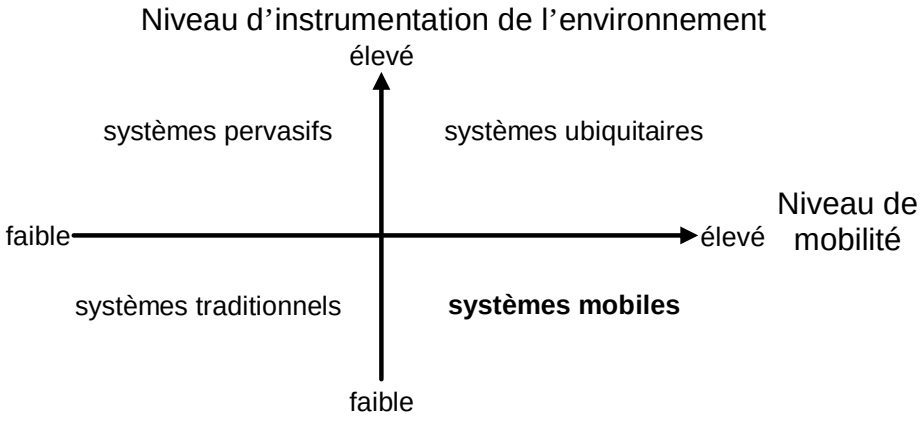


Figure 1. Catégories de systèmes distribués/Dimension de systèmes ubiquitaires

Les avancées technologiques permettent de produire de nouveaux types d'équipements qui peuvent être massivement enfouis et communiquer entre eux au moyen de liaisons sans fil. Le concept de base des systèmes *pervasifs* est de créer un environnement communicant dit "intelligent" à l'aide d'une forte intégration de composants tels que des capteurs, des actionneurs, des badges RFID dans des objets de la vie courante tels que les meubles, les véhicules, les vêtements, *etc*. Implicitement, il faut que ces dispositifs soient capables d'obtenir des informations de l'environnement dans lequel ils sont *enfouis* et de les utiliser pour construire dynamiquement une architecture/support de traitement informatique. La complexité engendrée par ce type de système vient de la densité des dispositifs qui composent l'environnement. Par exemple, un système pervasif dans une pièce d'un bâtiment peut être composé de centaines ou de milliers de dispositifs. Notons que, dans un système pervasif, les objets sont en général fixe mais la topologie qu'ils forment peut changer suite à, par exemple, des défaillances de nœuds qui la composent.

Les systèmes informatiques *mobiles* (en anglais, *mobile computing*) impliquent une capacité importante des dispositifs à pouvoir se déplacer. D'une manière générale, les systèmes informatiques mobiles incluent des dispositifs tels que des assistants personnels numériques (PDA), des ordinateurs portables, des voitures intelligentes, *etc*. qui se déplacent tout en étant connectés aux réseaux au moyen de liens sans fil (par exemple, IEEE 802.11 a/b/g/n, Bluetooth, ZigBee). Dans ce type de systèmes, les dispositifs exécutent des traitements indépendamment et communiquent entre eux tout en se déplaçant. En pratique, on ne considère qu'une faible densité d'équipements, *i.e.* quelques dizaines de dispositifs dans une zone géographique. La complexité engendrée par ce type de systèmes se situe au niveau des traitements collaboratifs entre les nœuds car nous sommes en présence de changements topologiques très fréquents.

La prise en compte de ces deux dimensions, les niveaux de mobilité et d'instrumentation de l'environnement, mène à une nouvelle catégorie de systèmes distribués : les systèmes *ubiquitaires*. Ce type de systèmes intègre les caractéristiques des systèmes *mobiles* et *pervasifs*. Le but est d'utiliser une multitude de dispositifs peu coûteux, intégrés et distribués

dans les équipements (fixes et mobiles) de notre environnement quotidien, pour effectuer des opérations courantes en utilisant des moyens de communication sans fil.

Dans la suite de ce mémoire, nous nous intéressons aux « systèmes mobiles » ou « systèmes informatiques mobiles ».

Nous pouvons distinguer deux modes de fonctionnement pour les systèmes mobiles. Le premier est le mode *infrastructure*. Dans ce mode, il y a des serveurs fixes dits « d'infrastructure » qui interagissent et prennent en charge la gestion des communications entre les dispositifs dans leur zone géographique de couverture. Le deuxième est le mode *ad hoc* où les dispositifs communiquent en l'absence de serveur d'infrastructure.

La mobilité abordée dans le cadre des systèmes mobiles est la mobilité *physique* des dispositifs. Notons qu'il existe un autre type de mobilité, la mobilité *logique* (en anglais *mobile computation*), qui concerne la mobilité du code et qui est définie dans [Fuggetta et al. 98] comme la capacité à changer dynamiquement les liens entre les fragments de code et le lieu (dispositif) où ces fragments sont exécutés. Ce type de mobilité n'est pas considéré dans nos travaux.

2.2. Exemples d'application avec mobilité physique

Les applications mobiles ayant recours aux réseaux de communication sans fil couvrent un très large spectre, incluant par exemple, les applications militaires, les applications véhicule-à-véhicule, ou des applications dans le domaine de la santé concernant la surveillance à distance de personnes. Certains projets ont porté sur le développement d'applications, telles que des guides de tourisme avec des dispositifs mobiles [GUIDE], [CYBERGUIDE], des applications véhicule-à-véhicule [COMeSafety], [PReVENT], [NOW], [Coopers], [HIDENETS], des services de sauvegarde [SUMO], [MoSAIC], des garanties de continuité de connexion pendant le déplacement d'un utilisateur [OpenMobileIS], etc.

A titre illustratif, prenons deux exemples de système mobile.

Le premier est une application véhicule-à-véhicule du domaine automobile dont les communications se font en mode *ad hoc* : la boîte noire répartie. Elle a été développée dans le cadre du projet européen HIDENETS (*Highly Dependable ip-based Networks and Services*). Le principe de cette boîte noire est identique à celle d'un avion. Elle enregistre à intervalle régulier des informations datées relatives à la vitesse, au freinage, à la position du conducteur, à l'environnement du véhicule, etc. sous forme de données. Dans cette application, les données sont répliquées temporairement sur des véhicules voisins du même réseau ad hoc qui sont croisés sur les routes et indéfiniment sur des serveurs dans le cas d'un accès à une infrastructure fixe. En cas d'accident d'un véhicule, les données pourront être récupérées à partir des autres véhicules et des serveurs.

Un autre exemple est une application qui s'inspire des guides touristiques et qui a été développée au cours d'un projet [GUIDE]. Elle utilise un serveur d'infrastructure. Un touriste qui visite un monument est équipé d'un assistant personnel numérique à son entrée sur le site. En se promenant dans les différents lieux du monument, il est guidé à l'aide des serveurs présents dans ces zones et peut obtenir certaines informations auxquelles il s'intéresse par interaction. Ces serveurs sont associés à des dispositifs spécifiques répartis sur le site qui détectent la présence des touristes et permettent ainsi de leur fournir les informations touristiques relatives à leur position.

2.3. Caractéristiques des systèmes mobiles

D'un point de vue théorique et technologique, les applications mobiles sont plus complexes que leurs homologues fixes car elles impliquent non seulement des aspects d'applications distribuées (par exemple, le parallélisme, la communication, la latence) mais aussi des caractéristiques propres aux systèmes mobiles. Les systèmes informatiques mobiles, et en particulier les systèmes mobiles ad hoc, se distinguent des systèmes distribués "traditionnels" par les aspects suivants : la dynamicité de la structure du système, la communication par diffusion, la dépendance vis-à-vis du contexte.

2.3.1. La dynamicité de la structure du système

Le nombre de nœuds dans un système mobile n'est pas fixe : il varie avec le temps. Cela est dû aux opérations (abstraites) dynamiques de création, suspension, disparition des nœuds (voir la Figure 2.a). C'est la conséquence du fait que les utilisateurs peuvent, entre autres, allumer/éteindre les dispositifs, les changer de mode d'opération pour réduire les coûts de connexion, économiser la batterie ou éviter des ruptures inopinées de connexions. En plus de cela, la liberté de mouvement des nœuds modifie constamment la connectivité car ils sont à même de joindre ou quitter le système de manière spontanée : la topologie du système n'est pas stable. Cependant, les changements topologiques peuvent être contraints par un modèle de mobilité : par exemple, si on veut étudier le comportement d'un système mobile composé de véhicules automobiles communicants, il faut considérer qu'ils se déplacent sur une route dans une ou deux directions à une vitesse limitée.

2.3.2. La communication avec les nœuds inconnus dans le voisinage local

Dans un réseau mobile ad hoc, une communication « naturelle » est la communication par diffusion dans le voisinage. En général, elle est utilisée comme une brique de base pour la découverte du voisinage dans les applications et les services mobiles, par exemple, le service de découverte de nœuds pour les protocoles d'appartenance de groupe ou la découverte de routes dans les protocoles de routage. Dans ce type de communication, un nœud émet un message à son voisinage et tous les nœuds qui sont dans sa portée de transmission peuvent recevoir le message (voir la Figure 2.b). Comme la topologie du système n'est pas connue, l'émetteur ne connaît pas *a priori* le nombre et l'identité des récepteurs potentiels. N'importe qui à portée de transmission de l'émetteur peut recevoir et traiter le message.

2.3.3. La dépendance du contexte

Dans un système, chaque nœud devrait avoir une définition explicite du contexte, des politiques pour le mettre à jour et réagir à ses changements. Le contexte inclut tout attribut approprié et détectable d'un dispositif, de ses interactions avec les autres dispositifs et de son environnement proche à tout instant (voir la Figure 2.c). Par exemple, le contexte peut être des informations sur les dispositifs eux-mêmes comme la mémoire, des informations récoltées au moyen de capteurs physiques comme la localisation, le temps, la vitesse de déplacement, ou encore des informations sur l'environnement opérationnel comme les paramètres du réseau, la bande passante, la latence, la topologie de connectivité, etc. En conséquence, du fait de la mobilité, le contexte est constamment en évolution. Il est donc important que les applications mobiles s'aperçoivent de ces changements et s'y adaptent.

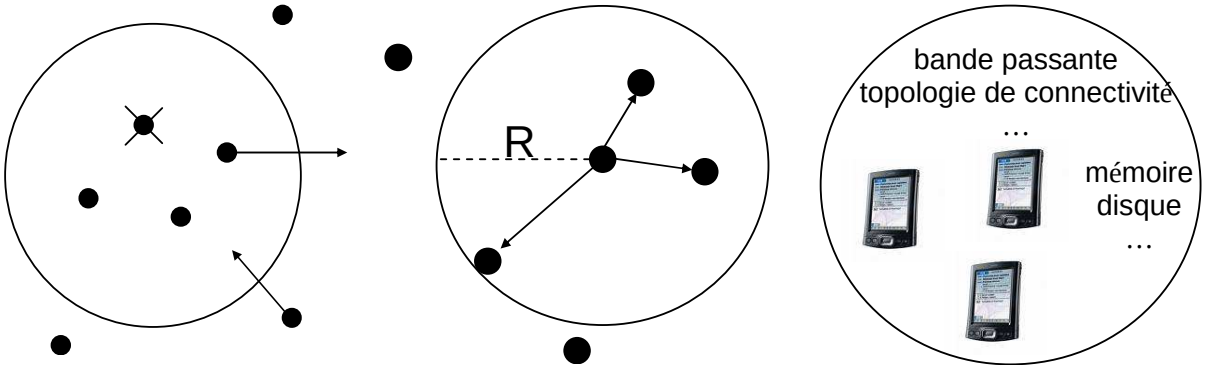


Figure 2. Caractéristiques des systèmes mobiles

Ces caractéristiques des systèmes mobiles posent de nouveaux défis pour le développement des applications mobiles en général, et pour la modélisation en particulier, sujet qui sera abordé dans le paragraphe suivant.

3. Modélisation de systèmes mobiles

La modélisation de systèmes mobiles doit prendre en compte non seulement des aspects comportementaux comme dans les systèmes classiques, mais aussi des aspects topologiques et contextuels. Divers travaux ont été effectués concernant la modélisation des systèmes mobiles. Ils concernent : la représentation de l'architecture de systèmes, les modèles de mobilité, les formalismes comportementaux incluant la mobilité et/ou la modélisation de contexte.

3.1. Représentation de l'architecture du système

Si on ne considère que la structure dynamique d'un système mobile, on pourrait argumenter que les graphes et les algorithmes de graphes constituent un cadre convenable de modélisation. Cependant, de tels formalismes ne capturent pas les aspects comportementaux des applications. A l'origine, les formalismes de graphes ont été largement utilisés pour spécifier des architectures logicielles dynamiques [Le Métayer 98] et ils peuvent être appliqués aussi pour modéliser la topologie d'un système mobile. Par exemple, dans [Nickerson 05], les graphes sont utilisés pour abstraire les relations spatiales et les distances de communication entre les nœuds mobiles. Les arêtes montrent une liaison physique (connecté/déconnecté), ainsi que la latence de communication (le temps pour qu'un message soit délivré). L'application de ces approches est discutée dans des situations mobiles, comme la mobilité dans un environnement réseau ad hoc, par exemple pour définir un algorithme pour la planification des routes.

Les algorithmes de transformation de graphes peuvent être appliqués pour spécifier l'évolution du contexte d'un système mobile. Le travail dans [Lei et Zhang 05] applique les théories des graphes *conceptuels* [Baget et Mugnier 02] pour modéliser les contextes mobiles. Un graphe conceptuel est un graphe étiqueté qui contient deux classes de sommets : une pour représenter les objets contextuels et l'autre pour les relations entre les objets. Des règles de transformation de graphe sont définies pour ajouter/supprimer des sommets dans un graphe et sont utilisées pour indiquer comment le contexte du système évolue. Les auteurs ont défini

quatre règles à appliquer dans un système mobile : insertion, suppression, activation et désactivation des services. Ces règles ont été utilisées dans l'implémentation d'un prototype de guide touristique portable de musée.

D'autres langages graphiques permettent de décrire l'architecture d'un système mobile. Le langage graphique *CML (Context Modelling Language)*, présenté dans [Henricksen et Indulska 06], permet aux développeurs de spécifier et d'explorer des exigences contextuelles. Ce langage fournit des primitives pour définir les types des informations contextuelles, leurs classifications, les méta-données quantitatives et les dépendances entre les types des informations. Un exemple simple est fourni dans la Figure 3. Le premier diagramme indique que l'entité *Person* peut être localisée alternativement dans plusieurs éléments *Location*. Le deuxième diagramme indique que *Person* participe, au maximum, à une activité à un instant donné.

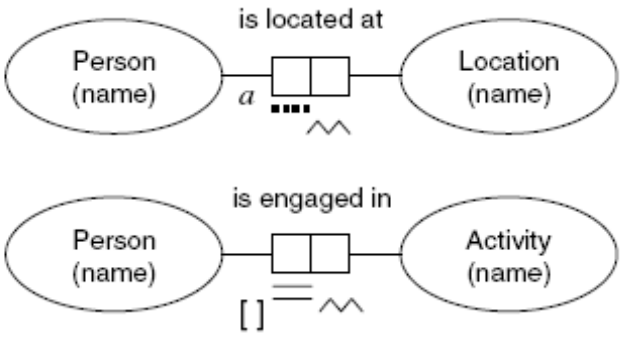


Figure 3. Un exemple d'utilisation de CML

Le langage UML constitue aussi un cadre approprié pour modéliser l'architecture d'applications dépendantes du contexte de part sa structure générique et ses différents diagrammes. Un exemple l'illustre dans [Sheng et Benatallah 05]. Ce travail se focalise sur un méta-modèle pour modéliser les Services Web en modifiant les méta-classes de UML.

3.2. Modèles de mobilité

En plus de leur utilisation pour modéliser l'architecture d'un système, les graphes peuvent être utilisés pour modéliser la mobilité des nœuds. Par exemple, dans [Tian et al. 02], les auteurs utilisent les graphes pour représenter des contraintes sur les déplacements qui sont imposées par les infrastructures. Les sommets du graphe représentent des lieux que les utilisateurs peuvent visiter, par exemple, les monuments d'un centre ville. Les arêtes représentent les connexions entre ces lieux, par exemple, les moyens de transport ou les rues. Le déplacement d'un nœud suit généralement le chemin le plus court entre deux sommets.

Cependant, l'application des formalismes de graphes aux modèles de mobilité est parfois trop abstraite et devient complexe si on veut détailler certains paramètres du mouvement d'un nœud tels que la position, la vitesse, etc. Afin d'avoir une vue cohérente entre les différents paramètres qui caractérisent la mobilité d'un nœud, des modèles spécifiques ont été conçus. Leur utilisation principale se situe dans le domaine de l'évaluation de protocoles.

Les modèles de mobilité ont pour objectif de décrire des configurations de mouvements des dispositifs/nœuds mobiles. De plus, ils montrent comment la localisation, la vitesse et l'accélération évoluent au cours du déplacement des nœuds. Il est donc nécessaire que les modèles émulent le plus précisément possible les configurations réelles. Dans les travaux de

recherche sur les protocoles de communication, des modèles de mobilité ont été proposés. Ils peuvent être classés dans quatre catégories [Bai et al. 03] : les modèles aléatoires, les modèles avec dépendances temporelles, les modèles avec dépendances spatiales et les modèles avec restrictions géographiques.

Concernant les modèles aléatoires, un exemple simple est *Random Waypoint Mobility* [Broch et al. 98]. Les nœuds se déplacent librement, sans aucune restriction. La destination, la vitesse et la direction sont choisies aléatoirement et indépendamment des autres nœuds.

Pour les modèles avec des dépendances temporelles, comme *Gauss-Markov Mobility* [Liang et Haas 99] ou *Smooth Random Mobility* [Bettstetter 01], des contraintes pour la vitesse, l'accélération et le taux de changement de direction sont introduites. La vitesse d'un nœud dépend donc de la vitesse précédente et par conséquent, les vitesses d'un nœud aux différents instants sont corrélées.

Dans les modèles avec des dépendances spatiales, *Reference Point Group Mobility* [Hong et al. 99] par exemple, la mobilité d'un nœud peut être influencée par les paramètres de ses voisins, ou de son leader, ce qui mène à une corrélation spatiale.

Dans certaines applications, nous avons pu constater que le mouvement d'un nœud est contraint par des restrictions géographiques, par exemple, le déplacement de véhicules sur une autoroute ou celui de personnes dans un bâtiment. Des travaux récents tiennent compte de ces caractéristiques et intègrent des chemins et des obstacles dans les modèles de mobilité comme dans *Freeway Mobility*, *Manhattan Mobility* [Bai et al. 03].

3.3. Les formalismes comportementaux incluant la mobilité

Les travaux présentés dans les paragraphes 3.1 et 3.2 ne traitent que de l'aspect topologique des systèmes, et de l'évolution de cette topologie. Afin de concevoir des applications mobiles, il faut pouvoir exprimer leur aspect comportemental. Dans la littérature, l'utilisation de certains formalismes traditionnels pour exprimer le comportement des applications a été étendue pour prendre en compte la mobilité. Nous présentons ici les formalismes les plus connus avec les *mobile ambients*, les extensions d'UML, Mobicharts et MobileUNITY.

Les mobile ambients. Les algèbres de processus qui possèdent une notion explicite de localisation, sont capables de couvrir à la fois les aspects comportementaux et structurels. Nous pouvons citer des travaux tels que pi-calculus [Milner et al. 92], join-calculus [Fournet et Gonthier 96], LLinda[De Nicolas et al. 97], Obliq [Boudol 92], spi-calculus [Abadi et Gordon 97]. Dans ce domaine, le calcul des *ambients*, « *ambient calculus* » [Cardelli et Gordon 00], est probablement le plus connu. Par définition, un *ambient* est une place bornée, « une boîte », où des calculs s'exécutent. Les ambients peuvent être emboîtés, ce qui conduit à une hiérarchie des localisations. Cette hiérarchie est modifiable par l'action de processus qui peuvent créer, détruire ou déplacer les ambients. Les auteurs ont défini des primitives pour cela, comme les primitives d'entrée et de sortie d'un ambient. La Figure 4 illustre l'effet de la primitive d'entrée *in*, en représentant graphiquement les ambients par des boîtes. La primitive *in* indique que l'ambient ***n*** entre dans l'ambient ***m***. L'ambient ***n*** change donc de niveau hiérarchique pour devenir un sous-ambient de ***m***. Ensuite, au sein de ***n***, l'exécution se poursuit avec le processus ***P|Q***.

Figure 4. Exemple de mobilité d'entrée d'une boîte dans Mobile Ambient

Le calcul des ambients possède une sémantique formelle. De plus, des propriétés déclaratives peuvent être exprimées en utilisant une logique avec des modalités spatiales et temporelles [Cardelli et Gordon 00 b]. En pratique, le calcul des ambients est plus conçu pour des études théoriques que pour spécifier des systèmes. Il offre un ensemble minimal de primitives de base ce qui rend l'encodage d'une application réelle relativement lourd. Le calcul des ambients peut plutôt être utilisé comme un cadre de base pour des formalismes de plus haut niveau. Il est toutefois important de noter que la vue sous-jacente de la mobilité n'est pas appropriée à toutes les applications. Comme indiqué précédemment, il se focalise sur l'entrée et la sortie de domaines « administratifs » qui sont organisés de manière hiérarchique. Cette vue est adéquate pour exprimer soit la mobilité logique, soit la mobilité physique depuis un point d'accès de l'infrastructure à un autre. Cependant, elle ne nous paraît pas adaptée pour prendre en compte la dynamique d'un réseau ad-hoc.

Le principe des ambients a fortement inspiré les auteurs travaillant sur la définition de formalismes de haut niveau utilisables par des ingénieurs.

Les extensions d'UML. Les extensions d'UML proposées dans [Baumeister *et al*. 03] et [Grassi *et al*. 04] ont une définition stéréotypée de la localisation qui ressemble au concept des ambients. Les différents travaux sur les extensions diffèrentes sur la manière dont elles influent sur les notations de UML.

L'extension proposée dans [Baumeister *et al*. 03] modifie la syntaxe des diagrammes d'activités et de séquence, afin de pouvoir exprimer des scénarios contenant les aspects mobile et comportemental. La mobilité des objets est décrite par des descriptions d'états placées dans une boîte en dessous du nom de l'objet, et la localisation de l'objet est indiquée par l'affectation *[atLoc = localisation]* (voir la Figure 5).

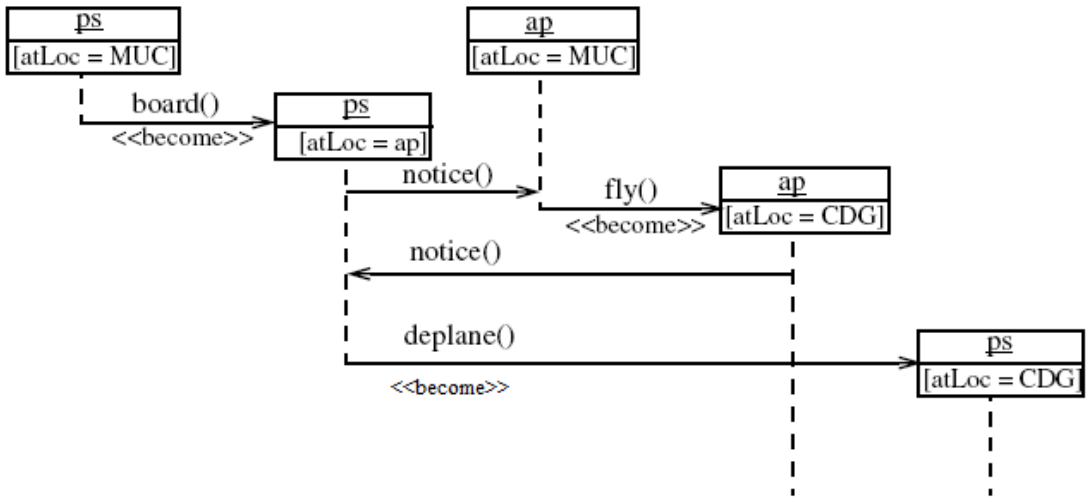


Figure 5. Exemple d'extension d'UML prenant en compte la mobilité

Quand un objet se déplace, il modifie son état et il faut créer une nouvelle boîte. Un nouveau type de message avec le stéréotype <<*become*>> permet de représenter les changements de localisation. Par exemple, dans la Figure 5, les messages *board()*, *deplane()* et *fly()* sont des artifices de modélisation pour exprimer la mobilité (ou le changement de localisation) des objets *ps* et *ap*. Par contre, le message *notice()* exprime une communication entre les objets, comme dans UML standard.

Le travail présenté dans [Grassi *et al*. 04] conserve les notations inchangées pour être compatible avec le standard UML 2.0. Des diagrammes de déploiement sont utilisés pour indiquer les configurations potentielles de localisations et la logique de la mobilité est modélisée par des diagrammes de type statechart, cette logique étant gardée séparément de la logique de l'application.

Aucune de ces extensions proposées ne fournit de sémantique formelle.

Mobicharts [Acharya *et al.* 04] est un cadre plus formel pour la modélisation des systèmes mobiles. Il fournit une notation graphique basée sur *ObjectChart* [Ernst et al. 95] qui est enrichie d'une notion de localisation. Cette notion s'inspire aussi de celle utilisée dans le concept des *ambients*. Mobicharts a été essentiellement utilisé pour modéliser des services pour la mobilité logique, comme la réplication ou la migration de tâches. Concernant la mobilité physique des dispositifs, Mobicharts se limite à décrire des fonctionnements en mode infrastructure avec des zones prédéfinies pour le déplacement des nœuds où une zone est définie comme une boîte. Comme la notation graphique ne peut pas fournir une sémantique précise, une spécification formelle des services a été donnée dans le langage de spécification RAISE [Nielsen et Geogre 98].

Les formalismes précédents, tous inspirés des ambients, considèrent une abstraction du mouvement en termes d'entrée et de sortie de boîtes. Parmi les formalismes offrant une autre vue de la mobilité, nous pouvons mentionner **MobileUNITY** [Roman et McCann 98] qui est une extension du langage de programmation Unity [Chandy et Mirsa 98]. Dans MobileUNITY, chaque processus a une variable qui modélise sa localisation, par exemple, la variable peut contenir les coordonnées physiques. Le mouvement est alors abstrait par l'attribution de valeurs aux variables. Des primitives de coordination sont proposées afin de faciliter la représentation des interactions entre les processus mobiles. Par exemple, la réception d'un message diffusé localement peut être représentée en utilisant une instruction réactive gardée par un prédicat sur les localisations de l'émetteur et du récepteur. MobileUNITY a une spécification formelle et une logique de preuve pour permettre la modélisation et la vérification des mouvements, des partages de données et des synchronisations dans un contexte mobile.

3.4. Modélisation du contexte

Nous avons vu dans l'introduction aux systèmes mobiles que le contexte joue un rôle important dans les applications mobiles. Par conséquent, il est nécessaire de pouvoir le prendre en compte au niveau du modèle. Des abstractions adéquates à sa prise en compte ont été étudiées dans la communauté des intergiciels. Le but est de fournir une interface de programmation qui masque à l'application les détails de la gestion du contexte. Les modèles de programmation soutenus par ces approches devraient avoir une incidence forte sur les méthodes pour spécifier, concevoir et valider des applications basées sur un intergiciel.

Modèle n-uplet. La notion de « n-uplet » (*tuple-space*) est une des abstractions les plus populaires. Originellement introduite dans le formalisme Linda [Gelernter 85], elle permet

aux entités de partager les informations stockées dans un espace global. Cet espace est structuré comme un ensemble de données élémentaires de n-uplet. Par exemple, <1, ''connected'', 2> pourrait être un n-uplet qui représente le fait que le nœud 1 est connecté au nœud 2. Des opérations ont été définies pour insérer ou rechercher des n-uplet. L'opération de recherche est réalisée par des requêtes qui cherchent des n-uplets qui s'apparient à un motif donné, par exemple le motif < ?int, ''connected'', 2>. L'abstraction de n-uplet peut être utilisée dans les systèmes mobiles, en rendant l'effet de la mobilité indirectement évidente par le biais d'un sous-espace accessible depuis un nœud. LIME [Murphy *et al.* 01] et SPREAD [Couderc et Banâtre 03] sont deux exemples d'intergiciels basés sur ce principe. Dans LIME, un n-uplet créé par une entité est rendu visible aux entités qui appartiennent au même groupe (le groupe est établi par un protocole d'appartenance de groupe, que nous étudierons au chapitre 2). Dans SPREAD, une zone physique est attachée à chaque n-uplet qui n'appariera que des requêtes émises par des entités contenues dans une portion spécifique de l'espace physique.

Modèle clé-valeur. C'est la structure de données la plus simple pour modéliser les informations contextuelles. Dans [Schilit *et al.* 94], des paires clé-valeur sont utilisées pour représenter le contexte en fournissant la valeur d'une information contextuelle (par exemple, la localisation) à l'application comme une variable d'environnement.

Modèle *Markup Scheme*. Dans ce modèle, la structure de données contextuelle est organisée hiérarchiquement avec des balises d'attributs et de contenus comme pour XML et HTML. Des exemples de cette approche ont été illustrés par les langages *Comprehensive Structured Context Profiles* [Held et al. 02], *Composite Capacities/Preferences Profile Context Extension* [Indulska et al. 03], *Pervasive Profile Description Language* (PPDL) [Chtcherbina et Franz 03], ConteXtML [Ryan 99].

3.5. Discussion

Au travers de cette partie sur la modélisation des systèmes mobiles, nous avons présenté divers formalismes et modèles de calcul mettant l'accent sur différents aspects des systèmes mobiles. Il est clair que la détermination d'abstractions adéquates pour modéliser et raisonner sur ce type de système est devenue un champ de recherche très actif. Toutefois, il apparaît que ce champ est encore exploratoire, et qu'il faudra du temps avant d'avoir des abstractions matures pour les systèmes mobiles prenant en compte toutes leurs spécificités. Dans une perspective de test, cela signifie qu'il n'y a pas encore de cadre bien établi pour se lancer dans des approches avancées de test basées sur des modèles. Dans les paragraphes suivants, nous allons présenter les défis posés par le test des systèmes mobiles.

4. Test des systèmes mobiles

Conformément à la terminologie de [Laprie et al. 96], la vérification consiste à déterminer si le système satisfait des propriétés, appelées *conditions de vérification*. L'objectif est de révéler les fautes de conception. Ces fautes ont pu être introduites au cours de n'importe quelle phase du cycle de développement. Il est important de les révéler rapidement pour raison de coût et d'efficacité. Le test est une des techniques de vérification existantes.

4.1. Positionnement du test parmi les techniques de vérification

Les techniques de vérification peuvent être classées selon qu'elles impliquent ou non l'exécution du logiciel (voir la Figure 6). Dans le cas sans exécution, la vérification est dite statique. Elle comporte l'analyse statique, la vérification par modèle et la preuve. Au contraire, la vérification dynamique, ou le test, se base sur l'exécution du programme. Ces différentes techniques sont résumées par la suite.

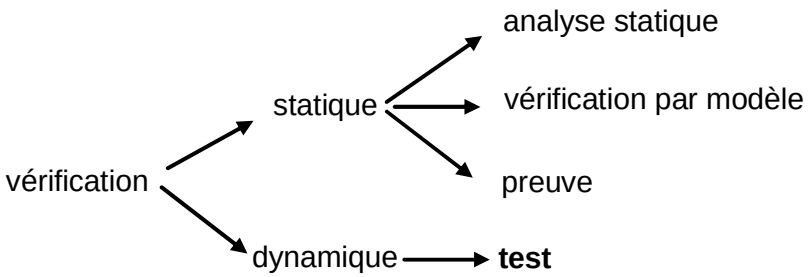


Figure 6. Classification des techniques de vérification

L'analyse statique vérifie des propriétés à partir d'une description du logiciel, sans l'exécuter. Elle est effectuée à l'aide d'outils informatiques, par exemple PolySpace [PolySpace], FLUCTUAT [FLUCTUAT], Astrée [Cousot et al. 05] ou encore aiT [Ferdinand et Heckmann 04].

Le principe de vérification par modèle (ou *model-checking*) consiste à vérifier certaines propriétés du modèle du système étudié. L'automate (fini), en général, est la technique de modélisation la plus utilisée. Les propriétés sont exprimées par une formule dans une logique temporelle. L'outil de vérification parcourt alors de façon exhaustive l'automate en vérifiant la formule pour chacun de ses états. Des outils performants ont contribué à la diffusion du model-checking, comme SMV [McMillan 93], SPIN [Homzmann 97], Uppaal [Larsen et al. 97], Mec [Arnold 90], KRONOS [Bogza et al. 98] et HyTech [Henzinger et al. 97].

La preuve a pour objectif de démontrer qu'un programme est correct vis-à-vis de sa spécification. Il existe des systèmes de preuve qui se distinguent par le degré d'automatisation de la preuve : vérificateur de preuve, outil d'assistance à la preuve, démonstrateur de théorèmes. Des exemples sont le système Boyer-Moore [Boyer et Moore 79], LP [Garland et Guttad 91], B [Abrial 96], AUTOMATH [de Bruijn 68], LCF [Gordon et al. 79], HOL [Gordon et Melham 93] et PVS [Owre et al. 95].

Dans la vérification dynamique, le programme est activé par des entrées numériques ou symboliques. Le test constitue la technique la plus populaire. Il consiste à exécuter une application en lui fournissant des entrées numériques. Les sorties sont analysées afin d'établir un verdict. Sa mise en œuvre nécessite la détermination de niveaux de test, la sélection de test, l'exactitude des résultats observés (ou le problème d'oracle). Dans la suite de ce paragraphe, nous allons mettre l'accent sur ces différents aspects (§ 4.2, 4.3 et 4.4). Nous mettons en parallèle l'état de l'art pour les systèmes traditionnels, et les travaux de recherche correspondants pour les systèmes mobiles.

4.2. Détermination des niveaux de test

4.2.1. Détermination des niveaux de test dans les applications traditionnelles

Typiquement, les niveaux de test sont déterminés selon le test unitaire, le test d'intégration et le test système [Beizer 90].

Test Unitaire. Une unité est la plus petite partie d'un programme/logiciel pour laquelle on peut entreprendre le test. Typiquement, c'est une fonction ou une procédure écrite par un seul programmeur. L'unité est vérifiée par rapport à sa conception détaillée.

Test d'intégration. L'intégration est un processus par lequel des composants sont agrégés pour former un composant plus complexe. L'objectif du test d'intégration est de tester l'intégration et la communication entre les composants. Par exemple, nous avons deux fonctions *F1* et *F2* qui ont passé le test unitaire. Le test d'intégration de *F1* et *F2* cherche à vérifier que les séquences d'appels à *F1* et *F2* soient correctes, qu'il n'y a pas d'incohérence de données communes entre *F1* et *F2*, etc. De plus, il peut vérifier si la fonction globale de *F1* et *F2* est correcte.

Test système. C'est la phase ultime du test, lorsque tous les composants du programme/logiciel ont été intégrés. Généralement, le test système est au moins en partie conduit sur des plateformes représentatives des interactions en vie opérationnelle : intégration avec le matériel et le support exécutif, utilisation d'un simulateur d'environnement ou test sur site. Ce niveau de test est différent du niveau de test d'intégration car il concerne tout le système, et pas seulement les interactions entre les composants (et avec des systèmes externes). En plus de tester les comportements fonctionnels, le test système peut inclure le test de performance, d'utilisation de ressources, de sécurité, etc.

4.2.2. Détermination du niveau de test dans les systèmes mobiles

Le choix du niveau de test dans un système mobile exige la détermination du sous-système, qui sera la cible du test, et de ses interfaces avec l'environnement. Typiquement, les niveaux de test sont déterminés dans le cadre d'une stratégie d'intégration, comme nous venons de le présenter. Cependant, dans les niveaux les plus intégrés de test, de nouvelles difficultés surgissent.

Une difficulté est que la notion de frontière des composants et des systèmes n'est pas aussi claire que dans les applications distribuées traditionnelles. Considérons un exemple où le système contient des nœuds avec des rôles différents : une application de type véhicule intelligent, qui implique à la fois des communications de véhicule à véhicule et de véhicule à l'infrastructure. Un véhicule peut alors être le système sous test, et ses voisins plus le serveur d'infrastructure constitueraient l'environnement. Nous pouvons aussi avoir un véhicule et le serveur d'infrastructure qui composent le système ciblé et le reste constitue l'environnement. Ou encore, le système ciblé peut être un ensemble de véhicules qui traversent une zone géographique donnée et le serveur d'infrastructure de cette zone est l'environnement, etc. la combinatoire est donc large. Notons aussi que, à un niveau donné, des instances alternatives peuvent être considérées, comme des instances avec des ensembles de véhicules qui correspondent à différentes caractéristiques de trafic routier.

En général, la détermination des niveaux de test doit dépendre des applications et des propriétés à vérifier. Nous n'avons pas trouvé dans la littérature de recommandations pour résoudre ce problème.

4.3. Sélection de cas de test

Sauf cas particulier, le test exhaustif sur toutes les entrées possibles n'est pas réalisable. On est donc amené à sélectionner, d'une manière pertinente, un sous-ensemble du domaine d'entrée. Cette décision est prise en fonction du critère de test, qui peut être lié à un modèle de la structure du programme ou à un modèle des fonctions que le programme doit réaliser. Ces deux cas conduisent respectivement au test structurel (§4.3.1) et au test fonctionnel (§4.3.2).

Les entrées de test peuvent aussi être générées selon deux procédés : la sélection déterministe ou la sélection probabiliste. Dans le premier cas, en donnant un critère de test, les entrées sont prédéterminées de manière à satisfaire ce critère. Dans le deuxième cas, défini par le test statistique [Waeselynck 93], les entrées sont tirées aléatoirement selon une distribution des probabilités sur le domaine d'entrée. Cette distribution est liée à la couverture des éléments spécifiés par le critère de sélection et détermine la longueur du test pour atteindre la qualité de test désirée.

Dans ce qui suit, nous mettons l'accent sur les différents types de critères, structurels et fonctionnels.

4.3.1. Test structurel

4.3.1.1. Test structurel dans les applications traditionnelles

Le test structurel sélectionne les entrées du programme à partir de la structure du programme. De nombreux critères sont définis dans [Beizer 90] pour effectuer cette sélection. Le modèle utilisé est le graphe de contrôle, éventuellement enrichi par des informations sur le flot des données dans ce graphe.

Le graphe de contrôle est construit à partir du code source de l'application. Il fournit une vue compacte de la structure de l'application. Les nœuds du graphe sont des blocs d'instructions qui sont exécutés toujours dans le même ordre. Les arcs entre les nœuds correspondent aux branchements conditionnels ou inconditionnels du programme. Une exécution peut alors être vue comme un parcours complet entre un nœud d'entrée et un nœud de sortie. Le chemin suivi est déterminé par les valeurs des entrées du test.

Le critère de sélection le plus sévère demande que soit activé, au moins une fois, chaque chemin exécutable entre les nœuds d'entrée et de sortie du graphe. Un chemin est exécutable s'il existe des entrées qui l'activent.

En pratique, ce critère n'est pas toujours applicable. D'une part, il n'existe pas d'algorithme général qui permette d'identifier automatiquement les chemins qui ne sont pas exécutables. D'autre part, dès lors qu'un programme contient une boucle, il peut comporter un grand nombre voire un nombre infini de chemins. Dans ce cas, on est amené à appliquer des critères plus faibles, comme "toutes les instructions" ou "toutes les branches". Ces critères demandent respectivement que toutes les instructions (tous les nœuds du graphe), ou toutes les branches du graphe soient bien activées au moins une fois pendant le test.

Le graphe de contrôle peut être enrichi avec des informations relatives à la manipulation de variables. Les critères dits de **flots de données** [Rapps et Weyuker 85] proposent alors de couvrir des sous-chemins entre les nœuds du graphe contenant des définitions des variables (elles reçoivent une valeur), et les nœuds (ou arcs) utilisant leurs valeurs dans des calculs (ou

des prédicats). On peut citer, par exemple, le critère "toutes les utilisations" ou ses versions moins sévères "toutes les définitions", "toutes les C-utilisations", et "toutes les P-utilisations".

D'une manière générale, le test structurel est appliqué pour des composants logiciels de petite taille, car le graphe de contrôle devient très vite complexe au fur et à mesure que la taille du code source croît.

4.3.1.2. Test structurel dans les systèmes mobiles

La justification du test structurel est que le comportement d'un programme est déterminé par sa structure. Cependant, dans les applications mobiles basées sur des intergiciels qui fournissent des paramètres contextuels, cette hypothèse est discutable. L'intergiciel peut activer des composants logiciels selon des contextes, via des interfaces cachées avec l'application. Donc, une partie du comportement de l'application peut être déterminé par des conditions contextuelles spécifiées dans l'intergiciel, plutôt que dans le code source de l'application.

Dans [Tse et al. 04], les auteurs montrent un exemple avec une application d'éclairage de lampadaires. Cette application tient compte de la localisation des visiteurs pour adapter la puissance lumineuse. Une faute dans le code source de cette application est liée aux variables contextuelles. Les auteurs ont montré que, même pour le test unitaire, une méthode traditionnelle (le critère de couverture « toutes les définitions, toutes les utilisations » dans ce cas) n'est pas suffisante car certaines conditions permettant de révéler les fautes intéressantes se situent dans l'intergiciel et non pas dans le code applicatif. Nous rencontrons ainsi un problème de délimitation de frontière entre l'application et le support de l'intergiciel : le comportement applicatif dépend d'une coopération entre les lampadaires qui est gérée par l'intergiciel.

Afin de considérer les données contextuelles, [Wang et al. 07] proposent une approche dont la nouveauté se situe dans l'application de techniques existantes pour identifier et contrôler des scénarios contextuels à explorer. La première étape est d'identifier les endroits où les paramètres contextuels peuvent avoir une influence sur le comportement du programme. Ces endroits sont appelés les *capps*. Ensuite, on construit le graphe de contrôle où certains nœuds sont annotés avec les *capps*. Dans la deuxième étape, on essaie d'explorer les scénarios contextuels qui génèrent, éventuellement, des comportements différents. Cette exploration consiste à traverser le graphe de contrôle annoté. Le critère principal est de couvrir au moins une fois les *capps* dans une exécution. Cette approche est intéressante car elle utilise effectivement les techniques classiques pour générer les cas de test afin de couvrir les paramètres contextuels.

4.3.2. Test fonctionnel

4.3.2.1. Test fonctionnel dans les applications traditionnelles

Le test fonctionnel repose sur un modèle qui décrit le comportement attendu d'un programme. Ce type de test recouvre donc un ensemble de méthodes diverses, selon le formalisme utilisé pour modéliser le comportement d'un programme. Les méthodes les plus classiques sont présentées par la suite :

Classes d'équivalence. Dans cette méthode, le domaine d'entrée du programme est partitionné en un nombre fini de sous-domaines, en distinguant des plages de valeurs valides

et invalides pour chacune des variables d'entrée [Meyers 79]. Le critère consiste à sélectionner un élément par classe. Cette méthode peut efficacement être complétée par un test en choisissant des valeurs qui se situent aux frontières des différents sous-domaines.

Tables de décision. Une table de décision est utilisée afin d'identifier les combinaisons d'entrées qui influent sur la logique du programme. Elle comprend deux parties : une liste de conditions ou prédicats sur les entrées, et une liste d'actions à entreprendre (sorties). Chaque colonne de la table définit une règle, qui lie une combinaison de valeurs de vérité des conditions à une liste d'actions attendues. Le critère de test associé à la table, clairement, est d'activer au moins une fois chaque règle.

Spécifications algébriques. Une spécification algébrique est composée de types abstraits de données (les *sortes*) et d'opérations axiomatisées pour construire et manipuler des éléments de sortes. Les sortes et les profils d'opérations forment la signature de la spécification. Les axiomes définissent le comportement attendu des opérations. Tester un axiome consiste à instancier ses variables et à vérifier que le programme satisfait la formule obtenue. Le choix des instances d'axiomes s'effectue à l'aide d'hypothèses de sélection [Gaudel 95, Marre 95].

Langages ensemblistes. Les langages ensemblistes, tels que Z, B, VDM, permettent de modéliser l'espace d'état d'un logiciel et les opérations faisant évoluer cet état. Les méthodes de test associées [Dick et Faivre 93, Hierons 97, Behnia et Waeselynck 99] comportent deux aspects : la sélection de test pour chaque opération analysée individuellement (par exemple, les critères de couverture du prédicat avant/après de l'opération) et des séquences d'appels aux opérations.

Machines à états finis. Les machines à états finis (appelé aussi automates finis) sont utilisées pour modéliser des comportements séquentiels. Une machine à état fini est un graphe, dont les nœuds représentent les états du système, et les arcs représentent les transitions entre les états. Les transitions sont étiquetées avec l'entrée qui les provoque et les sorties qui sont générées lors du franchissement de la transition. Les critères de sélection pour les machines sont des critères de couverture de la structure du graphe [Offutt et al. 03] : passer par tous les états, par toutes les transitions [Huang 75], par toutes les paires de transitions [Pimont et Rault 79].

Des versions plus sophistiquées du test de toutes les transitions ont été définies. Elles visent à l'exhaustivité vis-à-vis d'un modèle de faute simple, induisant des erreurs de transfert (la transition amène à un autre état que celui spécifié) ou de sortie (la transition ne génère pas l'événement attendu). Les travaux les plus connus sont : la méthode W [Chow 78], la séquence de distinction et les séquences UIO (*Unique Input Ouput*) [Sabnani et Dahbura 85, Ural 92]. Ces méthodes ont également été étudiées pour des machines indéterministes et des machines étendues avec des données [Phalippou et Groz 90, Cavalli et al. 96].

Systèmes de transition étiquetés. Un système de transition étiqueté (*Labelled Transitions System - LTS*) est un formalisme utilisé pour modéliser des systèmes communicants. En pratique, on ne spécifie pas directement selon ce formalisme : la spécification est donnée dans un langage de haut niveau (par exemple, SDL) dont la sémantique est définie par un LTS. Les synthèses de test sont effectuées par composition du LTS avec des objectifs de test, qui décrivent des séquences d'événements à tester [Fernandez et al. 96, Castanet et al. 98, Kock et al. 98, Kerbrat et al. 99].

4.3.2.2. Test fonctionnel dans les systèmes mobiles

Dans le monde mobile, les problèmes de sélection de test deviennent plus complexes. Il faut, maintenant, considérer non seulement les entrées explicites au niveau applicatif, mais aussi les paramètres contextuels jouant un rôle important dans les comportements du logiciel.

En pratique, les approches appliquées sont souvent informelles. Les cas de test sont construits manuellement, à partir des exigences de l'utilisateur. Un exemple est donné par [Henricksen et Indulska 06], qui abordent ainsi le test fonctionnel dans le processus de développement des applications mobiles.

Pour des approches plus formelles, la majorité des contributions provient de la communauté du test de protocoles. Les auteurs se basent sur des modèles utilisés dans les systèmes répartis, en les adaptant aux systèmes mobiles. Le langage de modélisation largement utilisé dans la communauté est SDL (*Specification and Description Language*) [ITU 02], et nous avons mentionné l'existence de méthodes pour générer des cas de test à partir d'une spécification SDL et un ensemble d'objectifs de test. Comme le langage SDL est conçu pour les protocoles de communication, il se focalise sur les émissions et les réceptions des messages. De plus, SDL ne supporte que la communication point-à-point, alors qu'il faut pouvoir modéliser la communication par diffusion radio dans le voisinage. Il faut donc introduire des artifices de modélisation pour couvrir les nouveautés des systèmes mobiles, comme cela est fait dans [Noudem et Viho 05], [Cavalli *et al.* 04].

Dans [Noudem et Viho 05], les auteurs étudient le MIPv6, un protocole dans IPv6 qui permet aux nœuds de rester connectés tout en se déplaçant. Dans ce travail, les auteurs ont voulu générer des cas de test automatiquement à partir d'une spécification. Pour cela, un modèle avec une topologie prédéfinie de l'environnement a été construit en SDL. Quant à la communication *multicast (*un message est envoyé à plusieurs destinataires*)*, le modèle suppose qu'un message sera envoyé à tous les processus dans l'environnement et qu'un contrôleur désignera les destinataires selon le contexte de mobilité.

Dans [Cavalli *et al.* 04], c'est le protocole de routage DSR (*Dynamic Source Routing protocol*) du domaine ad hoc qui est étudié. Ce protocole permet de découvrir et de maintenir les chemins dans le réseau ad hoc avec multi-sauts (plusieurs intermédiaires entre un émetteur et un destinataire). La spécification du protocole est décrite en langage SDL. Chaque nœud est représenté par un bloc SDL. Là encore, la communication entre les nœuds s'effectue via un contrôleur qui est responsable de toute la communication pour tout le système (gestion de la topologie, livraison de messages en *unicast* ou en *broadcast*, etc.). Le modèle est complété par une description de la configuration du système (par exemple, 5 nœuds connectés selon une certaine topologie, avec brisure d'une des connexions au bout de 15 secondes). Après avoir configuré le modèle sous test, la sélection de test est faite en accord avec un ensemble d'objectifs de test. Les scénarios de test sont générés par un outil développé par les auteurs.

Comme nous pouvons le constater, les travaux de [Noudem et Viho 05] ou [Cavalli *et al.* 04] ont adopté des solutions similaires pour contourner le fait que SDL n'est pas destiné à la modélisation des systèmes mobiles. Les auteurs ont dû ajouter des composants spécifiques (un composant pour chaque nœud et un contrôleur centralisé) afin de capturer la notion de communication dans le voisinage d'un nœud. De plus, pour générer les cas de test, les auteurs doivent choisir a priori les configurations de la topologie du réseau (ces configurations sont paramétrables, dans le cas de [Cavalli *et al.* 04]).

Ces contributions montrent qu'il est possible d'importer des techniques de test basées sur des modèles (*model-based testing*) dans le domaine des applications de l'informatique mobile. Cependant, cela nécessite d'adopter des artifices de modélisation et de choisir des configurations prédéfinies de topologie.

En conclusion de ce paragraphe, nous remarquons que l'absence de langage de spécification standard pour les systèmes mobiles pose problème pour la mise en œuvre du test fonctionnel. Comment choisir les entrées de test et les configurations expérimentales susceptibles de révéler les fautes reste une question ouverte.

4.4. Problème de l'oracle

Le test pose un problème de mise en œuvre qu'il ne faut pas négliger : le problème de l'oracle, ou comment décider de l'exactitude des résultats observés, fournis par le programme en réponse aux entrées de test [Weyuker 82]?

4.4.1. Solutions classiques pour l'oracle

La solution la plus répandue est manuelle. L'opérateur se base sur les documents de spécification, et aussi éventuellement sur des tableaux d'exemples, des plages de valeur attendues, des propriétés de cohérence entre différentes données, etc. Le verdict peut être facilité puisque le testeur choisit lui-même les entrées de test : il lui suffit alors de sélectionner des entrées pour lesquelles il connaît la réponse correcte attendue. Cependant, cette méthode conduit naturellement à restreindre le test à des entrées simples.

L'approche manuelle étant fastidieuse et source d'erreur, un dépouillement automatisé des résultats de test est toujours souhaitable. La solution la plus satisfaisante suppose l'existence d'une spécification formelle. Celle-ci est alors utilisée pour déterminer les sorties attendues, soit lors de la sélection des entrées de test, soit a posteriori (par exemple, en calculant les sorties sur une spécification exécutable). Une autre solution automatique au problème de l'oracle est le test dos-à-dos, utilisable pour des systèmes critiques comportant de la redondance logicielle. Les sorties sont comparés à celles fournies par une autre version du programme, développée indépendamment à partir de la même spécification. Une discordance indique la présence d'une faute dans une des deux versions ou dans les deux. C'est la spécification qui joue alors le rôle de référence pour identifier (manuellement) la ou les versions incorrectes. Ce type d'oracle ne permet cependant pas de révéler les fautes corrélées, présentes dans les deux versions.

Dans le cas où on ne dispose pas d'une spécification formelle complète, ou de plusieurs versions, des vérifications partielles automatisées peuvent être appliquées. L'oracle cible alors quelques propriétés. Cette approche signifie que, si les sorties ne passent pas les vérifications, elles sont incorrectes. Cependant, le fait qu'elles passent les vérifications ne garantit pas qu'elles soient correctes.

Le test passif est un exemple d'approche s'appuyant sur des vérifications partielles, pour des systèmes complexes. Il consiste à observer le comportement d'un système en exécution, soumis à une activité réelle ou synthétique. La trace enregistrée est analysée pour vérifier s'il y a violation de propriétés. En pratique, le test passif est complémentaire aux approches actives qui choisissent les cas de test avant d'exécuter le test. En effet, il permet d'observer plus de comportements que quelques cas de test prédéfinis. Les propriétés à vérifier sont

typiquement des propriétés invariantes qui peuvent, ou non, être tirées d'un modèle de comportement formel (par exemple, voir [Ladani et al. 05]).

Le test « métamorphique » (*Metamorphic testing*) [Chen 03] fournit un autre exemple d'oracle partiel. Son principe est d'utiliser les relations attendues entre les entrées et les sorties de plusieurs exécutions. Une relation métamorphique ne vérifie pas la correction d'une seule sortie mais plutôt des propriétés portant sur l'ensemble des exécutions.

4.4.2. Le problème de l'oracle dans les systèmes mobiles

Dans le cadre d'applications mobiles, il peut y avoir des cas où les entrées sont non observables et non contrôlables. Par exemple, si un dispositif est testé dans un environnement réel, les dispositifs avoisinants peuvent rejoindre et quitter le réseau de manière imprévisible. Les entrées du dispositif cible ne peuvent donc pas être déterminées et les sorties attendues ne peuvent pas être prévues.

Un exemple de non contrôlabilité et de non observabilité est présenté dans [Leung et al. 04] où les auteurs analysent les sorties d'une application embarquée dans les téléphones portables par rapport à quelques (faibles) propriétés invariantes. Dans leur cas, le problème d'oracle est plus compliqué, car ils exécutent un test sur site (l'application est embarquée dans le dispositif réel et testée dans un environnement urbain réel). Une partie des entrées est donc incontrôlable et même inobservable. Les dispositifs dans le voisinage peuvent entrer et sortir du réseau d'une manière spontanée. Par conséquent, les entrées pour le dispositif ciblé ne peuvent pas être déterminées, et donc les sorties attendues ne peuvent pas être décidées.

Comme nous l'avons mentionné, il est souhaitable d'avoir une spécification formelle complète pour construire un oracle. Cependant, nous constatons que la spécification d'applications mobiles est elle-même un problème.

Plusieurs travaux ont mis en œuvre des oracles partiels, basés sur des propriétés. Certains se sont inspirés de l'approche métamorphique pour tester des applications qui dépendent du contexte, dans un cadre mobile [Chan et al. 05], [Tse et al. 04]. Intuitivement, les relations contextuelles choisies sont isotropes et expriment le fait que des contextes similaires devraient produire des réponses similaires de l'application. Le test passif a été appliqué à un protocole de routage dans les réseaux ad hoc [Cavalli et al. 09].

4.5. Test et représentations graphiques de scénarios

Nous voulons aborder les langages de scénarios graphiques car ils ont prouvé leur utilité dans le cadre d'activités de test. Ils sont largement utilisés pour représenter des scénarios d'interactions dans les systèmes distribués. De fait, la représentation de scénarios peut avoir différents objectifs : la capture d'exigences [Kugler et al. 07], la spécification d'objectifs de test (c'est-à-dire, d'interactions à couvrir par les cas de test) [Grabowski et al. 93], la conception de cas de test [Pickin et Jézéquel 04], ou l'analyse de traces d'exécution [Briand et al. 06]. Leur popularité est due à leur syntaxe conviviale qui facilite leur compréhension tout en ouvrant la porte aux traitements formels (cela exige pourtant que la notation utilisée possède une sémantique précise).

En général, ces langages de scénario se basent sur l'ordonnancement partiel des événements. Un exemple est donné avec la Figure 7.

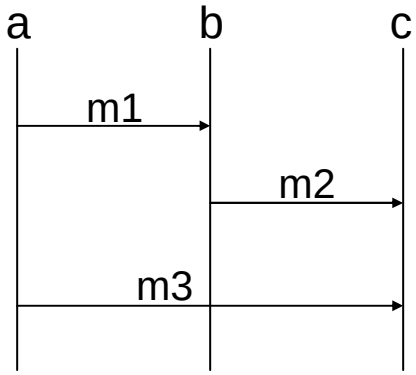


Figure 7. Exemple d'un diagramme MSC

Dans ce diagramme, *a*, *b* et *c* sont des lignes de vie, qui représentent des objets dans un système. *m1*, *m2* et *m3* sont des messages échangés où chacun correspond à un événement d'émission et de réception. L'ordre entre les événements est défini par l'émission/la réception d'un message (par exemple, l'émission du message *m1* est avant sa réception) ou sur une ligne de vie (par exemple, la réception du message *m2* est avant la réception du message *m3*). Cependant, il n'existe pas d'ordre entre l'émission de *m2* et l'émission de *m3*.

La Figure 7 montre un diagramme de base. En réalité, il existe des opérateurs qui permettent de combiner les diagrammes de base pour construire des systèmes plus complexes. Ce sont des opérateurs de séquence, de composition parallèle, de choix alternatif, d'itération, etc.

Dans la littérature, les langages de scénario les plus utilisés peuvent être classifiés en deux grandes familles : les Message Sequence Chart et ses dérivés d'un côté, et les diagrammes d'UML de l'autre. Notons que, dans les versions récentes, les notations dans ces deux familles sont assez similaires.

Les **MSC *(Message Sequence Chart)*** et **HMSC (*High-level Message Sequence Chart*)** [ITU 2000] sont les premiers formalismes pour la spécification des interactions entre les objets dans les systèmes distribués. Un diagramme MSC décrit un échange de messages que le système doit satisfaire quand le système est implémenté. L'enchaînement de telles interactions, ou scénario, représente une exigence du comportement du système. Comme la notion est précise et claire, le MSC est très intéressant. Par rapport aux notions textuelles, les diagrammes MSC sont plus faciles à analyser, à corriger et à partager entre les concepteurs. Notons que TTCN-3 (*Testing and Test Control Notation*) [Willcock et al. 05], un langage dédié à la spécification et à l'implémentation de cas de test, intègre une vue graphique dans laquelle les MSCs sont utilisés.

Des extensions des MSCs ont été proposées. Un des travaux d'extension le plus populaire a été de donner au langage des modalités de possibilité/obligation. On cherche par exemple à exprimer un scénario du type : si tel échange de messages se produit (possibilité), alors tel autre devra immédiatement suivre (obligation). De telles extensions sont développées dans les langages TMSC et LSC, tous deux dotés d'une sémantique formelle. Dans TMSC (*Triggered Message Sequence Charts*) [Sengupta et Cleaveland 06], le comportement qui active l'obligation est exprimé dans un fragment de diagramme appelé « trigger ». Dans le langage LSC (*Live Sequence Charts*) [Damm et Harel 01], la notion de température assignée aux messages offre beaucoup de souplesse pour combiner des comportements possibles (température froide) et obligatoires (température chaude). De plus, on distingue les

diagrammes « *existentiels* », qui expriment un comportement exhibé par au moins une exécution du système, et les diagrammes « *universels* » qui expriment un comportement vérifié par toutes les exécutions.

Les diagrammes de séquence de UML2.0. Les diagrammes de séquences montrent des interactions entre les objets dans le système selon un ordre chronologique. Par rapport aux versions précédentes, les nouveautés de UML 2.0 ont été d'introduire les relations de conformité par des opérateurs *neg*, *assert*, *ignore*, et *consider*, et une sémantique informelle des traces valides/invalides. Différents profils spécialisés existent, comme le profil UTP (*UML Testing Profile*) qui vise à décrire des cas de test. Un autre langage de description de test est TeLa [Pickin 03], basé sur une version antérieure (UML version 1.4), et disposant d'une sémantique formelle.

Comme indiqué plus haut, les deux familles de langages de scénarios – MSCs et diagrammes de séquences UML – sont étroitement liées. Par conséquent, il peut y avoir des fertilisations croisées entre ces notations. Un exemple est le langage MSD (*Modal Sequence Chart*) [Harel et Maoz 08]. Il est défini comme une extension des diagrammes de séquences UML 2.0 en se basant sur les notions centrales de LSC (température, diagrammes universels/existentiels). Une contribution de MSD est de proposer une interprétation claire des deux opérateurs *assert* et *negate* de UML 2.0, en utilisant la notion de température.

Les descriptions de scénarios se sont avérées utiles aux activités liées au test, dans les applications traditionnelles. Dans certains cas, les scénarios sont utilisés conjointement avec un modèle complet de comportement. Mais leur utilisation n'exige pas de disposer d'une spécification formelle complète. Le scénario modélise un fragment de comportement, ou une propriété sur les comportements attendus. La spécification complète d'une application mobile nous paraissant problématique, une approche basée sur des scénarios pourrait offrir une solution pragmatique au test de ce type d'application. Cette idée sera développée dans la suite de nos travaux.

5. Plate-forme expérimentale

Une plate-forme de test permet d'implémenter des jeux de test. Elle doit offrir des moyens pour observer et contrôler le transit des événements d'entrée et de sortie, reçus ou émis par chaque composant. Typiquement, ceci s'effectue au moyen de Points de Contrôle et d'Observation (PCO). Les PCO sont conçus pour accéder à l'interface du composant, c'est-à-dire, ils peuvent envoyer des événements au composant (par le point de contrôle) et observer le résultat (par le point d'observation). Un PCO peut être global ou local. Les architectures de test centralisées sont les plus simples car elles ne nécessitent qu'un PCO global. Dans les architectures distribuées de test, les PCO locaux doivent être synchronisés.

5.1. Discussion sur l'environnement de test des applications mobiles

Dans le cas général, il est souhaitable de tester des applications dans les conditions les plus réalistes possibles vis-à- vis des conditions opérationnelles. Par exemple, les protocoles et services WAP peuvent être validés et expérimentés sur la plate-forme PLATONIS [PLATONIS]. Cette plate-forme vise les technologies qui permettent de déployer des services pour la téléphonie mobile (UMTS, GPRS, GSM). D'autres auteurs présentent des travaux pour tester les applications dans un environnement opérationnel réel ou proche du réel : dans [Leung et al. 04] une application téléphonique est testée avec l'aide de personnes qui portent

des dispositifs dans une zone urbaine, dans [De Bruin et al. 04] une application de type véhicule-à-véhicule est testée avec trois véhicules sur une route ou encore, dans [Killijian et al. 08] une application de sauvegarde est évaluée avec des robots mobiles qui portent des dispositifs et utilisent un système de localisation à l'intérieur d'un bâtiment. Cependant, il est évident que de tels types de test sont limités par leur coût élevé de mise en œuvre et les problèmes d'observabilité et de contrôlabilité. Il est de plus difficile d'avoir des informations contextuelles toujours cohérentes car il est difficile d'avoir les mêmes conditions de test. Par exemple, en utilisant le GPS pour obtenir l'information de localisation, il est possible d'obtenir des valeurs différentes pour un même point qui sont fonction des incertitudes des appareils GPS utilisés. De plus, on ne peut pas toujours effectuer le test dans un environnement totalement réel. Par exemple, les appareils GPS ne fonctionnent pas à l'intérieur des bâtiments. Enfin, il faut aussi faire attention aux problèmes de sécurité dans certains types d'applications. Par exemple, les scénarios qui sont expérimentés avec une application mettant en jeu un véhicule ne doivent pas mettre le conducteur en danger. Pour ces raisons, il est préférable qu'une majeure partie du test soit implémentée avec des moyens de simulation/émulation.

Pour les systèmes informatiques mobiles, les moyens utilisés doivent essentiellement couvrir l'aspect topologique, la technologie de communication (sans fil et aussi filaire) et la dépendance vis-à-vis du contexte. Ceci peut être fait à l'aide d'un simulateur de réseau de communication et d'un simulateur de contexte, présentés respectivement dans les paragraphes 5.2 et 5.3. Dans le paragraphe 5.4, nous allons présenter des exemples de plates-formes qui utilisent ces types de simulateurs. Il existe un autre type de plate-forme dans lequel la mobilité physique est simulée par la mobilité logique. Cela sera aussi illustré par un exemple dans le paragraphe 5.4.

5.2. Simulateur de réseau de communication

Un simulateur de réseau de communication est un programme qui permet d'évaluer le comportement d'un réseau en calculant les interactions entre les entités du réseau. Un simulateur de réseau peut être implémenté selon une approche centralisée où les propriétés du réseau sont contrôlées par un processeur, ou selon une approche décentralisée où chaque nœud dans le système calcule ses propriétés du réseau et ces données doivent être synchronisées entre les nœuds participants via un réseau filaire.

Des exemples de la première approche sont ns-2 [NS2], GlomoSim [GlomoSim], JiST/SWANS [Jist/SWANS]. Parmi eux, le simulateur ns-2, développé à l'Université de Berkeley, est le plus utilisé. C'est un simulateur à événements discrets. Il permet de simuler les réseaux sans fil avec multi-sauts. Il fournit des protocoles au niveau MAC (IEEE 802.11), des propagations des signaux, des protocoles de routage, etc.

La deuxième approche peut être considérée comme l'émulation du réseau sans fil par un réseau filaire. Des émulateurs comme EMWIN [Zheng et Ni 02], JEmu [Flynn et al. 02], ou MobiEmu [Zhang et Li 02] sont représentatifs de cette approche. Par exemple, MobiEmu utilise un réseau filaire de *n* machines pour émuler un réseau mobile de *n* nœuds. Un contrôleur est utilisé pour gérer la topologie du système. La communication entre les nœuds est simulée par des filtres des paquets de messages en se basant sur l'information de topologie.

5.3. Simulateur de contexte

Un simulateur de contexte est un outil pouvant créer et gérer des informations contextuelles demandées par les applications. Ce type de simulateur est utilisé pour développer, visualiser, et évaluer des applications mobiles dépendantes du contexte de localisation. Certains simulateurs sont construits à partir de moteurs graphiques de jeux vidéo et sont utilisés pour simuler une vue 2D ou 3D de l'environnement physique, par exemple Ubiwise [Barton et Vijayaragharan 03]. Les applications véhicules-à-véhicules peuvent utiliser des simulateurs de traffic (comme STRAW [Choffnes et Bustamante 05]) qui simulent les mouvements dans une zone routière. Il existe également des simulateurs non spécialisés, qui génèrent des événements de localisation génériques, comme l'outil GLS [Sanmiglingam et Coulouris 02]. Nous allons décrire ces différents outils ci-dessous.

Ubiwise. Ubiwise [Barton et Vijayaragharan 03] simule des aspects d'informatique ubiquitaire. Il se focalise sur les traitements et les dispositifs utilisés dans l'environnement sans fil. Il fournit aux utilisateurs une interface graphique (un modèle en 3 dimensions) pour représenter les dispositifs simulés et l'environnement. Cependant, Ubiwise n'est pas construit sur un simulateur réseau existant (comme ns-2, par exemple). Donc, l'aspect réseau dans Ubiwise reste limité. L'outil ne supporte que la connexion entre les applications et les dispositifs, il ne fournit pas de connexion entre les applications et la couche réseau.

Un autre simulateur de contexte qui se base sur le même principe est QuakeSim [Bylund et Espinoza 01].

STRAW. Contrairement à l'environnement ouvert où les dispositifs peuvent se déplacer librement (par exemple, dans une salle), les véhicules sont souvent contraints par plusieurs éléments. STRAW [Choffnes et Bustamante 05] prend en compte les contraintes de déplacement des véhicules sur les routes en définissant et intégrant des données réelles des plans de villes des Etats-Unis. Il essaye de fournir un modèle réaliste afin d'avoir des résultats précis des simulations.

Il existe d'autres simulateurs qui génèrent un trafic routier dans un réseau de véhicules automobiles comme SUMO [SUMO], CARSSIMA [Eichler et al. 05], VISSIM [Lochert et al. 05], VanetMobiSim/CanuMobiSim [Harri et Fiore 06], ou encore le simulateur développé dans le cadre du projet IMPORTANT [IMPORTANT], que nous avons utilisé pour des expérimentations décrites dans le chapitre 5 de ce mémoire.

GLS. GLS [Sanmiglingam et Coulouris 02] est développé dans le contexte du projet QoSDReAM [Naguib et Coulouris 01]. Ce simulateur se focalise sur une modélisation réaliste des déplacements des personnes (appelées *locatables*) dans un environnement donné. Les personnes cherchent automatiquement le chemin le plus court pour atteindre leur destination et évitent les obstacles. Dans le simulateur, il existe des modèles des zones géographiques. Quand les *locatables* se déplacent, les sorties des capteurs de localisation simulés ont le même format que les capteurs de la réalité. On est donc capable d'utiliser ces sorties comme entrées pour tester des applications, sans avoir à les déployer physiquement.

5.4. Exemples de plates-formes expérimentales

En utilisant ces composants, simulateur de réseau et simulateur de contexte, des plates-formes expérimentales pour évaluer des applications mobiles peuvent être construites. Nous en présentons quelques exemples trouvés dans la littérature.

5.4.1. Exemple dans le domaine de la santé

Dans [Morla et Davies 04], le simulateur réseau ns-2 est utilisé pour évaluer une application du domaine de la santé. Pour cela, les auteurs ont construit un environnement de test distribué. Cet environnement est composé de trois composants principaux : le simulateur réseau ns-2, l'application à tester et le moniteur du système. Le moniteur est responsable de la conduite des expérimentations, de la configuration du simulateur et de l'enregistrement des données. Il joue, donc, le rôle d'un simulateur de contexte. Les coopérations entre les composants sont possibles en utilisant un ensemble d'interfaces ouvertes. Ces coopérations sont implémentées en utilisant les services Web. L'architecture de cette plate-forme est illustrée dans la Figure 8.

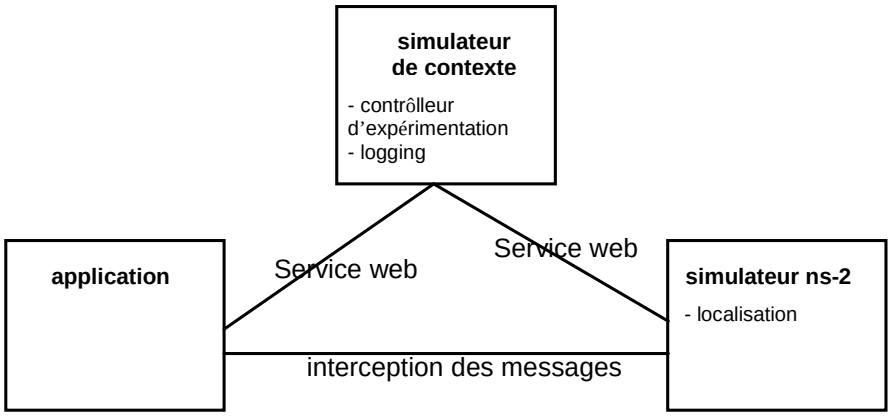


Figure 8. Architecture de la plate-forme pour l'évaluation de l'application

5.4.2. Exemple dans le domaine des véhicules intelligents

Le travail dans [C. Schroth *et al.* 05] évalue un système de communication par messages entre les véhicules. Conceptuellement, l'environnement de simulation contient trois composants : (1) le simulateur du trafic qui calcule les positions des véhicules pour un scénario donné, (2) le simulateur réseau qui imite le comportement du réseau et (3) l'application elle-même. Dans l'implémentation, l'application est en fait intégrée dans le simulateur réseau comme un module supplémentaire. Cependant, les auteurs indiquent bien que, dans le cas général, l'environnement doit comprendre les trois composants.

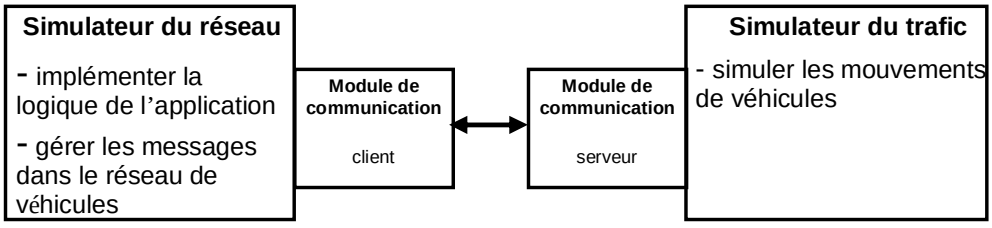


Figure 9. Architecture du simulateur pour l'échange de messages entre véhicules

L'architecture de l'environnement est montrée dans la Figure 9. Le simulateur du réseau est considéré comme un client du simulateur du trafic. Par exemple, il interroge le simulateur

de trafic sur le nombre de véhicules dans le réseau et leur position. Le simulateur du réseau est implémenté en utilisant ns-2. Le simulateur CARISMA est utilisé pour gérer le trafic.

5.4.3. Un exemple d'émulateur

Dans [Nickelsen *et al.* 06], les auteurs proposent une plate-forme pour valider des propriétés temps réel. La simulation du réseau est implémentée par un émulateur (central) de topologie et un commutateur (central) de réseau auquel les nœuds sont connectés par des liens filaires. Chaque nœud est émulé par une machine équipée d'une connexion Ethernet. C'est-à-dire, les liens sans fil sont simulés par les liens filaires afin d'avoir une meilleur contrôlabilité. En temps réel, l'émulateur impose les propriétés du réseau (la latence, la perte de messages, etc.) aux nœuds. Ces propriétés sont établies en fonction de la mobilité des nœuds qui est simulée. Son architecture est illustrée dans la Figure 10.

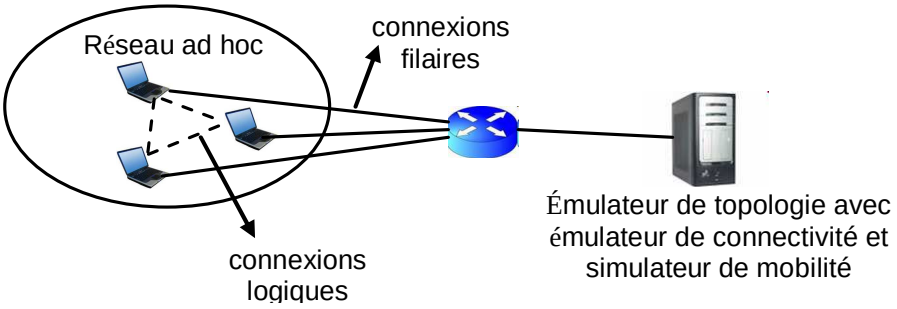


Figure 10. Architecture d'émulateur développé à Åalborg

5.4.4. Flying Emulator : simuler la mobilité physique par la mobilité logique

Une approche très différente des précédentes consiste à simuler la mobilité physique par la mobilité logique. *Flying Emulator* [Satoh 03] (illustré dans la Figure 11) en est un exemple. L'auteur de cet émulateur considère les dispositifs physiques et les logiciels comme des agents mobiles se déplaçant d'un point d'accès (*Access Point*-AP) à un autre. Les agents sont hiérarchiques, par exemple un agent portant du code applicatif peut être défini comme un sous-agent d'un dispositif mobile. Le mouvement de l'agent dispositif entraîne alors celui de tous ses sous-agents. Chaque AP représente un réseau local sans fil avec une infrastructure et permet au logiciel, porté par un agent mobile, de se connecter aux services sur le serveur. En plus des interactions internes (dans le dispositif mobile), les applications peuvent communiquer avec l'environnement externe (c'est-à-dire, avec le serveur local). Le déplacement des dispositifs mobiles vers un nouveau réseau local est simulé par la migration des agents vers l'AP de ce nouveau réseau. Un serveur de contrôle à distance est responsable de la gestion de tous les processus de déplacement. Notons que, dans ce travail, on ne s'intéresse pas aux applications fonctionnant sur des réseaux ad hoc.

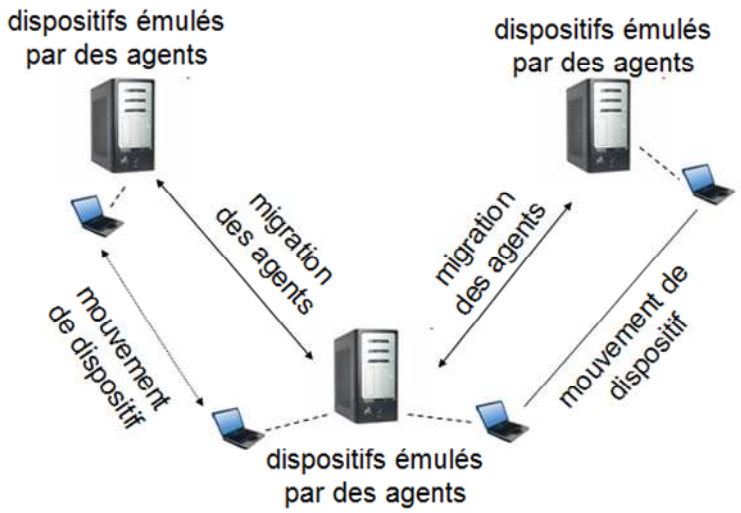


Figure 11. Architecture de Flying Emulator

5.5. Discussion des environnements de simulation

Au travers des différents exemples, sauf pour le cas de la simulation de la mobilité physique par la mobilité logique, nous avons pu constater qu'une plate-forme de test comporte typiquement les composants suivants :

- Un support d'exécution afin de pouvoir émuler le support exécutif pour le code applicatif.
- Un simulateur réseau pour simuler le comportement global du réseau réel pouvant gérer différents types de connexions, différents protocoles de routage, des défaillances de nœuds ou de connexions, etc.
- Un contrôleur de contexte pour simuler les informations contextuelles, comme la position des nœuds dans une application inter véhicules. Éventuellement, le mouvement est contraint par un modèle de mobilité, conformément auquel les positions des nœuds sont générées.

Ces plates-formes ont été développées principalement pour des objectifs d'évaluation, mais elles peuvent également servir pour des objectifs de vérification. Les plates-formes de test utilisant de tels outils rencontrent des problèmes qui sont classiques dans le cadre des systèmes distribués (par exemple, des problèmes de synchronisation des composants de test). De plus, elles peuvent avoir des problèmes supplémentaires qui sont liés à la simulation discrète couplée à des processus continus. Par exemple, dans [Schroth *et al.* 05], il y a une discussion sur le couplage du trafic de véhicules avec les simulateurs réseau, et en particulier sur l'approximation des déplacements continus des véhicules par des calculs échantillonnés.

Un autre problème a été soulevé dans [Cavin *et al.* 02], [J. Lessman *et al*. 08], [Di et al. 08] qui montre que, pour les mêmes ensembles de données, différents simulateurs de réseau produisent des résultats totalement différents. Par exemple, dans [Cavin *et al.* 02], les auteurs ont conduit des expérimentations d'un algorithme simple de diffusion des messages sous différents simulateurs (ns-2, GlomoSim, Opnet) dans les mêmes conditions (paramètres, scénarios). L'évaluation des performances de l'algorithme donne des résultats différents d'un simulateur à l'autre, aussi bien d'un point de vue quantitatif que qualitatif. Ceci pose la question de la confiance que l'on peut accorder à des résultats de simulation.

Le problème de concevoir une plate-forme appropriée reste donc ouvert et peut mener à de nouvelles directions pour des travaux de recherche dans le domaine des systèmes informatiques mobiles.

6. Conclusion

Les avancées technologiques du monde "sans fil" ont conduit au développement des systèmes informatiques mobiles. Leurs caractéristiques propres (dynamicité de la structure du système, communication avec des partenaires inconnus dans le voisinage, dépendance vis-à-vis du contexte) posent de nouveaux défis au processus de développement des applications mobiles. En particulier, il a fallu identifier les problèmes spécifiques posés par la vérification des applications mobiles par test.

Dans la littérature, il existe des travaux qui essaient de définir une spécification de ces applications ainsi que des travaux qui portent sur leur test. Au travers de ce chapitre, nous avons pu d'abord constater qu'il n'y a aucun langage de spécification standard pour les systèmes mobiles. Cela pose des soucis majeurs pour le test. Il est ainsi difficile d'effectuer du test fonctionnel mais aussi de construire un oracle pour comparer les sorties de test. Nous avons argumenté que les représentations graphiques de scénarios pouvaient constituer une solution.

Par la suite, nous avons présenté des problèmes potentiels, rencontrés en résolvant les questions fondamentales de test (détermination du niveau de test, sélection de test, problème de l'oracle, plate-forme de test). Dans le chapitre 2, nous allons illustrer certains de ces problèmes par un cas d'étude dans le domaine des réseaux ad hoc.

CHAPITRE 2

CAS D'ETUDE :

UN PROTOCOLE D'APPARTENANCE DE GROUPE DANS LES RESEAUX AD HOC

1. Introduction

Nous avons débuté nos travaux par un état de l'art sur le test dans les systèmes mobiles (chapitre 1). Nous constatons qu'il y a des défis posés au test dans ce nouveau type de système. Afin de mieux comprendre ces problèmes, et d'en obtenir une vue concrète, nous conduisons maintenant une étude de cas. Il s'agit d'un protocole d'appartenance de groupe (en anglais, *Group Membership Protocol-GMP*) dans le domaine des réseaux ad hoc.

Un protocole d'appartenance de groupe est un service de base implanté au cœur des systèmes distribués tolérants aux fautes. Son but est de maintenir une vue cohérente des membres du groupe, et ce en dépit des fautes pouvant affecter certains nœuds. Le problème a été abondamment étudié pour les systèmes distribués traditionnels. Cependant, il doit être revisité pour prendre en compte les nouvelles spécificités des systèmes mobiles, comme la mobilité des nœuds et l'adaptation à un contexte qui évolue dynamiquement.

Le GMP que nous avons retenu comme cas d'étude a été proposé par Huang et al. [Huang et al. 04]. Il est destiné à des nœuds mobiles formant des réseaux ad hoc. Sous certaines hypothèses (fortes) sur l'environnement, le GMP étudié vise à offrir un service de groupe dépendant de la localisation des nœuds. L'article de référence [Huang et al. 04] précise les propriétés attendues de ce service, et donne une vue haut niveau de l'algorithme proposé, sous forme textuelle et sous forme de pseudo-code pour certaines parties. Une implémentation est distribuée sous licence libre par les auteurs, et est donc disponible pour nos expérimentations.

Notre étude du protocole a comporté plusieurs étapes, incluant (1) une analyse de la spécification publiée dans l'article de référence, (2) une étape de rétro-conception du code source en UML, (3) la comparaison des algorithmes spécifié et implémenté, et (4) le test de l'implémentation vis-à-vis des propriétés attendues. Les trois premières étapes ont été faites

en collaboration avec l'Université Technologique de Budapest (BUTE - *Budapest University of Technology and Economics*), en particulier avec Zoltan Misckei, doctorant. Les expériences de test ont été entièrement conçues et réalisées de notre côté.

Nos travaux sur cet exemple ont d'abord fait l'objet d'un livrable du projet européen HIDENETS [HIDENETS D5.2], puis d'une publication internationale [Waeselynck et al. 07]. Pour présenter ces travaux, nous adoptons le canevas suivant : d'abord, nous introduisons le principe du GMP et les propriétés attendues. Puis, le paragraphe suivant porte sur les analyses de la spécification et de l'implémentation. Ensuite, nous présentons les expériences de test réalisées. Enfin, un dernier paragraphe fait le bilan des enseignements tirés de ce cas d'étude.

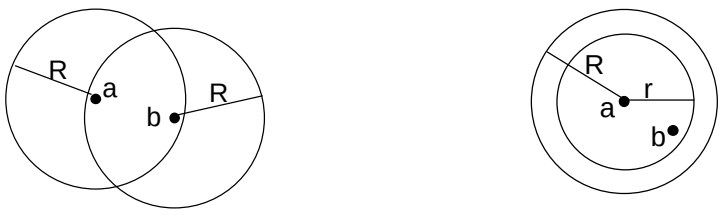
2. Présentation du GMP étudié

Dans cette partie, nous allons présenter le cas d'étude : principes généraux de ce GMP, propriétés attendues, vues haut niveau de l'algorithme spécifié et du code source disponible.

2.1. Principe du GMP

Le but de ce GMP est d'offrir un service d'appartenance de groupe qui gère les connexion/déconnexions induites par la mobilité, par exemple, quand un nœud mobile s'éloigne hors de portée de transmission. Chaque groupe est dirigé par un leader qui peut décider de fusionner (faire un *merge*) avec d'autres groupes, ou de diviser son groupe (faire un *split*). La décision est basée sur la notion de *Distance de Sécurité*. Le principe de la Distance de Sécurité est le suivant : si deux nœuds sont "suffisamment proches", il ne pourra pas y avoir de déconnexion induite par le mouvement pendant un certain temps. En particulier, les deux nœuds auront le temps d'effectuer une opération de scission de groupe avant d'être hors de portée de transmission.

Ce concept est illustré dans la Figure 12. Dans la Figure 12a, deux nœuds *a* et *b* sont à la limite de la portée de transmission *R*. S'ils essaient d'effectuer une opération de groupe (fusion ou scission), il se peut que cette opération ne se termine pas, du fait du déplacement d'un nœud hors de portée de l'autre. Pour garantir la délivrance des messages et la terminaison d'une opération de groupe, il faut que la distance entre a et b soit inférieure à un seuil *r* (Figure 12b), qui est la distance de sécurité. Le calcul de la distance de sécurité est basé sur différents paramètres comme la portée de transmission des nœuds mobiles, leur vitesse maximale et le temps nécessaire pour l'exécution d'une opération de groupe.



(a) (b)

Figure 12. Exemple de Distance de Sécurité

Un groupe est alors sûr si toute paire de nœuds est connectée via un chemin dans lequel tous les nœuds consécutifs sont à une distance de sécurité. Quand deux groupes peuvent

fusionner en un groupe sûr, ils doivent le faire (critère de fusion). Quand un groupe n'est plus sûr, il doit se scinder en plusieurs groupes (critère de scission). Notons que la notion de groupe sûr, avec un chemin sûr entre toute paire de nœuds, autorise la communication multi-sauts entre membres d'un même groupe.

Les auteurs du protocole posent des hypothèses fortes sur l'environnement dans lequel fonctionne leur protocole. Elles sont : pas de perte de messages, pas de défaillance de nœuds, latence de communication bornée, communication fiable, chaque nœud connaît exactement sa localisation physique à chaque instant. Ces hypothèses fortes permettent aux auteurs de se concentrer sur le problème des déconnexions induites par la mobilité, et d'ignorer toutes les autres menaces possibles. Afin de simplifier le calcul de la distance de sécurité, des hypothèses additionnelles sont également introduites : tous les nœuds ont la même portée de transmission, ils ont la même vitesse maximale et les communications entre nœuds sont bidirectionnelles. Par ces hypothèses, tous les nœuds ont une même distance de sécurité.

2.2. Propriétés attendues du GMP

Le service d'appartenance de groupe est spécifié par la définition de variables d'état locales à chaque nœud, et par les propriétés que ces variables d'état doivent satisfaire. Deux variables d'état sont :

- *group(p,t)* : l'identifiant du groupe d'un nœud *p* à un instant local *t*. L'identifiant du groupe est composé de l'identifiant du leader et d'un numéro de séquence qui croît à chaque changement de groupe,
- *mem(p,t)* : la vue locale des membres du groupe du nœud *p* à un instant local *t*.

Le numéro de séquence de l'identifiant de groupe permet de définir une relation d'ordre sur les groupes successifs auxquels le nœud *p* appartient. Un groupe est appelé *g* si son identifiant est *g*. Un groupe *g'* est un successeur du groupe *g* s'il existe un membre *p* de *g* qui quitte *g* pour rejoindre *g'*. On note *succ(g,p)* le successeur du groupe *g* pour le nœud *p*. D'une manière similaire, *pred(g,p)* indique le prédécesseur du group *g* pour le nœud *p*.

Le service d'appartenance du groupe est alors caractérisé par les huit propriétés présentées dans le Tableau 1, qui doivent être satisfaites lorsque les hypothèses du protocole sont vérifiées. On trouvera une description un peu plus détaillée de ces propriétés dans la section 2 de l'article [Huang et al. 04].

Afin de préparer le test du GMP, nous classons ces propriétés en deux catégories :

- les propriétés qui peuvent être vérifiées localement sur un nœud (AI, ML, VIG, JCG),
- les propriétés qui doivent être vérifiées par des relations entre au moins deux nœuds (AM, CVD, FCG, SCG).

Tableau 1 . Les propriétés du protocole

Propriété	**DEFINITION INFORMELLE**
Auto inclusion (AI)	Un noeud appartient toujours à sa propre vue du groupe. $p \in mem(p,t)$
Monotonicité locale (ML)	Sur chaque nœud, l'identifiant du groupe courant croît à chaque changement de groupe. $pred(g,p) < g < succ(g,p)$
Accord sur les membres de groupe (AM)	Si deux noeuds ont le même identifiant de groupe, ils ont la même vue du groupe. $group(p, t_p) = group(q,t_q) \Rightarrow mem(p, t_p) = mem(q,t_q)$
Vue initiale du groupe (VIG)	Un noeud est le seul membre de sa propre vue quand il est créé. $mem(p, t_{init}) = \{p\}$
Justification de changement de groupe (JCG)	Le successeur du groupe *g* pour le noeud *p* est soit un sur-ensemble strict, soit un sous-ensemble strict du groupe *g*.
Cohérence de vues à la délivrance de messages (CVD)	Si un noeud *p* envoie un message applicatif m_{pq} à un noeud *q* à une date *t* et *q* est dans mem(*p*, *t*) alors le message m_{pq} sera délivré à *q* à une date t_0, et mem(*q*, t_0) = mem(*p*, *t*).
Fusion conditionnelle des groupes (FCG)	Si deux groupes g_1 et g_2 satisfont le critère de fusion à un instant *T* et s'ils le satisfont assez longtemps (au moins pendant une constante de temps T_c) alors ils fusionneront en un seul groupe. Soit p un membre de g_1 et q un membre de g_2 : $\exists t_p, t_q \in [T, T + T_c], mem(q, t_q) = mem(p, t_p)$
Scission conditionnelle du groupe (SCG)	Un groupe se divise seulement si c'est nécessaire.

2.3. Algorithme du GMP

Pour satisfaire les propriétés précédentes, le fonctionnement du protocole décrit dans [Huang et al. 04] se décompose en deux étapes principales : (1) la découverte du groupe et (2) la reconfiguration des membres de groupes.

Dans la première étape, un nœud diffuse périodiquement sa localisation physique (message « hello » contenant ses coordonnées GPS) et reçoit celles des autres nœuds. Les messages « hello » permettent aux nœuds voisins de détecter l'arrivée ou l'éloignement d'un nœud. Ces informations seront transmises au leader du groupe qui pourra décider d'effectuer des opérations de fusion/scission (*merge/split*).

La deuxième étape est la plus importante. Elle consiste à décider et effectuer les opérations de fusion/scission, en tenant compte du critère de distance de sécurité. A partir des informations collectées dans l'étape précédente, un leader peut découvrir qu'il y a des groupes voisins candidats à une fusion. Dans ce cas, il initie la fusion avec ces candidats. Une fois

terminée la négociation de la fusion, qui détermine notamment un nouvel identifiant de groupe, tous les nœuds affectés reçoivent une notification formelle de la part du nouveau leader. Dans le cas d'une scission, lorsque le leader est informé de l'éloignement de certains membres de son groupe, il effectue une partition du groupe en sous-groupes sûrs, et notifie le changement à ses anciens membres. Dans les deux cas, fusion ou scission, une synchronisation est mise en œuvre pour marquer la fin de l'opération de groupe. Après réception d'une notification de changement de groupe, chaque nœud envoie un message de synchronisation (appelé message *flush*) à ses anciens pairs. Le nœud cesse alors d'émettre de nouveaux messages, jusqu'à réception de tous les messages *flush* de son ancien groupe. Ce n'est qu'après les avoir reçus que le nœud entre dans la nouvelle configuration de groupe. Il est alors prêt à traiter et émettre des messages estampillés par le numéro de séquence de cette configuration. La synchronisation par messages *flush* est ainsi destinée à assurer l'atomicité des opérations de groupe.

Une description détaillée de l'algorithme du GMP est donnée dans la section 3 de l'article [Huang et al. 04], avec le pseudo-code des principales fonctions.

2.4. Code source du GMP

Une implémentation du GMP est réalisée dans LIME, un intergiciel pour les systèmes mobiles [LIME]. Cette implémentation n'est pas triviale : chaque nœud se compose d'environ 4000 lignes de code de Java, contient 22 classes et implique 6 threads concurrents. Nous pouvons constater que ce n'est pas un exemple jouet, et qu'il présente un réel intérêt du point de vue de la vérification.

3. Analyse du GMP

Dans cette partie, nous allons présenter les étapes d'analyse que nous avons effectuées en préliminaire au test du service d'appartenance de groupe. Ces analyses préliminaires ont été conduites en collaboration avec l'Université Technologique de Budapest (BUTE), et ont porté à la fois sur la spécification et le code source.

3.1. Analyse de la spécification

Typiquement, une revue de spécification a pour but de révéler des incohérences, des omissions, ou des ambiguïtés. Dans notre cas, la spécification est le matériel publié dans l'article [Huang et al. 04]. Comme indiqué précédemment, il contient une description textuelle des propriétés attendues et du fonctionnement du GMP, ainsi qu'une description plus précise sous forme de pseudo code.

3.1.1. Analyse des propriétés

Les huit propriétés dans le Tableau 1 fournissent les exigences de haut niveau du protocole. Leurs définitions détaillées ont été soigneusement examinées.

Nous avons remarqué que la définition de la propriété *Cohérence de vues à la délivrance de messages (CVD)* ne concerne que les listes de membres entre deux nœuds. Elle n'exige pas que les identifiants de groupe soient identiques. Selon cette propriété, des opérations successives de fusion et scission pourraient théoriquement se produire entre les événements d'envoi et de réception du message.

La formulation de deux propriétés, *Fusion conditionnelle des groupes (FCG*) et *Scission conditionnelle du groupe* (*SCG*), est jugée insuffisante dans une perspective de vérification. Dans la définition de la propriété FCG, la constante T_c durant laquelle le critère de fusion reste satisfait n'est pas définie de façon constructive. De plus, la propriété FCG n'est définie que pour deux groupes. Elle ne précise pas explicitement le cas où plus de deux groupes sont impliqués dans une fusion. De façon similaire, la conformité à la propriété SCG ne peut être vérifiée en pratique, car il n'existe pas de formulation précise pour cette propriété.

Dans le cadre de nos expériences de test, l'oracle de test ne pourra pas vérifier la satisfaction des propriétés FCG et SCG. Par contre, les six autres propriétés du Tableau 1 pourront être considérées.

Il est à noter que la spécification formelle de propriétés telles que FCG et SCG est intrinsèquement difficile. En effet, les critères de fusion et scission mettent en jeu des relations spatio-temporelles entre nœuds mobiles. Comment déterminer des bonnes abstractions, et comment choisir un formalisme approprié de spécification, sont des problèmes ouverts.

3.1.2. Analyse du calcul de la Distance de Sécurité

Le concept clé de *Distance de sécurité*, qui est au cœur du protocole, est également problématique. Intuitivement, le pire cas de calcul implique la considération combinée (1) du mouvement des nœuds et (2) de l'exécutions répartie des opérations de groupe. Deux définitions sont données pour la distance de sécurité dans l'article [Huang et al. 04] (respectivement dans les sections 3 et 5). Nous les reproduisons ci-dessous. La formulation de gauche est une définition générale de la distance de sécurité (donnant un seuil abstrait *r*), tandis que celle de droite raffine la définition générale afin de prendre en compte la spécification détaillée du GMP (donnant une valeur concrète d_s).

$r \leq R - 2v(t + t')$	$d_s = R - 2V_{max}(t_u + 7t_d)$
R: portée de transmission v : vitesse maximale t: temps pour une tâche de niveau applicatif t': temps nécessaire pour une opération de groupe	V_{max}: vitesse maximale t_u: période d'émission de "hello" t_d: latence maximale du réseau

Il semble que les deux formulations soient incohérentes. A droite, la partie de calcul notée ($t_u + 7t_d$) comprend : le temps nécessaire pour effectuer une scission ($2t_d$, car deux messages sont nécessaires à la réalisation de l'opération), l'incertitude sur la localisation des nœuds voisins ($t_u + t_d$), et l'éventualité d'une fusion en cours qui ne pourrait pas être interrompue ($4t_d$). Ceci correspond à la partie *t'* dans la formulation de gauche. La partie *t* dans la formulation de gauche, qui prend en compte l'application utilisant le service de groupe, est donc manquante.

La définition et le calcul de t_d sont également ambigus. Dans la section 1 de l'article, t_d est mentionné comme la latence du réseau, alors que dans la section 5, il s'agit la latence du réseau plus les délais induits par les files d'attente. Le calcul des délais d'attente dans les files est problématique, car il dépend des autres messages et synchronisations en cours (par exemple, messages bloqués lorsque le nœud attend les messages *flush* de ses pairs).

Nous pouvons constater que le calcul de la distance de sécurité est un problème très complexe, qui aurait probablement nécessité plus d'attention. Pour nos expériences de test, nous avons retenu la formulation de droite, avec une valeur arbitraire – mais pessimiste – pour t_d (1 seconde, voir §4.2)

3.1.3. Analyse de la spécification détaillée du GMP

Les descriptions en pseudo-code sont très utiles pour comprendre le fonctionnement du protocole proposé. Cependant, nous avons identifié quelques points nécessitant des clarifications.

Les flots de contrôle et de synchronisation ne sont pas indiqués. Il n'est pas précisé quelles fonctions peuvent être exécutées en parallèle, et quelles sont les interruptions possibles. Par exemple, est-il possible de traiter une mise à jour de localisation pendant une opération de fusion ? En l'absence d'information sur le flot de contrôle, l'atomicité des opérations de changement de groupe est difficile à établir. En particulier, elle n'est pas assurée par les pré-conditions exprimées dans le pseudo code des fonctions (par exemple, aucune fonction relative à la scission de groupe ne comporte de pré-condition exigeant l'absence de fusion en cours). Le mécanisme de synchronisation par messages *flush* n'est pas explicitement décrit, il est "caché" dans une fonction *ClearOldChannels()* supposée donnée. En fait, les seules informations dont nous disposons sur ce mécanisme sont des explications en langage naturel, et des exemples graphiques de scénario. Ce matériel est à notre avis insuffisant.

3.2. Analyse de l'implémentation

L'analyse du code implémenté a été réalisée avec un double objectif : (1) l'étude de la conformité vis-à-vis de la spécification détaillée (notamment le pseudo-code), et (2) la recherche d'éventuelles fautes dans l'implémentation.

3.2.1. Comparaison avec la spécification détaillée

Pour faciliter la compréhension du code Java, notre collègue Zoltan Miskei en a effectué une rétro-conception sous forme de modèles UML [Micskei et al. 06]. Ses analyses montrent que le code Java ne suit pas le pseudo code inclus dans la spécification. Il a essayé de faire correspondre les types de messages entre les deux versions, mais sans succès dans la plupart des cas.

Le problème vient de ce que les opérations implémentées procèdent très différemment des opérations spécifiées. Prenons l'exemple d'une fusion. Comme nous l'avons vu, dans la spécification, cette opération procède en deux phases : une phase de négociation et une phase de réalisation du changement (*request – commit*). Dans l'implémentation, ces deux phases n'existent pas et il n'y a aucune possibilité de refuser un changement de groupe. Après s'être informé du leader d'un groupe voisin, et de la connectivité des membres du groupe, le nouveau leader effectue directement la fusion et le notifie à tous les membres. Il n'y a pas de mécanisme de synchronisation par messages *flush* pour clore l'opération, ni de numéro de séquence dans l'identifiant de groupe. On trouvera dans [Micskei et al. 06] un exemple de diagramme de séquence UML montrant un scénario de fusion typique pour l'implémentation. L'analyse de l'implémentation révèle également des différences dans la surveillance de la topologie des nœuds, qui est plus décentralisée que dans la spécification.

L'article de référence [Huang et al. 04] mentionne l'absence des numéros de séquence et des messages *flush* dans l'implémentation disponible. La justification donnée est une séparation des préoccupations liées au protocole et aux applications utilisatrices du service de groupe. D'après les auteurs, chaque application doit pouvoir choisir ses mécanismes de synchronisation pour l'atomicité des opérations. Notamment, une application non critique peut choisir d'utiliser le service de groupe sans garantie d'atomicité.

A l'issue de la comparaison du code et de la spécification, nous sommes arrivés à la conclusion suivante : bien que les deux versions du GMP (celle dans [Huang et al. 04] et celle dans l'implémentation) soient basées sur le même concept de distance de sécurité, et soient caractérisées par les mêmes propriétés attendues, elles doivent en réalité être considérées comme deux protocoles différents.

Une conséquence pour nous est que la description du pseudo code ne peut pas être utilisée comme référence pour les expériences de test. En particulier, l'oracle de test doit se baser uniquement sur les propriétés attendues. Cependant, la vérification de deux de ces propriétés – la monotonicité locale (ML) et l'accord sur les membres du groupe (AM) – pose problème du fait de l'absence de numéro de séquence dans l'identifiant de groupe. Nous avons décidé de modifier très légèrement l'implémentation en introduisant ce numéro de séquence. Notons que la modification ne change pas la structure et la logique de l'implémentation : elle porte uniquement sur le calcul d'un nouveau numéro à chaque changement de groupe, et n'ajoute aucune synchronisation au niveau global.

Afin de faciliter les analyses de scénarios qui seront présentées dans les paragraphes suivants, le Tableau 2 résume les principaux messages échangés dans la version implémentée du GMP.

Tableau 2. Principaux messages échangés dans le GMP implémenté

Message	Objectif
Hello	Diffuser ses coordonnées aux nœuds à portée de transmission
SPGetLeader	Demander l'identifiant du leader d'un nœud
SPLeaderAddress	Répondre à la demande SPGetLeader avec l'identifiant du leader de son groupe.
SPGroupInfo	Envoyer la connectivité actuelle de son groupe
SPGroupChange	Notifier aux membres la nouvelle composition du groupe après un changement (*merge* ou *split*).

3.2.2. Recherche de fautes dans l'implémentation

De façon générale, le fait que l'implémentation :

- ne permette pas de refuser un changement de groupe,
- n'offre aucun mécanisme de synchronisation pour clore les opérations de changement de groupe,

nous paraît très problématique vis-à-vis de la satisfaction des propriétés du service d'appartenance de groupe.

Nous avons cherché à identifier des scénarios qui violent les propriétés attendues. Nous en présentons un ici à titre d'exemple.

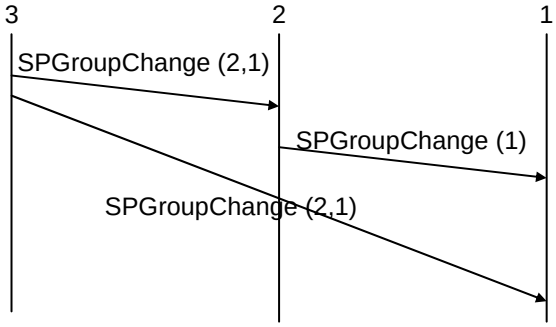


Figure 13. Exemple d'ordonnancement des messages

La Figure 13 montre un scénario potentiellement dangereux du fait de l'absence de synchronisation par messages *flush*. Il concerne un problème d'ordonnancement global des messages pour deux scissions successives.

Au début de ce scénario, les trois nœuds 1, 2, 3 sont dans le même groupe et le nœud 3 est le leader. Le nœud 3 détecte qu'il doit se séparer de ses pairs (nœuds 1 et 2). Il décide d'effectuer une scission et envoie l'information du nouveau groupe aux nœuds 1 et 2. Mais, du fait de délais de communication, le nœud 1 recevra cette information plus tardivement que le nœud 2. Avant qu'il l'ait reçue, le nœud 2 (qui est le leader du nouveau groupe composé des seuls nœuds 1 et 2), détecte également qu'il doit se séparer du nœud 1. Il effectue donc une nouvelle scission et en informe le nœud 1. Regardons maintenant le comportement du nœud 1. Ce nœud reçoit et traite le message qui vient du nœud 2 avant celui qui vient du nœud 3. Clairement, le traitement des changements de groupe dans un ordre différent sur les nœuds 1 et 2 est susceptible de poser problème (par exemple, violation de la cohérence de vue à la délivrance de messages). Dans la spécification, ce scénario est impossible car le nœud 2 devrait attendre la clôture de la première scission avant de lancer la deuxième.

Les scénarios de violation imaginés lors de cette étape d'analyse ne sont que potentiels, car raisonner sur le comportement global du GMP implémenté s'avère complexe. Ceci justifie l'intérêt d'une étude plus complète au moyen d'expériences de test.

4. Test du GMP

Les expériences de test visent à déterminer si l'implémentation satisfait les propriétés de haut niveau du service de groupe, récapitulées dans le Tableau 1. Comme expliqué précédemment (§ 3.1.1), nous ne disposons pas de formulation précise de deux propriétés, FCG et SCG. Par conséquent, le test ne porte que sur les six autres propriétés. De plus, nous avons été amené à introduire une modification mineure dans le code source (ajout d'un numéro de séquence dans l'identifiant de groupe) pour permettre la vérification de certaines de ces propriétés (ML et AM).

Ce paragraphe présente les expériences réalisées. Nous commençons par une description de la plate-forme de test, et des paramètres retenus pour configurer le système sous test. Ensuite, nous présentons les résultats observés.

4.1. Plate-forme de test

Le test requiert de faire tourner le GMP sur un ensemble de nœuds mobiles et d'observer le résultat. Notre plate-forme expérimentale (Figure 14) est entièrement basée sur les moyens de prototypage offerts dans la distribution du code source, et utilisés par les auteurs pour effectuer la démonstration de leurs travaux.

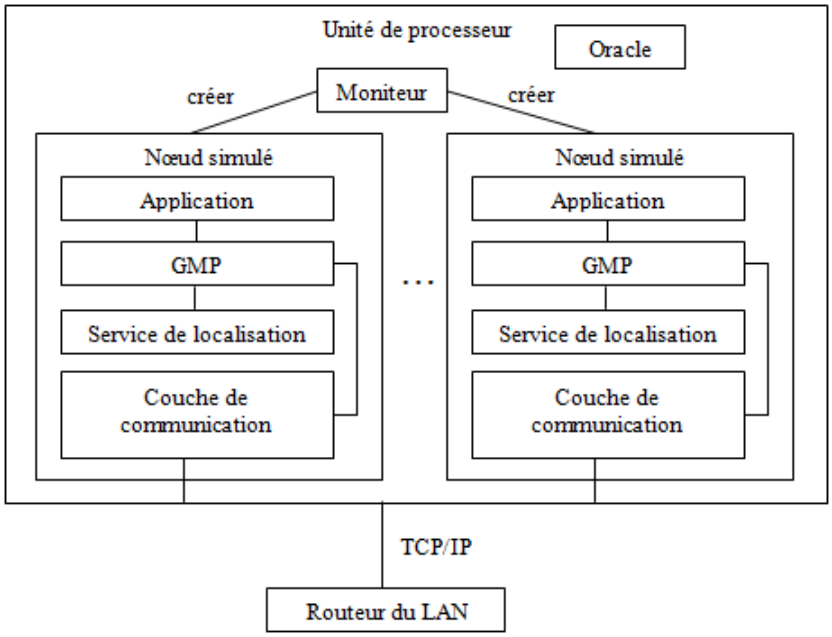


Figure 14. Architecture de la plate-forme de test du GMP

Il s'agit d'une plate-forme centralisée. Physiquement, tous les nœuds résident en fait dans la même unité de calcul. Ils communiquent entre eux via le réseau filaire, en utilisant TCP/IP. Ce choix nous était imposé par les interfaces de communication disponibles dans le code source. Au sommet de l'architecture, un moniteur de test est chargé de créer un ensemble de nœuds. Sur chaque nœud, l'instance du GMP exécutée est connectée à deux bouchons (*stubs*):

- un bouchon d'application, qui reçoit une notification du GMP à chaque changement du groupe. Ce bouchon génère également un trafic applicatif par émission de messages à tous les membres du groupe courant, à des instants aléatoires.
- un bouchon pour le service de localisation, qui fournit au GMP des coordonnées aléatoires de position du nœud.

Tous les événements survenant au niveau d'un nœud, ainsi que leur date, sont enregistrés dans un fichier *log*. A la fin d'une expérience, l'oracle de test collecte les fichiers log de tous les nœuds, et effectue une analyse post-mortem pour détecter d'éventuelles violations de propriétés. Nous ne détaillons pas ici l'implémentation des vérifications relatives aux six propriétés testées. Elle se base sur les formulations données dans le Tableau 1, et n'a pas présenté de difficulté particulière.

Il est à noter que les propriétés sont testées sous les hypothèses de GMP (par exemple, pas de défaillance de nœuds, la communication entre nœuds est fiable…). La plate-forme n'a donc pas besoin d'émuler certains aspects complexes de la réalité, qui sont hors hypothèses.

Cependant, nous sommes conscients des limitations de cette plate-forme, qui seront analysées ultérieurement (§5.2.1).

4.2. Paramétrage des expériences de test

Les configurations expérimentales utilisées pour tester le GMP dépendent de paramètres que nous classons en deux catégories :

- des paramètres dimensionnants pour le GMP, comme la vitesse maximale des nœuds ;
- des paramètres de test liés à l'activation du GMP, par exemple, le modèle de mobilité utilisé pour générer les coordonnées des nœuds.

Le Tableau 3 fournit une vue globale des valeurs des paramètres utilisées dans nos expériences.

Tableau 3. Paramètres des expériences de test

Paramètres du GMP	**Valeur**
Portée de transmission	300 m
Vitesse maximale	10 m/s
Période de mise-à-jour de la localisation	1 s
Latence du réseau	1 s
Paramètres de test	**Valeur**
Nombre de noeuds	Aléatoire dans [3,15]
Date d'initialisation d'un noeud	Aléatoire dans [0, 1000] millisecondes
Position initiale	x,y: aléatoire dans [-150,150] m
Modèle de mobilité	Mouvement aléatoire à vitesse maximale
Messages applicatifs	Temps aléatoire entre deux messages [0,10000] millisecondes
Durée d'une exécution	5 minutes

En appliquant la formule de calcul de la distance de sécurité (§3.1.2), les valeurs dimensionnantes produisent une distance de sécurité de 140m.

En ce qui concerne les paramètres de test, notons que nous ne cherchons pas à activer un ensemble de cas de test prédéfinis. Le comportement du GMP (opérations de fusion/scission) est contrôlé indirectement via les positions des nœuds générées. Nos expériences peuvent être vues comme une forme de test passif, qui observe le comportement du GMP soumis à une activité synthétique. L'activité inclut le mouvement des nœuds et le trafic des applications qui utilisent le service de groupe.

Une raison pour laquelle nous avons choisi d'effectuer un test passif est que nous ne disposons pas d'un modèle formel pour dériver des cas de test.

4.3. Résultats de test

Nous avons réalisé une expérimentation avec 100 exécutions du GMP. Nous en avons trouvé 82 qui violent au moins une propriété. Durant les cinq minutes que dure chaque exécution, un grand nombre de violations peuvent être observées. Il n'était pas possible de les analyser toutes manuellement. Nous avons alors extrait un sous-ensemble de scénarios de défaillance, en analysant la première violation d'une exécution, puis en cherchant des cas de défaillance ultérieurs violant de nouvelles propriétés. Au total, 164 scénarios ont été extraits. Comme montré dans le Tableau 4, le problème diagnostiqué est toujours la non-atomicité des opérations de changement de groupe. Une fusion peut être entrelacée à une scission ou à une autre fusion. Le problème induit une grande variété de comportements erronés de l'implémentation, allant de la violation d'une des propriétés ML, JCG, AM ou CVD, à des cas de violations multiples.

Tableau 4. Analyse de 164 cas de violation

Violation	**Fusions et scissions concurrentes**	**Fusions concurrentes**
ML	5	38
JCG	10	19
AM	10	0
CVD	18	22
2 propriétés violées	18	16
3 propriétés violées	6	2
# de scénarios	67	97

La Figure 15 montre un exemple de scénario de fusion et scission concurrentes, conduisant à une violation multiple.

Pour faciliter la compréhension du scénario, nous plaçons des annotations à différents points des lignes de vie des nœuds. Notamment, certaines annotations indiquent la vue locale du groupe, le leader étant identifié par le signe '*'. Ainsi, on peut voir que le scénario débute avec deux groupes, un singleton avec le nœud 1, et un autre groupe avec les nœuds (2, 3, 4, 5, 6, 7*) dont le leader est le nœud 7.

Le nœud 7 détecte que son groupe n'est plus sûr. Il exécute donc une scission qui divise le groupe actuel en deux : (2, 5*) et (3, 4, 6, 7*) dont les nouveaux leaders sont respectivement les nœuds 5 et 7. Il envoie une série de messages à ses pairs pour notifier le changement de groupe. Dans la Figure 15, nous ne représentons que le message SPGroupChange envoyé au nœud 5.

En parallèle avec cette scission, le nœud 1 découvre le nœud 5 comme un nouveau voisin à distance de sécurité et commence une fusion. Le nœud 1 ajoute le nœud 5 dans son graphe de connectivité et demande l'adresse du leader du nœud 5. Quand le nœud 5 reçoit cette requête, il est encore dans son ancien groupe (avant la scission). Il envoie donc l'identifiant du nœud 7 comme son leader. Le nœud 1 décide qu'il ne sera pas le nouveau leader après la fusion (le leader choisi est toujours celui ayant le plus grand identifiant, c'est-à-dire ici 7). Le

nœud 1 envoie alors son graphe de connectivité au nœud 7. Le nœud 7 doit accepter et effectuer la fusion, qui produit un nouveau groupe (1, 3, 4, 5, 6, 7*).

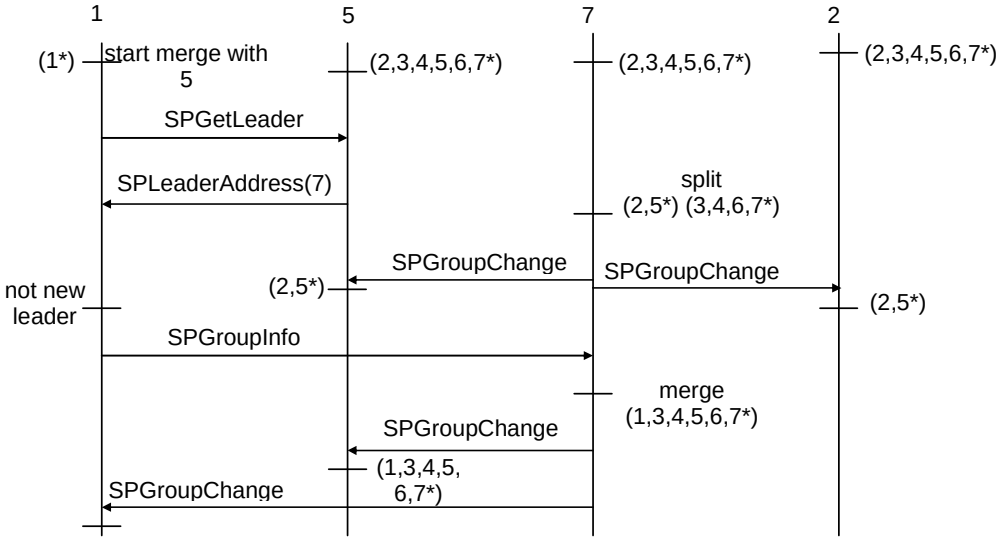


Figure 15. Exemple de fusion et scission concurrentes

Nous pouvons constater plusieurs problèmes dans ce scénario.

- Le nœud 5 est notifié de deux changements de groupe consécutifs (2, 3, 4, 5, 6, 7*) → (2, 5*) → (1, 3, 4, 5, 6, 7*). Comme le dernier groupe n'est pas un sur-ensemble du précédent, la propriété JCG est violée.
- A la fin du scénario, le nœud 2 croit qu'il est encore dans le groupe (2, 5*). Une violation de la propriété CVD est observée via un message applicatif (non montré ici).
- Plus généralement, le fait que le nœud 5 soit ré-intégré dans le groupe du nœud 7, alors qu'il n'est plus à distance de sécurité, soulève le problème de la protection fournie par le concept de distance de sécurité.

De nombreuses variantes d'entrelacements de fusions et scissions ont été observées. De même, les entrelacements de fusions concurrentes ont donné lieu à différents scénarios. Les détails des scénarios varient selon les mouvements sous-jacents des nœuds et selon l'ordre des événements de communication. On voit là l'intérêt du test. Bien que l'absence d'atomicité ait été identifiée comme problématique lors de la rétro-conception du code source, il aurait été bien difficile d'imaginer ces différents scénarios – et la variété des comportements défaillants associés – par un raisonnement seulement basé sur notre compréhension du protocole.

5. Discussion à l'issue de l'étude de cas

Nous allons maintenant tirer le bilan des analyses et expériences de test conduites au cours de l'étude de cas. Dans la discussion, nous distinguons d'une part les retours obtenus sur le comportement du GMP, et d'autre part les enseignements plus généraux que nous pouvons en tirer pour notre problématique.

5.1. Conclusions sur le GMP étudié

Le GMP étudié est représentatif d'un problème classique en algorithmique distribuée : l'obtention d'un accord sur les membres d'un groupe. Cependant, le contexte mobile particularise ce problème. Dans un cadre non mobile, on observe typiquement de longues périodes de stabilité du système, et quelques périodes d'instabilité comparativement courtes [Cristian et Schmuck 95]. Les systèmes mobiles sont susceptibles d'exhiber une instabilité bien plus forte, particulièrement dans un environnement réseau de type ad hoc où les possibilités de communication varient avec la localisation et le mouvement de tous les nœuds voisins.

Pour prendre en compte la mobilité des nœuds, les auteurs de [Huang et al. 04] ont introduit la notion de *Distance de Sécurité*, qui est au cœur de leur GMP. Malheureusement, d'après nos analyses, cette notion s'avère difficile à gérer. Ceci est dû au fait que la détermination de la distance de sécurité dépend de la capacité à exhiber un scénario de pire cas. L'obtention du pire cas est complexe, car on doit déterminer toutes les configurations qui peuvent affecter le comportement du protocole.

Notre étude a couvert plusieurs étapes, de l'analyse des propriétés attendues au test de l'implémentation proposée. Chacune a révélé des problèmes. Notamment, l'atomicité des opérations de changement de groupe a été une question récurrente. Dans le code source, cette question n'est pas traitée. Les auteurs proposent que les mécanismes pour garantir l'atomicité soient choisis et mis en œuvre au niveau applicatif. Des applications avec de faibles exigences de cohérence de groupe pourraient utiliser le protocole GMP tel quel, sans garantie d'atomicité. Toutefois, nos résultats de test montrent qu'il ne s'agirait pas seulement d'accepter quelques défaillances occasionnelles : l'implémentation a en fait été incapable de délivrer le service attendu dans la plupart des cas. Clairement, la non-atomicité compromet l'utilité même du protocole.

La correction de ce problème dans l'implémentation ne serait pas simple. Il faudrait des changements majeurs dans la définition des phases du protocole, ainsi que dans les mécanismes de synchronisation locaux des *threads* au sein de chaque nœud (le détail sur le fonctionnement des *threads* est donné dans [Micskei et al. 06]). Le pseudo-code donné dans le papier [Huang et al. 04] pourrait servir de base pour retravailler l'implémentation. Cependant, comme l'ont montré nos analyses de la spécification détaillée, il reste encore des points à clarifier. Ces points concernent les flots de contrôle et de synchronisation internes à un nœud, et les mécanismes de synchronisation globale par messages *flush*. Quelle que soit la solution retenue, il faut souligner que l'atomicité doit être assurée au niveau du GMP. Le problème ne peut pas être résolu au niveau applicatif.

5.2. Enseignements tirés de l'étude de cas

Notre étude a fourni des retours sur la spécification et l'implémentation du GMP. Cependant, il est important de noter que notre objectif premier n'était pas tant de trouver des défauts dans un prototype de recherche. Il était plutôt de s'attaquer à un exemple non-trivial de système mobile et d'obtenir une vue concrète des problèmes de test associés. Dans cette perspective, nous pensons que le GMP a été un exemple difficile mais intéressant à traiter. Nous livrons ci-dessous les enseignements que nous en tirons.

5.2.1. Discussion sur la plate-forme de test

Rappelons que le test de systèmes mobiles soulève un problème technologique de mise en œuvre : comment les plates-formes de test peuvent-elles prendre en compte la technologie sans fil et la mobilité des dispositifs ? Dans le chapitre précédent, nous avons constaté qu'une grande partie des activités de test doit être réalisée en utilisant des moyens d'émulation/simulation. Pour le test du GMP, nous sommes allés au plus simple. Notre plate-forme de test s'est basée sur les moyens de prototypage déjà offerts dans la distribution du code source. L'architecture correspondante (Figure 14) n'inclut ni simulateur de réseau, ni simulateur de contexte. Le réseau filaire est utilisé, et les bouchons (*stubs*) mis pour le service de localisation sur chaque nœud offrent un moyen rudimentaire de produire des données contextuelles (coordonnées de position). Cette approche s'est avérée suffisante pour avoir une idée rapide du comportement de l'implémentation, mais elle présente des limitations notables.

Les expériences de test n'ont pas pu produire des instances concrètes du scénario imaginé lors de l'analyse du code (voir la Figure 13), qui illustrait un problème d'ordonnancement global de messages. Ceci est dû à une limitation de la plate-forme, qui n'offre pas de moyens de contrôler les délais de communication. Il n'a pas été possible de provoquer des situations où des messages de changement de groupe sont reçus dans un ordre différent par les nœuds, comme dans le scénario imaginé. Ceci n'est pas satisfaisant, car on peut s'attendre à ce que les problèmes d'ordonnancement global soient fréquents dans les systèmes mobiles. En particulier, le GMP est censé prendre en compte la communication multi-sauts entre membres d'un groupe (rappelons qu'un groupe est sûr si toute paire de nœuds est connectée via un chemin dans lequel les sauts consécutifs ne dépassent pas la distance de sécurité). Il semble alors raisonnable de supposer que les délais de communication varient en fonction de la topologie du système. Selon les chemins suivis, rien ne garantit que l'ordre des réceptions au niveau d'un nœud respecte des relations de causalité au niveau global. Ces situations ne peuvent pas être testées dans l'architecture actuelle.

De manière générale, la plate-forme ne nous permet pas de contrôler la délivrance des messages en fonction d'informations de localisation. C'est une limitation majeure vis-à-vis du concept de distance de sécurité. Par exemple, nous voudrions pouvoir observer la perte de messages chaque fois un nœud s'éloigne hors de portée de son groupe sans avoir terminé une opération de scission. Mais dans la plate-forme, il n'existe pas de possibilité pour adapter la couche de communication à la topologie du système. Les scénarios qui mettent en cause le concept de distance de sécurité ne peuvent être identifiés que par une analyse post-mortem (par exemple, on constate que le scénario de la Figure 15 produit la ré-intégration du nœud 5).

L'utilisation d'un simulateur de réseau aurait évité ces problèmes.

En ce qui concerne les données contextuelles, chaque nœud reçoit les coordonnées aléatoires générées par son service de localisation. La plate-forme n'offre pas de contrôle centralisé des mouvements des nœuds. Ceci a également été identifié comme une limitation. Tout d'abord, il ne serait pas possible de gérer les positions relatives des nœuds, ce qui serait un obstacle vis-à-vis d'un test actif cherchant à provoquer des configurations spatiales spécifiques. Ensuite, il ne serait pas possible de simuler des modèles de mobilité complexes tels que ceux décrits dans le chapitre précédent, impliquant par exemple de gérer les mouvements des nœuds dans une zone physique avec des obstacles.

Nous pensons que les points soulevés lors de l'étude cas :

- être capable de gérer les délais et pertes de messages en relation avec le contexte spatial des nœuds,
- avoir un contrôle centralisé du mouvement relatif des nœuds et de la production de données contextuelles nécessaires à l'application testée,

sont représentatifs de besoins récurrents lorsque l'on teste des systèmes mobiles. Il est donc souhaitable que les plates-formes de test intègrent des outils pour prendre en charge ces aspects.

Si l'on se réfère aux exemples d'architectures expérimentales que nous avons trouvées dans la littérature (paragraphe 5.4 du chapitre 1), on constate que la plupart d'entre elles suivent le modèle montré en Figure 16. Dans ce modèle, le simulateur de réseau a pour objectif d'imiter le comportement du réseau réel pour la communication entre les nœuds du système. Un simulateur de contexte est utilisé pour gérer l'environnement géographique et produire les données contextuelles des nœuds, par exemple, les données de localisation. L'application exploite ces données pour déterminer les actions à effectuer. Une certaine connaissance du contexte est aussi nécessaire au simulateur réseau afin de déterminer les valeurs des paramètres qui gouvernent le comportement du réseau.

Notons que la Figure 16 est schématique. Les architectures utilisées dans [Morla et Davies 04] et [Schroth et al. 05] (cf. chapitre 1, §5.4.1-2) distinguent bien les trois catégories de composant montrés. Mais certains outils peuvent offrir des fonctionnalités qui recouvrent plusieurs catégories de composants. Par exemple, l'émulateur de topologie développé à l'Université d'Åalborg [Nickelsen et al 06] (cf. chapitre 1, §5.4.3) peut être vu comme un simulateur réseau qui inclut des parties relatives à la simulation de contexte et au support applicatif.

De façon générale, les outils de simulation ont principalement été développés pour permettre des évaluations expérimentales (par exemple, des évaluations de performance). Notre expérience indique qu'ils sont également pertinents pour servir un objectif de vérification. Une plate-forme trop simple, telle que celle utilisée pour le GMP, peut s'avérer insuffisante pour provoquer et observer certains cas de défaillance.

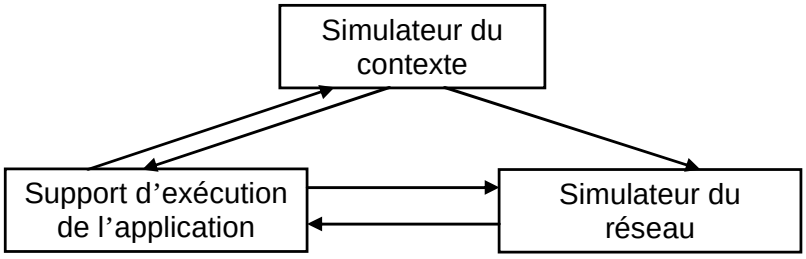


Figure 16. Modèle générique d'une plate-forme de test

5.2.2. Modélisation des interactions dans les systèmes mobiles

Comme nous l'avons souligné dans le chapitre 1, il n'existe pas de cadre de formalisation bien établi pour faciliter les activités de Vérification & Validation (V&V) dans le cas de systèmes mobiles. Dans l'exemple du GMP, le travail de rétro-conception du code source a montré que l'UML standard pouvait être suffisant pour représenter un nœud isolé, mais qu'il

était beaucoup plus difficile de capturer le comportement et la structure au niveau du système global [Waeselynck et al. 07].

Les diagrammes de séquences UML se sont avérés la vue la plus utile pour représenter le comportement au niveau système. Ces diagrammes ont constitué un support privilégié lors des étapes d'analyse préliminaire. Ils ont été utilisés pour illustrer les différences entre la spécification et l'implémentation du protocole, ainsi que pour identifier des scénarios de violations potentielles de propriétés. La représentation graphique de scénarios a également été cruciale pour interpréter les résultats de test. Les diagrammes construits manuellement à partir des traces d'exécution (par exemple, la Figure 15) nous ont permis d'acquérir une meilleure compréhension des cas de violation observés. Sans ce support graphique, il aurait été difficile d'obtenir une vue claire des entrelacements de messages avant la défaillance.

Toutefois, nous avons constaté que la représentation classique des scénarios en termes d'entrelacements de messages est insuffisante pour caractériser les cas de violation. En essayant d'interpréter le comportement observé, nous avons dû compléter les diagrammes de scénarios par des commentaires informatifs tels que "le nœud 7 détecte que son groupe n'est plus sûr", ou "le nœud 1 découvre le nœud 5 comme un nouveau voisin".

En fait, ces commentaires font appel aux notions suivantes :

- Il existe une configuration spatiale sous-jacente, où les nœuds sont – ou ne sont pas – à une distance de sécurité.
- Les nœuds reconnaissent les changements de configuration par réception de messages 'hello' des autres nœuds à portée de transmission.

Les changements de configuration et les messages 'hello' n'apparaissent pas explicitement dans les scénarios obtenus à partir des phases d'analyse ou de test. Cela est dû au fait que la topologie spatiale n'est pas considérée par les langages de scénario, et au fait que ces langages n'offrent pas de primitives pour représenter la communication par diffusion dans le voisinage. Or, les cas de violation observés durant le test dépendent de chaînes causales telles que : quel est le premier nœud envoyant un message 'hello' après un changement de configuration spatiale ? Quels sont les nœuds récepteurs du message ? Comment réagissent-ils à ce message selon leur position relative à l'émetteur, et comment leurs réactions s'entrelacent-elles au niveau global ? Actuellement, seul ce dernier type d'éléments apparaît dans la vue graphique des scénarios.

Notre conclusion est que la modélisation de scénarios est très utile dans le cadre d'activités de V&V de niveau système, mais que les langages existants doivent être étendus pour mieux prendre en compte le contexte mobile. En particulier, la notion de configuration spatiale nous paraît devoir occuper une place centrale, et les primitives de communication devraient inclure la diffusion radio dans le voisinage.

5.2.3. Des stratégies passives aux stratégies actives de test

Nos expériences consistaient en un test passif du GMP. Les données générées ne visaient pas à couvrir des objectifs de test. Leur rôle était simplement de fournir une activité de fond pour l'observation du comportement du protocole. Il se trouve que cette approche a été suffisante pour observer un grand nombre de cas de défaillance. Un point à souligner est que la diversité des comportements défaillants était due non seulement aux différentes possibilités d'entrelacement de messages, mais également à la variété des configurations spatiales de

noeuds. Dans l'optique d'une stratégie active de test, une première conclusion est que cet aspect spatial devrait être explicitement pris en compte dans la sélection des cas de test. Un test se limitant à un petit nombre de configurations pourrait être très insuffisant pour vérifier un système mobile. Comment déterminer les classes de configurations à couvrir pour une application donnée reste cependant un problème ouvert. Ce problème est, à notre avis, lié au problème plus général de la modélisation des systèmes mobiles.

Supposons maintenant qu'un ensemble de cas de test soit sélectionnés, éventuellement en se basant sur la seule expertise du testeur. Par exemple, à l'issue des analyses préliminaires du GMP, nous avons identifié des problèmes potentiels d'atomicité. Une stratégie évidente de test serait de considérer différentes configurations spatiales induisant des opérations concurrentes de fusion et de scission. Ainsi, le scénario de la Figure 15 aurait pu être le résultat d'un test cherchant à couvrir le cas où deux nœuds (2 et 5 dans l'exemple) s'éloignent de leur ancien groupe (de leader 7) en s'approchant d'un autre groupe (de leader 1).

L'implémentation d'un tel cas de test requiert de gérer les trajectoires d'au moins quatre nœuds de manière à ce que ce changement de configuration apparaisse. Le cas n'est pas nécessairement difficile à activer au moins une fois lors d'une simulation aléatoire (de fait, nos expériences ont produit une grande variété de configurations spatiales successives). Mais produire ce cas de façon déterministe, en réglant *a piori* les mouvements des nœuds, peut s'avérer fastidieux. Notons que, pour le GMP, les contraintes à respecter ne sont pas très compliquées (la vitesse des nœuds est bornée). D'autres applications peuvent nécessiter de considérer des contraintes de mobilité plus complexes. De plus, les données contextuelles du GMP sont restreintes aux coordonnées physiques, tandis que dans le cas général, on peut avoir à produire un ensemble de données plus riche, avec là encore des contraintes à respecter selon la localisation des nœuds.

On peut donc s'attendre à ce que l'implémentation de cas de test soit plus complexe pour les systèmes informatiques mobiles que pour les systèmes distribués traditionnels, de par le besoin d'instancier les données contextuelles de façon appropriée. Des solutions automatisées sont souhaitables, notamment en s'aidant d'outils de simulation de contexte.

6. Conclusion

Nous avons présenté, dans ce chapitre, un cas d'étude - un protocole d'appartenance de groupe dans les réseaux ad hoc. L'objectif de ce GMP est de maintenir une vue cohérente des membres du groupe, en présence de mobilité physique de dispositifs. Son principe se base sur la notion de *Distance de sécurité*. Nous avons conduit des analyses et des expériences de test du protocole. Les résultats ont révélé plusieurs problèmes, montrant que travaux de recherche sur les systèmes mobiles devraient accorder plus d'attention aux questions de V&V. Cependant, notre objectif premier n'était pas tant de trouver des bogues dans un prototype de recherche. Il était plutôt de s'attaquer à un exemple non-trivial de système mobile et d'obtenir une vue concrète des problèmes de test associés.

D'un point de vue technologique, nous avons constaté la nécessité d'inclure, dans les plateformes de test, des fonctionnalités de simulation avancées pour gérer les aspects réseau et les données contextuelles.

D'un point de vue plus conceptuel, un problème majeur pour le test est l'absence de formalismes adéquats pour modéliser un système mobile. Par exemple, la rétro-conception du

GMP a montré que si UML était suffisant pour modéliser un nœud, la représentation de la structure et du comportement d'un système composé de plusieurs nœuds mobiles reste difficile. Certains travaux sur le test de protocole [Noudem et Viho 05], [Cavalli *et al*. 04], mentionnés le chapitre précédent, ont montré qu'il était possible de s'en sortir avec des formalismes classiques (SDL dans leur cas), dans une certaine mesure. Cependant, cela nécessite l'introduction d'artifices de modélisation pour prendre en compte le contexte mobile. Il serait plus avantageux de partir de formalismes de spécification et conception déjà équipés de concepts relatifs à la mobilité.

Les formalismes pertinents pour le test ne se limitent pas à ceux permettant une modélisation complète du système. La modélisation de fragments de comportement, via des scénarios, est notamment très utilisée dans le cadre des systèmes distribués traditionnels. Dans l'exemple du GMP, la représentation graphique de scénarios s'est avérée très utile pour offrir une vue système lors des analyses de la spécification, de l'implémentation, et des traces de test du protocole. Nous avons cependant mis en évidence certaines lacunes des langages de scénarios existants, qui n'offrent pas de concepts pour exprimer des relations spatiales entre les nœuds, ou pour représenter la communication par diffusion dans le voisinage. Ces deux points nous sont apparus très importants pour représenter des interactions dans un contexte mobile.

Dans la suite de ce mémoire, nous orientons nos travaux vers la définition d'un cadre de test basé sur des scénarios, qui inclut (1) la proposition d'extensions aux langages de scénarios existants, pour mieux prendre en compte le contexte mobile, et (2) des traitements automatisés pour analyser et implémenter des scénarios sur une plate-forme de test intégrant des outils de simulation.

CHAPITRE 3

APPROCHE DE TEST BASEE SUR DES DESCRIPTIONS DE SCENARIOS

1. Introduction

Ce chapitre présente les principes généraux d'une approche de test basée sur des descriptions de scénario. Dans une telle approche, les descriptions de scénario sont utilisées pour spécifier des exigences fonctionnelles, des objectifs de test, des cas de test et des traces d'exécution. Ces différents types d'artefact, et les relations qui existent entre eux au sein de l'approche, seront montrés au paragraphe 2. En pratique, l'approche décrite est déjà appliquée dans le cadre des systèmes distribués traditionnels. L'enjeu pour nous est de permettre son transfert aux systèmes mobiles. Comme nous l'avons vu au chapitre précédent, ce transfert nécessite d'étendre les descriptions de scénario pour couvrir les spécificités des systèmes mobiles. Dans le paragraphe 3, nous introduisons les extensions proposées, qui portent essentiellement sur les relations spatiales entre les nœuds et la communication par diffusion dans le voisinage. L'utilité de ces extensions sera illustrée par plusieurs exemples de scénarios dans un contexte mobile (paragraphe 4). Ensuite, dans le paragraphe 5, nous discuterons des traitements automatiques qui peuvent être effectués à partir des descriptions de scénarios. Nous mettrons l'accent sur les problèmes d'appariement de graphes qui apparaissent dans ces traitements, du fait de la prise en compte explicite des relations spatiales entre nœuds.

Les idées et propositions présentées dans ce chapitre ont d'abord fait l'objet d'un livrable du projet européen HIDENETS [HIDENETS D5.2], puis d'une publication internationale [Nguyen et al. 08].

2. Vue générale de l'approche de test basée sur les scénarios

Comme mentionné dans le chapitre 1, les descriptions de scénario sont largement utilisées pour aider des activités liées au test. Ces descriptions peuvent intervenir dans la capture d'exigences, la spécification d'objectifs de test, la conception de cas de test, ou encore l'analyse de traces d'exécution. L'approche générale décrite dans la Figure 17 montre les artefacts basés sur les scénarios qui peuvent être produits durant les différentes phases de V&V.

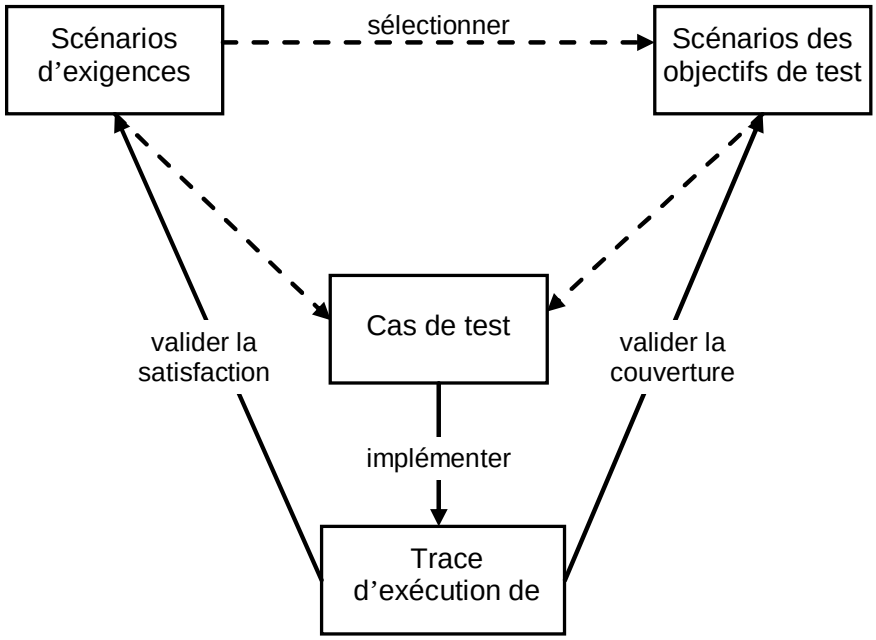


Figure 17. La vue globale de l'approche

L'approche montrée n'exige pas de disposer d'un cadre complètement formel. La transition d'un artefact à un autre peut donc être informelle, ce qui est représenté par des lignes pointillées dans la figure. Par exemple, les objectifs de test peuvent être dérivés manuellement des exigences fonctionnelles. Ou encore, des cas de test peuvent être proposés par l'utilisateur pour couvrir un objectif donné. Notons que nous n'écartons pas la possibilité d'utiliser des approches plus formelles. Si une spécification complète de comportement était disponible, on pourrait envisager des traitements formels, tels que : la vérification qu'un modèle de comportement satisfait les exigences, ou la génération automatisée des cas de test à partir d'un modèle et d'un ensemble d'objectifs de test. Nous n'avons cependant pas étudié ce type de traitement, la modélisation complète du comportement d'un système mobile nous paraissant problématique.

Même si une spécification complète de comportement n'est pas disponible, certains traitements automatisés deviennent possibles du fait de la modélisation partielle apportée par les scénarios. Ils sont indiqués par des lignes pleines dans la figure. Ces traitements consistent à:

- Vérifier qu'une trace d'exécution satisfait un scénario d'exigence.
- Vérifier qu'une trace d'exécution couvre un objectif de test.
- Aider dans l'implémentation de cas de test (ou plus précisément dans la production des données contextuelles concrètes, comme nous le verrons par la suite).

Cela suppose cependant que le langage de scénario utilisé possède une sémantique bien définie. Notons que, selon le type d'artefact considéré, différentes variantes d'un langage de base peuvent être utilisées (par exemple, différents profils de diagrammes de séquence UML). Les scénarios de la Figure 17 n'utilisent pas nécessairement les mêmes ensembles d'éléments du langage, et on pourra choisir les éléments qui conviennent à chaque type de scénario.

Scénario d'exigence. Un scénario d'exigence exprime une propriété attendue du système sous test. Cette propriété doit être satisfaite par toutes les exécutions du système. Bien évidemment, l'ensemble des scénarios d'exigence ne produit pas une description complète du comportement attendu, il se focalise sur quelques propriétés de haut niveau. Dans ce type de scénario, on veut pouvoir exprimer:

- des modalités de type « possibilité/obligation », par exemple pour représenter le fait qu'un fragment de scénario *peut* se produire, et lorsque c'est le cas alors un autre fragment *devra* suivre.
- des traces partielles, ne représentant qu'un sous-ensemble des entités du système et un sous-ensemble des messages échangés.

Objectif de test. Un objectif de test indique un fragment de comportement que l'on souhaite couvrir durant le test. L'objectif de test se focalise sur :

- des traces partielles,
- des classes de comportements, dans le sens où plusieurs cas de test peuvent être créés en se basant sur un objectif donné.

Cas de test. Un cas de test peut être créé à partir d'un objectif ou directement à partir d'une exigence fonctionnelle. Il doit contenir :

- une description complète des événements de contrôle (issus des composants de test) et des événements observables (issus du système sous test) relatifs au cas de test.
- des messages de synchronisation entre les composants de test,
- une description précise du verdict de test (fail/pass/inconclusive) basé sur les événements observables, pour statuer sur le résultat du test.

Trace d'exécution de test. Il s'agit ici de représenter, sous forme de scénario graphique, les événements collectés durant une exécution. Le langage utilisé peut être simple (pas de modalités, de boucles, de choix, de parallélisme, etc.).

Si l'on prend l'exemple des diagrammes de séquences UML, on trouvera des spécialisations très différentes, telles que le profil U2TP (*UML 2.0 Testing Profile*) [U2TP], qui cible la représentation de cas de test, ou encore les MSD (*Modal Sequence Diagrams*) [Harel et Maoz 04] qui ciblent la représentation d'exigences.

Quelle que soit la spécialisation retenue, les langages de base pour représenter des scénarios n'ont pas été conçus pour décrire des interactions dans un contexte mobile. Nous proposons maintenant des extensions à ces langages, pour prendre en compte certaines spécificités des systèmes mobiles.

3. Extensions proposées pour les langages de scénario dans le cadre de systèmes mobiles

Comme nous l'avons expliqué dans le chapitre 1, les langages de scénarios s'intéressent typiquement au comportement d'un système en termes d'événements de communication. Certains événements sont ordonnés, d'autres peuvent s'entrelacer de façon non-déterministe, produisant ainsi des ordres partiels. A titre d'illustration, la Figure 18 fournit la représentation

– sous une forme proche d'une MSC – d'un scénario de défaillance trouvé en testant le GMP. Ce scénario a été présenté dans le chapitre précédent. Nous le reproduisions ici pour faciliter la discussion (notons que nous avons renommé les nœuds en *n1, … n4*, pour nous abstraire des identifiants concrets de la trace d'exécution). La figure montre un entrelacement spécifique des messages relatifs à deux opérations concurrentes de fusion et de scission de groupe. À la fin du scénario, une violation de la propriété JCG (voir le Tableau 1) est observée sur le nœud *n2*.

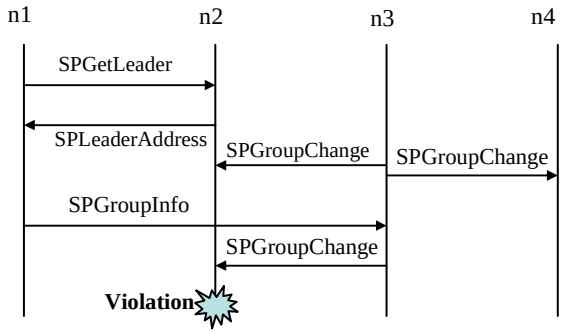


Figure 18. Un scénario de défaillance dans le GMP

Dans l'étude de cas du GMP, de telles représentations graphiques se sont avérées très utiles pour comprendre le comportement défaillant pendant le test. Lors de l'analyse post-mortem, nous avons reformulé une partie des traces de test sous forme de diagrammes MSC, de façon à acquérir une vue plus claire de ce qui se passait. Grâce à l'analyse de ces scénarios, nous avons pu distinguer et regrouper les grandes classes de violations. Les représentations graphiques de scénarios s'étaient également avérées utiles lors des phases précédentes de l'étude du GMP, portant sur l'analyse de la spécification et de la conception. Cependant, nous avons constaté que les langages de scénario classiques ne pouvaient pas capturer certaines spécificités des systèmes mobiles.

Revenons à l'exemple de la Figure 18. Premièrement, il met l'accent sur l'ordre partiel de messages, alors que la topologie spatiale des nœuds est tout aussi importante pour caractériser le scénario. En effet, le comportement de fusion et de scission est gouverné par les changements topologiques perçus par les noeuds. Deuxièmement, la notation MSC ne permet pas d'exprimer la notion de communication par diffusion – sans parler de la diffusion de messages *dans le voisinage*. Ici, les messages "hello" (pour la découverte de groupe) n'ont pas été représentés, faute de disposer de concepts convenables dans le langage. Notons que l'expéditeur d'un message "hello" ne connaît pas *a priori* le nombre et l'identité des récepteurs potentiels. Tout nœud se situant à portée de transmission est susceptible de traiter le message.

Notons qu'une extension de MSC [Kruger et al. 04] a été proposée pour représenter la communication par diffusion. Nous la montrons dans la Figure 19. Le nœud *n2* diffuse un message *m* aux nœuds *n1*, *n3*, *n4* (l'événement d'émission est représenté en blanc, tandis que les événements de réception sont en noir).

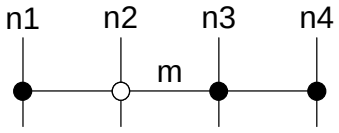
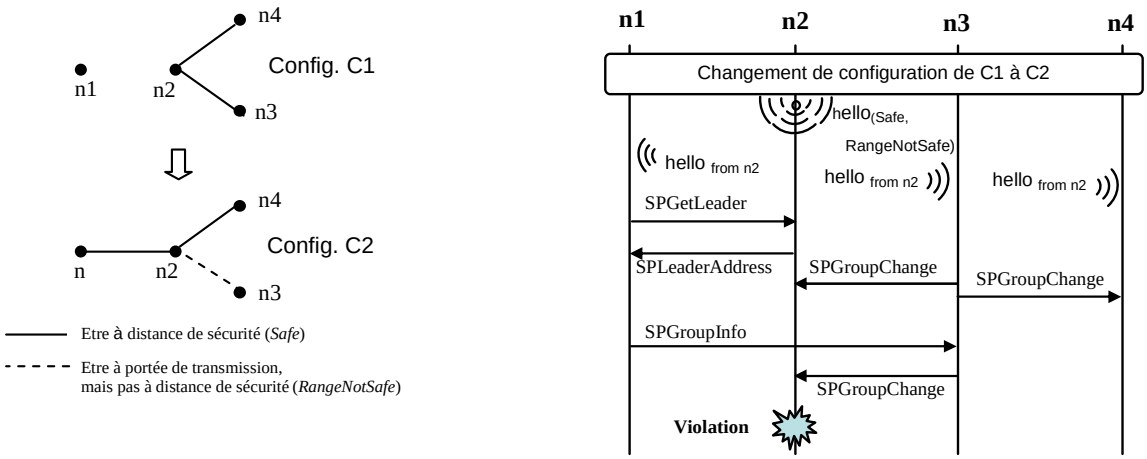


Figure 19. Communication par diffusion en MSC

Cette représentation suggère visuellement une simultanéité des événements d'émission et de réception. Il serait malaisé de représenter, par exemple, le fait que la réception sur le nœud *n4* précède celle sur le nœud n3, avec éventuellement un événement intervenant entre les deux. Ainsi, cette représentation nous paraît plus naturelle pour des systèmes où la communication est synchrone. Dans le cas des systèmes mobiles (et des systèmes répartis en général), il nous semble préférable de distinguer explicitement l'événement d'émission et chacun des événements de réception correspondants.

Considérons maintenant la représentation étendue montrée dans la Figure 20. Elle contient à la fois (a) une vue sous forme de graphes des configurations spatiales successives, et (b) une vue de type MSC des événements de communication, avec des références explicites aux configurations spatiales. Il est maintenant clair que le scénario est déclenché par le message "hello" diffusé par le nœud *n2* après un changement de configuration spatiale. Plus spécifiquement, *n2* n'est plus à une distance de sécurité du nœud *n3*, tout en se rapprochant du nœud *n1*. Les deux messages *SPGetLeader* du nœud *n1* (qui initie une fusion de groupe) et *SPGroupChange* du nœud *n3* (relatif à une scission) sont causalement rattachés au message "hello" : en recevant les données de localisation de *n2*, les autres nœuds s'aperçoivent d'un changement dans la topologie. Sans entrer dans les détails du scénario, nous insistons sur le fait que *tant* l'ordre des messages *que* la topologie de nœuds sont importants pour que la défaillance se produise. En particulier, il est important que :



(a) Configurations spatiales (b) Evénement: vue de type MSC

Figure 20. Scénario combinant vue spatiale et événementielle

- Le changement de configuration spatiale brise le chemin transitif de sécurité qui existait entre *n3* et *n4* (c'est précisément le traitement de ce nœud *n4* qui provoquera la violation de JCG au niveau du nœud *n2*, suite à deux changements de groupe incohérents).
- Interrogé sur l'adresse de son leader, le nœud *n2* répond avant d'être informé de la scission en cours.

En faisant abstraction de la syntaxe concrète montrée dans la Figure 20, les extensions du langage que nous proposons pour mieux prendre en compte le contexte mobile sont les suivantes :

- Les relations spatiales entre nœuds doivent être considérées comme des concepts de première classe. Il incombe à l'utilisateur de déterminer quelles relations sont les plus adéquates pour représenter de façon abstraite la configuration spatiale de l'application ciblée. Par exemple, deux relations spatiales gouvernent le comportement de GMP : être à une distance de sécurité (ce qui détermine la décision de scinder ou fusionner des groupes) et être à portée de transmission (ce qui détermine la connectivité à un saut). Les graphes étiquetés sont adéquats pour modéliser de telles relations et nous pouvons retenir le principe d'utiliser un formalisme visuel pour représenter les configurations spatiales successives.
- Les scénarios d'interactions sont décomposés en fragments, où chaque fragment se déroule dans une des configurations spatiales précédemment définies. Les changements de configuration sont représentés par une synchronisation globale (ou de façon équivalente, nous avons une composition séquentielle forte des fragments). De cette manière, les dépendances causales entre les changements de configuration et les interactions entre nœuds sont rendues explicites. Il est également indiqué explicitement quel événement se produit dans quelle configuration.
- La communication par diffusion est introduite (dans la Figure 20b, elle est concrètement représentée par un signe de transmission radio). Il y a un unique événement d'envoi, suivi par un ensemble d'événements concurrents de réception. La notation doit également nous permettre de spécifier le sous-ensemble de nœuds récepteurs, en faisant référence aux relations spatiales dans la configuration sous-jacente.

Au delà de l'exemple du GMP, nous pensons que ces trois extensions répondent à des besoins récurrents lorsque l'on cherche modéliser des scénarios dans un contexte mobile. Ces extensions peuvent être introduites quelle que soit la variante de la notation choisie (par exemple, langages dérivés des MSC ou des diagrammes de séquences UML) et nous paraissent utiles quelle que soit la finalité des scénarios (exprimer les exigences d'une application mobile, spécifier des objectifs de test ou des cas de test, faire une rétro-conception de traces d'exécution).

4. Exemples de scénarios utilisant nos extensions

Pour illustrer l'intérêt des extensions proposées, nous allons donner des exemples simples de scénarios inspirés de deux études de cas : le GMP et une boîte noire pour les véhicules automobiles. Cette dernière étude de cas provient de travaux menés au LAAS sur la sauvegarde coopérative de données dans les systèmes mobiles [Killijian et al. 09]. Une boîte noire dans un véhicule peut enregistrer des données comme la vitesse, le freinage, la position de conducteur, etc... L'idée de nos collègues est de remplacer une boîte noire physique par une boîte noire virtuelle, dans laquelle les données locales sont redondées en utilisant l'espace de stockage des véhicules voisins. Lorsqu'un véhicule rencontre un point d'accès à l'infrastructure, il peut effectuer l'archivage des données qui lui ont été confiées, et libérer la mémoire correspondante. On a ainsi à la fois des communications de véhicule à véhicule (via des réseaux ad hoc) et des communications de véhicule à infrastructure.

Pour ces deux applications, GMP et boîte noire, nous allons montrer que les extensions proposées sont pertinentes pour différents types de scénarios : scénarios d'exigence, objectifs de test et cas de test.

4.1. Exemples de scénarios d'exigence

Dans ce paragraphe, nous allons utiliser le langage MSD [Harel et Maoz 08] pour exprimer des exigences. MSD permet de définir des fragments d'interaction de type « possibilité/obligation ». Par exemple, des messages *doivent* être envoyés, des conditions *peuvent* être vraies, etc. Un exemple basique de MSD est montré dans la Figure 21.

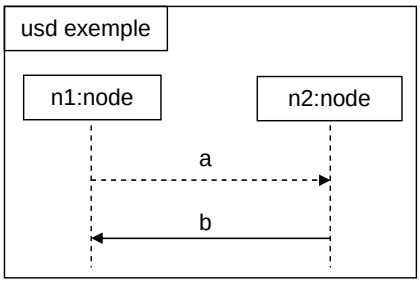


Figure 21. Un exemple de MSD

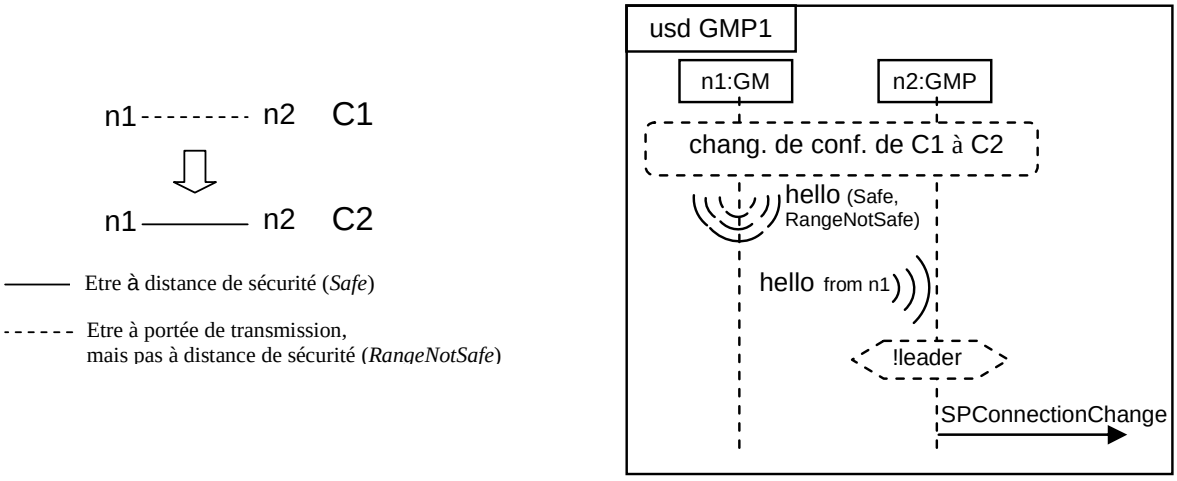
Ce MSD est de type « obligation », comme indiqué par le cadre en trait plein et le mot clé « usd » (*universal sequence diagram*). De façon générale, une possibilité est indiquée par un trait pointillé, alors qu'une obligation est associée à un trait plein. Le diagramme de la Figure 21 spécifie que, si le message *a* apparaît, alors le message *b* doit être inclus dans la suite de l'exécution pour satisfaire l'exigence.

4.1.1. Illustration sur le GMP

Nous voulons exprimer l'exigence suivante:

« ***Si un nœud détecte un nouveau voisin à distance de sécurité, il doit notifier le changement de configuration à son leader*** ».

Pour représenter la notion de « nouveau » voisin, il est commode de considérer deux configurations spatiales successives (Figure 22a) : une première configuration dans laquelle les nœuds *n1* et *n2* ne sont pas encore à distance de sécurité, puis une configuration dans laquelle ils se sont rapprochés. La « détection » du voisin s'exprime par la réception d'un message *hello* postérieur au changement de configuration. Ce message *hello* ne correspond pas à une communication point-à-point : il est diffusé par émission radio, et tous les nœuds à portée de transmission (c'est à dire connectés à *n1* par une relation *Safe* ou *RangeNotSafe*) peuvent le traiter. Notons que, dans la représentation du scénario, *n1* et *n2* sont des identifiants symboliques de nœuds, pouvant prendre n'importe quelle valeur concrète. Toute paire de nœuds du système exhibant la séquence de configurations spatiales montrée dans la Figure 22.a doit satisfaire l'obligation spécifiée dans l'interaction de la Figure 22.b.



(a) Vue spatiale (b) Vue événementielle

Figure 22. Exemple de scénario d'exigence du GMP

4.1.2. Illustration sur la boîte noire

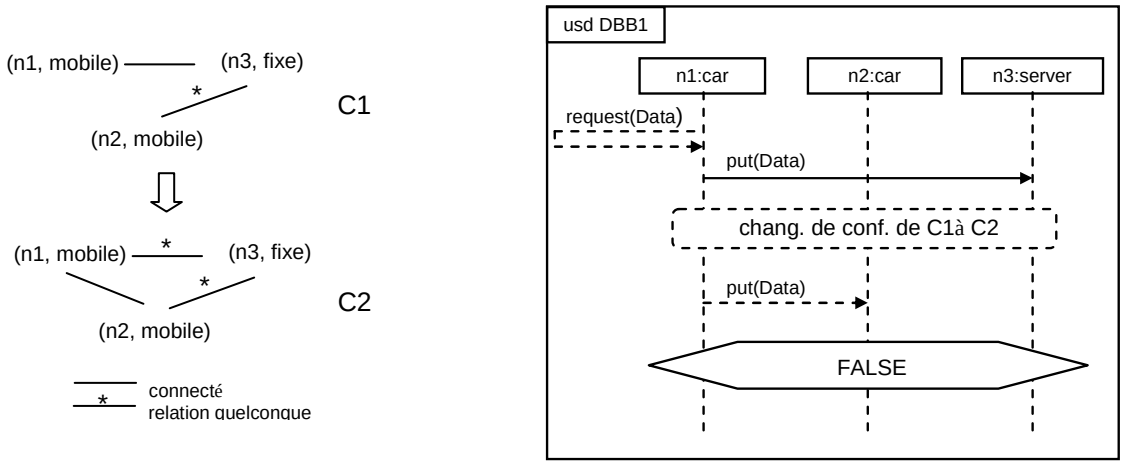
Une exigence de la politique de sauvegarde coopérative est la suivante :

« Après avoir sauvegardé une donnée sur un serveur d'infrastructure, un véhicule ne peut pas sauvegarder la même donnée sur un autre véhicule »

On ne doit donc pas "gaspiller" l'espace mémoire des véhicules voisins en leur envoyant des données déjà archivées sur l'infrastructure (considérée comme offrant un support sûr).

Nous allons nous servir de cette exigence pour illustrer la possibilité d'avoir une vue spatiale plus riche que dans le scénario du GMP précédent. Dans la Figure 23.a, nous utilisons des graphes avec des n-uplets d'étiquettes, ces étiquettes pouvant être des variables (ex : l'identifiant symbolique *n1*), des constantes (ex : *mobile*) ou des métacaractères ayant une interprétation particulière (ex : le joker '*', indiquant que la valeur de l'étiquette correspondante est non spécifiée). Ainsi, les configurations spécifiées mettent en jeu deux nœuds ayant l'attribut *mobile* (les véhicules *n1* et *n2*) et un nœud ayant l'attribut *fixe* (le serveur *n3*). Dans la configuration *C1*, les véhicules *n1* et *n2* ne sont pas à portée de communication l'un de l'autre, *n1* est connecté à *n3*, et la connexion de *n2* et *n3* est non spécifiée. Dans la configuration *C2*, *n1* et *n2* peuvent communiquer, et leur relation à l'infrastructure est quelconque.

L'exigence ci-dessus est alors décrite par deux fragments d'interaction, l'un se déroulant dans la configuration *C1* et l'autre dans *C2* (Figure 23.b). Dans un premier temps, *n1* sauve une donnée sur l'infrastructure (noter l'obligation d'effectuer la sauvegarde lorsque l'application boîte noire le demande et qu'il y a une connexion à l'infrastructure). Dans un deuxième temps, *n1* rencontre *n2* (configuration *C2*) et ne doit pas lui envoyer la même donnée. En MSD, l'interdiction s'exprime par une obligation *FALSE* placée en fin de fragment.



(a) vue spatiale (b) vue événementielle

Figure 23. Exemple de scénario d'exigence de la boîte noire

Au travers de cet exemple, nous avons voulu montrer qu'on peut avoir besoin d'une syntaxe de graphe relativement riche, pour représenter commodément les configurations spatiales intéressantes. De fait, la syntaxe que nous avons retenue pour nos travaux (qui sera présentée au chapitre suivant) offre les possibilités montrées dans la Figure 23.a : n-uplet d'étiquettes, étiquettes variables ou constantes, joker.

4.2. Exemples d'objectifs de test

Un objectif de test spécifie un fragment de comportement à couvrir pendant le test. De même qu'un scénario d'exigence, un objectif de test décrit une interaction partielle. Il focalise sur un sous-ensemble de nœuds du système et un sous-ensemble de messages entre ces nœuds.

Dans ce paragraphe, nous allons représenter des objectifs de test avec des diagrammes de séquence UML 2.0.

4.2.1. Illustration sur le GMP

Nous voulons spécifier un objectif de test relatif à des fusions concurrentes de groupe :

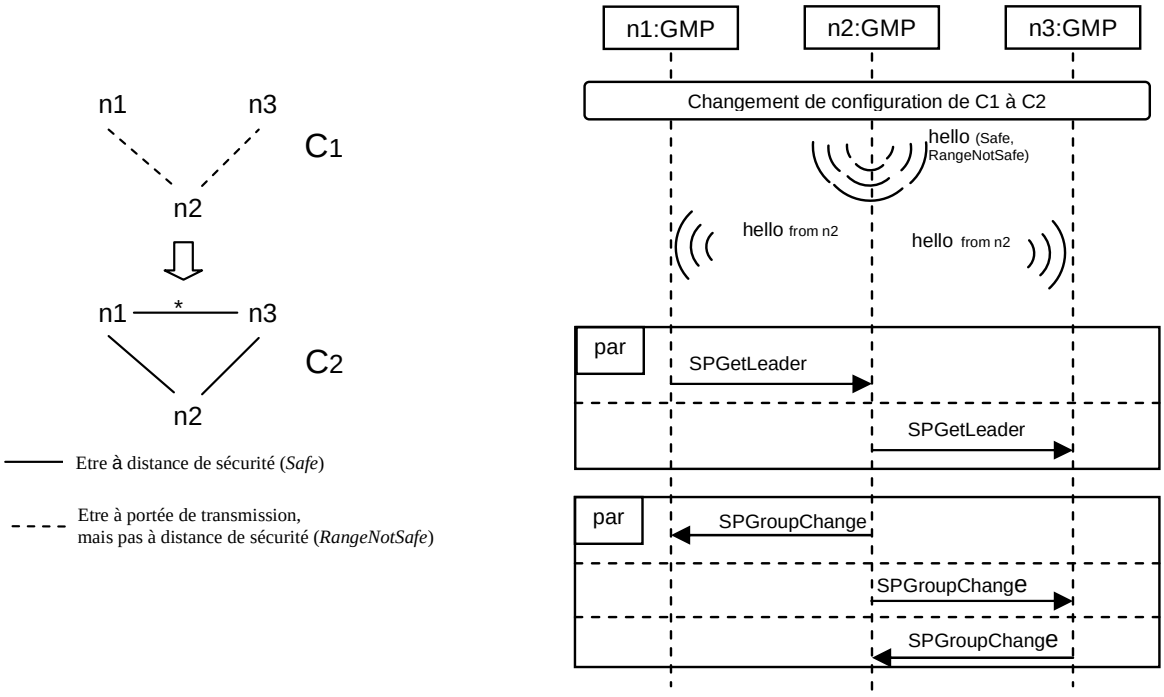
'Tester un cas de deux fusions concurrentes, dans lequel un même nœud joue à la fois le rôle de nouveau leader vis-à-vis d'une fusion, et le rôle de nouveau membre vis-à-vis de l'autre fusion'.

La topologie choisie pour ce scénario est la suivante. Au début, nous avons trois nœuds *n1*, *n2* et *n3* qui ne sont pas à distance de sécurité les uns des autres. Les groupes singletons correspondants sont disjoints. Ces nœuds se déplacent et provoquent un changement de configuration : les nœuds *n1* et *n3* entrent dans la distance de sécurité de *n2*. Les configurations sont montrées dans la Figure 24a.

Dans la vue événementielle (Figure 24b), *n2* diffuse ses coordonnées après le changement de configuration (message *Hello*). Chacun des deux autres nœuds initie une

fusion avec ce nouveau voisin (messages *GetLeader*). Ces fusions mettent en jeu des messages intermédiaires non détaillés dans l'objectif de test. Le message *SPGroupChange* de *n2* vers *n1* indique qu'il joue le rôle de nouveau leader vis-à-vis de la fusion avec *n1*, alors que le nouveau leader est *n3* pour l'autre fusion.

D'après nos résultats de test du GMP, certains cas de test couvrant cet objectif vont provoquer une violation de propriété, alors que d'autres vont se dérouler correctement. On pourrait raffiner l'objectif de test, en montrant plus de messages intermédiaires et en spécifiant différents cas d'entrelacements à couvrir.



(a) Vue spatiale (b) Vue événementielle

Figure 24. Exemple d'objectif de test du GMP

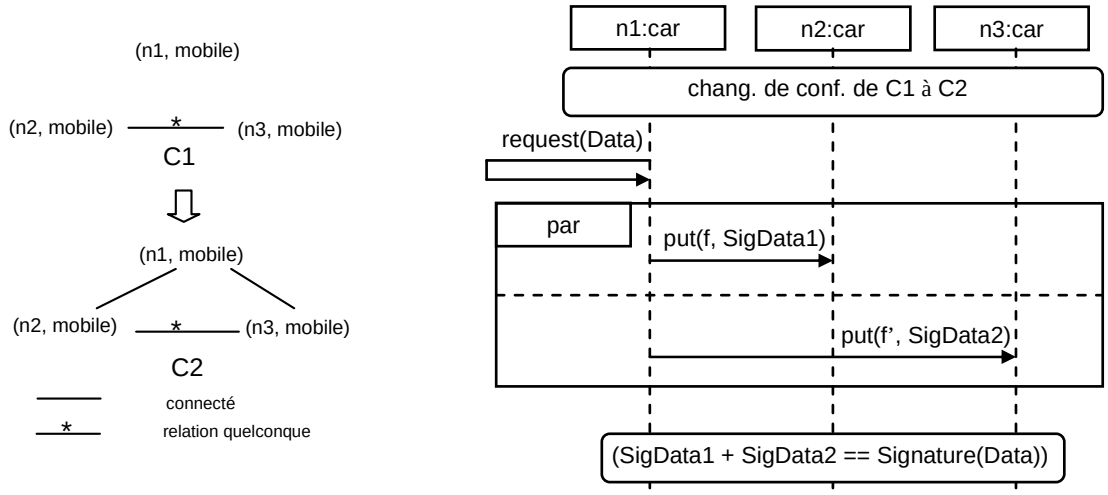
4.2.2. Illustration sur la boîte noire

Dans l'application développée par nos collègues, les données à sauvegarder sont fragmentées et cryptées. Nous voulons exprimer l'objectif suivant :

'Supposons qu'un véhicule n1 ait besoin de sauvegarder une donnée Data qui consiste en deux fragments. Tester au moins un cas où n1 rencontre deux véhicules n2 et n3, et met sa donnée Data sur ces deux véhicules'.

Une donnée ou un fragment de donnée a une signature. Quand un véhicule effectue la sauvegarde, il envoie une paire (fragment, signature) aux voisins. Dans la description du scénario, nous utilisons une étiquette de type entier pour représenter la signature, et nous supposons que le contrôle de signature peut être effectué par une opération arithmétique élémentaire.

L'évolution topologique de ce scénario est décrite dans la Figure 25.a. Dans la configuration *C1*, le véhicule *n1* n'est pas connecté aux véhicules *n2*, *n3*. Ensuite, un changement de configuration se produit et il devient connecté. Les véhicules *n2*, *n3* peuvent avoir n'importe quelle relation spatiale, représentée par le joker '*'. Dans la vue événementielle (Figure 25.b), la sauvegarde des deux fragments s'effectue dans un ordre quelconque (opérateur *par*). La vérification *SigData1+SigData2==Signature(Data)* indique que les deux messages *put* sont bien relatifs au cas que l'on cherche à couvrir, c'est-à-dire qu'ils correspondent à la sauvegarde de la donnée *Data*.



(a) Vue spatiale (b) Vue événementielle

Figure 25. Exemple d'objectif de test de la boîte noire

4.3. Exemple de cas de test

Le profil U2TP (*UML 2.0 Testing Profile*) spécialise les diagrammes de séquence UML pour la représentation de cas de test. Ce profil introduit des concepts tels que : l'architecture de test, le verdict de test, les données de test et le temps. Ensemble, ces concepts résultent en un langage de modélisation adéquat pour visualiser, spécifier, analyser, construire et documenter les cas de test d'un système de test. Dans ce paragraphe, nous allons utiliser ce profil pour décrire un cas de test dans un contexte mobile.

Nous prenons un exemple relatif GMP. Il s'agit d'une fusion entre deux singletons. Le cas de test que nous voudrions spécifier est comme suit :

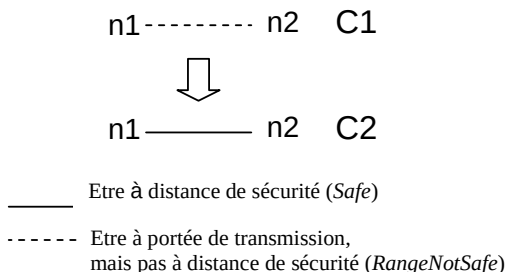
'Vérifier qu'une fusion entre deux groupes singletons est accompli en 4*t_d'

(Rappelons que t_d est défini comme la latence maximale du réseau)

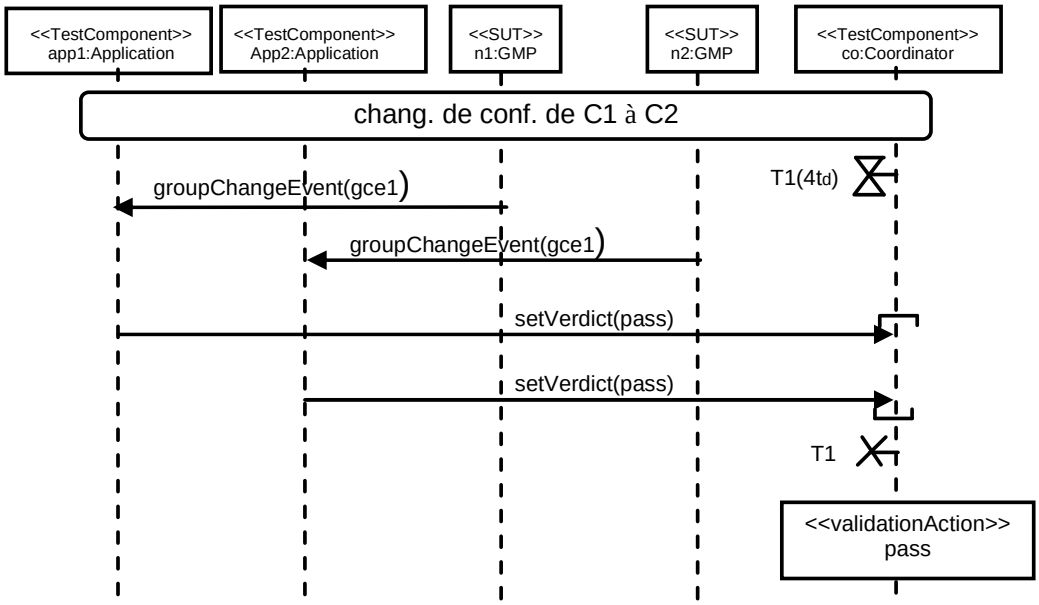
Ce cas de test peut être décrit comme dans la Figure 26.b, avec l'évolution de la topologie des nœuds de la Figure 26.a, où deux singletons deviennent connectés.

Dans ce scénario, chaque nœud mobile est composé de deux lignes de vie : l'une pour le protocole GMP (qui a le stéréotype *<<SUT>>* pour *System Under Test*), et l'autre pour l'application qui utilise le GMP (en fait un bouchon d'application, qui a le stéréotype *<<TestComponent>>*). Les messages du protocole sont considérés comme internes au

système sous test, et ne sont donc pas représentés dans le scénario. Nous focalisons sur les actions observables dans l'architecture considérée, c'est-à-dire dans cet exemple les messages *groupChangeEvent* envoyés du GMP à l'application. Le composant *Coordinator* récolte les verdicts partiels et synthétise un verdict final. Ce composant vérifie également le temps nécessaire pour traiter une fusion.



(a) Vue spatiale



(b) Vue événementielle

Figure 26. Exemple de cas de test pour le GMP

4.4. Discussion

Nous avons présenté des exemples de scénarios d'exigence, d'objectifs de test et de cas de test. Ces exemples visaient à montrer que, pour les différents types de scénarios, les extensions proposées étaient utiles afin d'exprimer les spécificités de l'environnement mobile.

Dans l'ensemble de ce paragraphe 4, les exemples imaginés avaient un but purement illustratif, et leur présentation est restée à un niveau informel. En pratique, l'introduction de nos extensions à un langage de scénario nécessite d'avoir une définition précise tant de la syntaxe que de la sémantique. L'exercice a été réalisé par des collègues du LAAS et de

l'Université Technologique de Budapest, pour des scénarios d'exigence basés sur les diagrammes de séquence UML 2.0. Le langage TERMOS (*Test Requirement language for Mobile Setting*) [HIDENETS D5.3] a ainsi été développé. Nos collègues ont proposé une solution pour représenter nos extensions tout en restant compatible avec la syntaxe UML spécifiée par l'OMG. Ils ont également défini la sémantique des vues événementielles de TERMOS, en se basant sur les sémantiques existantes pour d'autres langages d'exigences tels que MSD [Harel et Maoz 08] ou les LSC [Damm et Harel 01]. Nous ne détaillons pas davantage ces travaux auxquels nous n'avons pas participé : notre contribution s'est située en amont, dans la proposition des extensions. Notons simplement que, dans la définition de la sémantique des vues événementielles, les extensions proposées n'ont pas posé de problèmes particuliers. En fait, les problèmes venaient plutôt des éléments propres aux diagrammes de séquence UML, pour lesquels différentes interprétations formelles ont été proposées dans la littérature.

Dans la suite, nos travaux ne se focaliseront pas sur un langage particulier. Nous supposerons que nous avons un langage de scénario incorporant nos extensions, et pour lequel la sémantique de la vue événementielle est formellement définie. Nous allons maintenant nous intéresser aux traitements spécifiques à la vue spatiale, et à la façon dont ces traitements complètent ceux réalisés pour interpréter la vue événementielle.

5. Traitements automatisés des descriptions de scénario

Dans l'approche de test de la Figure 17, les descriptions de scénarios ne sont pas utilisées seulement pour faire de la documentation. On veut qu'elles puissent être compilées en programmes qui analysent automatiquement des traces d'exécution. Ce paragraphe explique les différents types de traitement à réaliser.

5.1. Traitements associés aux scénarios d'exigence et aux objectifs de test

On cherche à comparer une trace d'exécution à un scénario (scénario d'exigence ou objectif de test). La comparaison vise différents buts, selon le type de scénario. Les scénarios d'exigences sont utilisés pour détecter si des propriétés sont violées pendant le test. Cela fournit une solution automatisée au problème de l'oracle de test. Les scénarios des objectifs de test sont utilisés pour vérifier que certains fragments de comportements sont provoqués au moins une fois pendant le test. Si c'est le cas, l'objectif est couvert.

On suppose que les traces à analyser ont été collectées lors d'une exécution sur une plate-forme de test exhibant l'architecture générale montrée dans la Figure 16. Les données enregistrées par le *simulateur de contexte* (ex : les coordonnées physiques des nœuds) permettent de déterminer les configurations spatiales rencontrées lors de l'exécution. Cela exige une étape préliminaire d'abstraction pour interpréter les données contextuelles brutes en graphes étiquetés qui seront, ensuite, comparés aux configurations du scénario. Les événements de communication sont observés par des moyens d'instrumentation au niveau du *simulateur du réseau* et du *support d'exécution* de l'application.

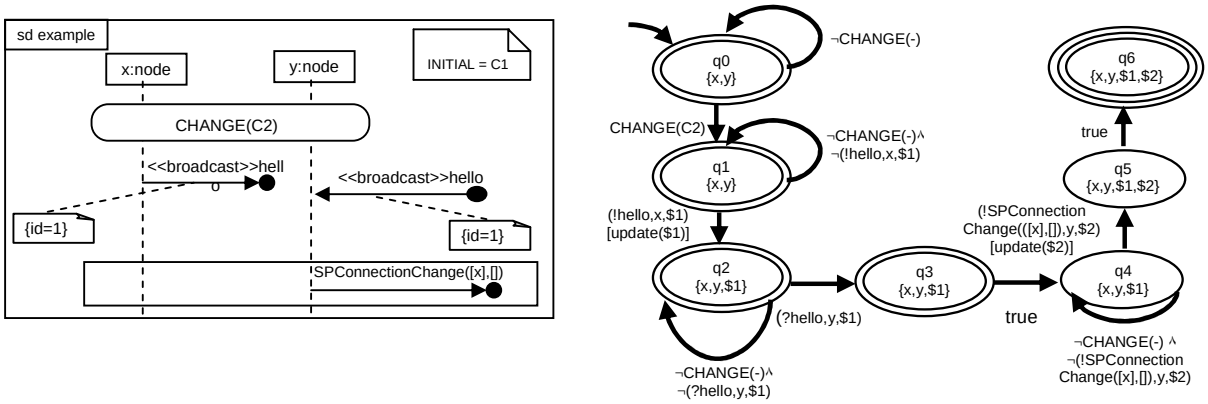
Que l'on cherche à comparer une trace de test à un objectif de test (pour détecter la couverture de l'objectif), ou à un scénario d'exigence (pour détecter la violation de l'exigence), la comparaison implique deux étapes :

1. Déterminer quels nœuds physiques dans la trace de test peuvent jouer le rôle des nœuds apparaissant dans la vue spatiale du scénario;
2. Pour ces nœuds, analyser l'ordre des événements dans les configurations identifiées.

La première étape correspond à un problème d'appariement de graphes. Les configurations spatiales du scénario fournissent un motif que l'on cherche à retrouver dans les configurations concrètes d'exécution (construites à partir des données contextuelles). A l'issue de cette étape, on connaît tous les sous-ensembles de nœuds de la trace qui exhibent la séquence de configurations du scénario, ainsi que les dates des événements de changement de configuration correspondants.

La deuxième étape correspond à une interprétation classique d'un scénario en termes d'ordres partiels d'événements. Cette interprétation s'effectue selon la sémantique définie pour la vue événementielle, en utilisant les informations fournies par l'appariement de graphes.

Pour illustrer ces notions, nous allons prendre l'exemple d'un scénario TERMOS. Si l'on se réfère à la seule vue événementielle, la sémantique définie par nos collègues consiste à produire un automate qui distingue les traces valides et invalides (voir la Figure 27). Nous ne commentons pas en détail l'exemple montré dans la figure. Nous voulons juste attirer l'attention sur le fait que l'automate fait référence à des variables qui devront être instanciées : ce sont notamment les identifiants symboliques des nœuds, *x* et *y*. De plus, les transitions de l'automate font apparaître des événements symboliques de changement de configuration,



(a) Vue événementielle (b) Automate correspondant

Figure 27. Interprétation de la vue événementielle d'un scénario TERMOS

par exemple *CHANGE(C2)*. C'est la première étape de la comparaison, basée sur un algorithme d'appariement de graphes, qui fournira les données de configuration permettant d'instancier l'automate. On pourra alors effectuer l'analyse de l'ordre des événements de la trace (étape 2).

Dans le cadre de nos travaux, nous nous sommes concentré sur la réalisation de l'étape d'appariement de graphes, qui sera décrite au chapitre suivant.

5.2. Aide à l'implémentation des cas de test

Nous avons vu que, du fait de l'introduction d'une vue spatiale, les traitements de scénarios d'exigences et des objectifs de test mettent en jeu des algorithmes d'appariement de graphes. Nous allons maintenant montrer que l'appariement de graphes peut également jouer un rôle dans l'implémentation de scénarios de cas de test, notamment en ce qui concerne la production de données contextuelles.

Supposons qu'un cas de test du GMP ait été spécifié en utilisant les notations étendues que nous proposons. Dans la description du cas de test, le mouvement est abstrait par des modifications de graphe. Mais au final, l'implémentation devra être testée avec des données de localisation physique pour chaque nœud. Comment produire des données contextuelles concrètes, de façon à instancier les évolutions topologiques souhaitées, est un problème aigu.

D'une manière générale, les données de localisation peuvent obéir à un modèle de mobilité complexe (par exemple, véhicules se déplaçant sur un réseau de routes dans une région géographique donnée). De plus, les abstractions utilisées dans le graphe étiqueté peuvent être plus riches que la simple considération des distances entre nœuds. La production manuelle de données contextuelles peut alors être fastidieuse, voire irréaliste.

S'il paraît difficile d'ajuster manuellement les valeurs des données contextuelles, il pourrait être plus facile de provoquer les configurations souhaitées au moins une fois lors d'une simulation aléatoire. Dans les plates-formes de test, les simulateurs de contexte sont utilisés pour gérer le mouvement des nœuds et produire les données contextuelles correspondantes. Notre proposition est alors d'avoir une phase préliminaire de production de données contextuelle, basée sur des exécutions du seul simulateur de contexte :

- Une exécution peut impliquer de nombreux nœuds se déplaçant selon le modèle de mobilité implémenté. À chaque pas de simulation, les données contextuelles associées à chaque nœud sont enregistrées.
- La trace concrète de simulation est alors abstraite par une séquence de graphes représentant l'évolution de la topologie du système.
- Ensuite, on cherche à vérifier si des sous-graphes peuvent correspondre au motif d'évolution désiré (défini dans le scénario), au moyen d'algorithmes d'appariements de graphe.
- La liste des correspondances trouvées fournit des configurations concrètes alternatives pour implémenter le scénario

Par exemple, à partir d'une simulation aléatoire, on pourrait identifier quatre nœuds exhibant les configurations spatiales montrées dans la Figure 20.a. Les données contextuelles enregistrées pour ces nœuds (par exemple, leurs coordonnées physiques) peuvent alors être extraites de la trace complète, et rejouées pour l'exécution du cas de test.

6. Conclusion

Nous avons présenté dans ce chapitre une approche de test qui se base sur des descriptions de scénarios. Les langages de scénarios n'ayant pas été conçus pour décrire des interactions dans un contexte mobile, il est nécessaire d'introduire des extensions pour couvrir

les nouvelles caractéristiques des systèmes mobiles. Nous avons proposé trois extensions, qui portent sur les relations spatiales des nœuds et sur la communication par diffusion radio :

- La configuration spatiale des nœuds devient un concept de première classe, et est représentée par des graphes étiquetés ;
- Les changements de configuration spatiale apparaissent explicitement dans l'ordre partiel des événements du système ;
- La notion de communication par diffusion dans le voisinage est introduite.

Un exemple d'utilisation de ces extensions est le langage d'exigences TERMOS, qui a été développé dans le cadre du projet HIDENETS.

Comme les descriptions de scénario considèrent explicitement les relations spatiales entre nœuds, les traitements associés mettent en jeu des problèmes d'appariement de graphes, au moins en partie. Ceci est une nouveauté par rapport aux traitements classiquement associés aux scénarios. On a maintenant besoin de déterminer si un nœud physique apparaissant dans une trace peut correspondre à un nœud abstrait du scénario dans la vue spatiale. Pour remplir ce besoin, nous avons développé un outil d'appariement de graphes, *GraphSeq* (*Graph matching tool for Sequence of configurations*), qui sera présenté au chapitre suivant.

CHAPITRE 4

GRAPHSEQ : UN OUTIL D'APPARIEMENT DE GRAPHES POUR L'EXTRACTION DES MOTIFS DE MOBILITE

1. Introduction

Dans l'approche que nous avons introduite dans le chapitre 3, les descriptions de scénarios incluent des graphes pour représenter des configurations spatiales successives. Le traitement formel des scénarios, et plus précisément de la vue spatiale des scénarios, fait donc appel à des algorithmes d'appariements de graphes. Dans ce chapitre, nous présentons la conception de *GraphSeq* (*Graph matching tool for Sequences of configurations*), l'outil que nous avons développé pour rechercher des occurrences de configurations spatiales dans une trace d'exécution.

L'outil *GraphSeq* peut notamment être utilisé en relation avec le langage TERMOS. Plus généralement, il peut être utilisé avec tout langage de scénario incluant une vue spatiale, et montrant les événements de changement de configuration dans la vue événementielle (conformément aux extensions que nous avons proposées pour les langages de scénarios).

Le cadre classique de l'appariement de graphes est la comparaison de deux graphes. Une contribution de *GraphSeq* est de construire des appariements portant sur des séquences de graphes, ce qui est un problème beaucoup moins étudié dans la littérature. De plus, l'application d'algorithmes d'appariement à l'analyse des traces de test de systèmes mobiles est, à notre connaissance, une proposition originale.

La comparaison de deux graphes est vue comme une fonctionnalité de base utilisée par *GraphSeq*. Elle nous permet de déterminer si un graphe *G1* (venant d'une description de scénario) est apparié à un sous-graphe *G2* (venant d'une trace d'exécution). En se basant sur cette fonctionnalité, *GraphSeq* raisonne sur des séquences de graphes. Les définitions relatives à la comparaison de deux graphes, ainsi que le choix d'un outil existant pour mettre en œuvre cette fonctionnalité, sont présentés au paragraphe 2. Le paragraphe 3 introduit le principe du raisonnement séquentiel. Nous énonçons les propriétés que doivent satisfaire les appariements de séquences produits par *GraphSeq*. Ensuite, nous présentons les algorithmes détaillés de *GraphSeq* dans les paragraphes 4 et 5. Nous commençons par le traitement de

scénarios impliquant un ensemble fixe de nœuds, puis ajoutons des considérations pour le cas plus complexe où des nœuds sont créés et détruits dynamiquement. Dans chaque cas, le fonctionnement de *GraphSeq* est illustré sur un exemple comportant deux séquences de graphe à comparer.

Les travaux présentés dans ce chapitre ont fait l'objet d'un livrable du projet HIDENETS [HIDENETS D5.3].

2. Comparaison de deux graphes

Le raisonnement séquentiel élaboré par *GraphSeq* s'appuie sur une série d'étapes plus élémentaires de comparaisons de deux graphes, l'un provenant de la vue spatiale d'un scénario, et l'autre étant extrait d'une trace d'exécution. Il s'agit de déterminer si une configuration du scénario apparaît comme sous-graphe d'une configuration système observée lors de l'exécution. Techniquement, ceci implique une recherche d'*homomorphismes de graphe*.

Dans ce paragraphe, nous rappelons d'abord quelques définitions de base sur les notions de graphe et d'homomorphisme de graphe. Ensuite, nous discuterons du choix d'un outil effectuant la recherche d'homomorphismes pour le compte de *GraphSeq*.

2.1. Définition de graphe

Dans sa forme la plus simple, un graphe est défini par l'ensemble de ses nœuds (ou sommets) et l'ensemble de ses arêtes (ou arcs). Ces graphes basiques sont décrits par un couple $G = (V, E)$ où :

- V est l'ensemble des nœuds constituant le graphe,
- $E \subseteq V \times V$ est l'ensemble des arêtes du graphe où chaque arête est définie par les deux nœuds qu'elle relie.

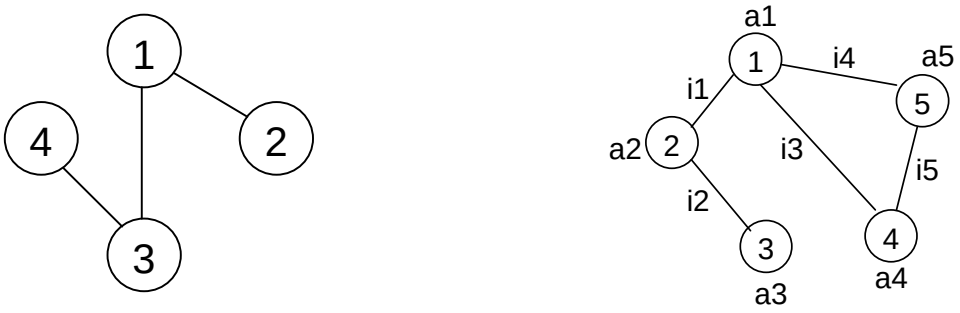
Un exemple de graphe de base est donné dans la Figure 28a.

Afin de permettre des descriptions plus riches, d'autres types de graphes sont définis. Parmi eux, les graphes étiquetés nous intéressent plus particulièrement.

Soient L_V et L_E les ensembles des étiquettes de sommets et d'arêtes. La structure du graphe devient un quadruplet $G = (V, E, \mu, \nu)$ où:

- V est l'ensemble des sommets,
- $E \subseteq V \times V$ est un ensemble des arêtes,
- $\lambda : V \rightarrow L_V$ est la fonction qui assigne les étiquettes aux sommets,
- $\mu : E \rightarrow L_E$ est la fonction qui assigne les étiquettes aux arêtes.

La Figure 28b donne un exemple de graphe étiqueté.



(a) Graphe de base (b) Graphe étiqueté

Figure 28. Exemple de graphes

2.2. Définition d'homomorphisme de graphes

L'apparition d'un graphe *G1* comme sous-graphe de *G2* est traduite mathématiquement par l'existence d'un homomorphisme entre ces deux graphes. Nous présentons d'abord la définition d'homomorphisme pour les graphes de base.

Deux graphes sont homomorphes s'il existe une correspondance injective entre leurs ensembles de nœuds qui préserve la structure de connexion. Ainsi, si deux nœuds d'un graphe sont reliés par une arête, alors les deux nœuds qui leur correspondent par cette injection sont aussi reliés par une arête. On obtient donc la définition formelle suivante.

Définition. Un homomorphisme d'un graphe $G1 = (V1, E1)$ vers un graphe $G2 = (V2, E2)$ est une injection $f : V1 \rightarrow V2$ telle que pour chaque arête $(v_i, v_j) \in E1$, on a $(f(v_i), f(v_j)) \in E2$.

Dans le cas des graphes étiquetés, la contrainte sur la préservation de la structure de connexion reste nécessaire, mais des contraintes supplémentaires sur les étiquettes sont aussi considérées.

Définition. $G_1 = (V_1, E_1, \lambda_1, \mu_1)$ et $G_2 = (V_2, E_2, \lambda_2, \mu_2)$ sont deux graphes étiquetés. Une fonction $f : V_1 \rightarrow V_2$ est un homomorphisme de G_1 à G_2 si et seulement si:

- Elle est injective,
- $\lambda_1(v_i) = \lambda_2(f(v_i))$ pour tout $v_i \in V_1$,
- Pour toute arête $e_1 = (v_i, v_j) \in E_1$, il existe une arête $e_2 = (f(v_i), f(v_j)) \in E2$ telle que $\mu_1(e_1) = \mu_2(e_2)$.

Un exemple d'homomorphisme de graphe est donné dans la Figure 29.

Cette définition formalise la notion de correspondance entre G1 et un sous-graphe de G2. Dans le cadre de nos travaux, *G1* provient d'une description de scénario et sera appelé le graphe « motif » (*pattern graph*). *G2* est extrait d'une trace de simulation ou d'exécution et sera appelé graphe de « configuration concrète ».

Les algorithmes classiques de recherche d'homomorphismes peuvent être trouvés dans [Ullmann 76, Messmer et Bunke 00]. En pratique, nous ne pouvons pas les réutiliser tels quels, car les définitions fondamentales de structure de graphe et d'homomorphisme de graphe doivent être légèrement étendues pour remplir nos besoins.

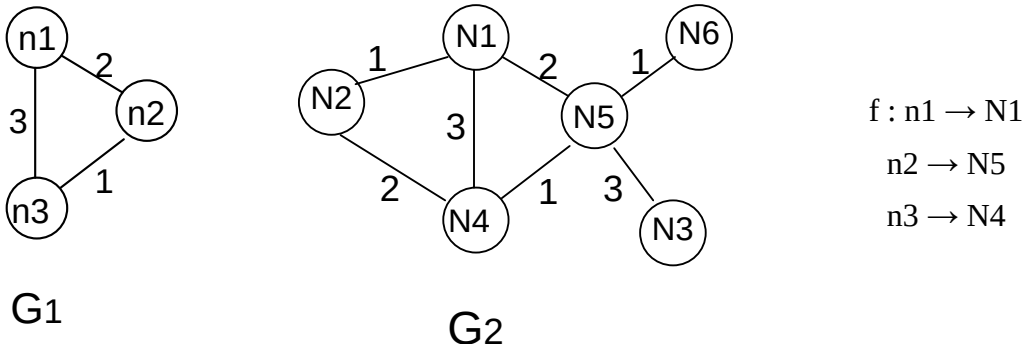


Figure 29. Exemple d'homomorphisme

2.3. Discussion de nos besoins

Tout d'abord, considérons le cas où un motif comporte des nœuds v_i, v_j qui ne sont pas connectés par une arête. D'après les définitions d'homomorphisme ci-dessus, il n'y a pas de contrainte sur la connexion (ou la déconnexion) des nœuds $f(v_i)$, $f(v_j)$. Or, dans les exemples de scénarios montrés dans le chapitre 3, l'absence d'arête entre deux nœuds spécifiait une configuration spatiale dans laquelle ces deux nœuds n'étaient *réellement pas* à portée de communication. Si nous voulons prendre en compte les déconnexions dans les motifs, nous devons donc les considérer comme des arêtes avec une étiquette spécifique. Ceci peut être facilement réalisé lors de l'interfaçage avec l'outil d'appariement de graphes. Ainsi, nous considèrerons par la suite que l'utilisateur exprime la déconnexion par une absence d'arête, mais que l'encodage dans le format d'entrée de l'outil introduit l'arête spécifiant la relation de déconnexion. De même, si l'utilisateur veut laisser la connexion de deux nœuds non-spécifiée, il l'exprimera par une arête étiquetée par un joker '*', et l'encodage dans le format d'entrée de l'outil supprimera cette arête. Les encodages peuvent être réalisés automatiquement, de façon transparente à l'utilisateur. On voit là que le besoin de distinguer (i) la déconnexion, et (ii) l'absence de contrainte sur la connexion, peut se résoudre simplement. Nos autres besoins ont cependant un impact plus fort sur les définitions de graphes et d'homomorphismes de graphes.

Dans le chapitre 3, nos exemples de scénarios ont montré qu'il peut être commode d'allouer un n-uplet d'étiquettes aux sommets et aux arêtes, afin de permettre une représentation plus riche des nœuds et des relations entre nœuds. Par exemple, l'application de la boîte noire implique deux types de nœuds : des nœuds d'infrastructure et des nœuds mobiles. Un nœud peut alors être caractérisé par un 2-uplet <*id, type*>, où *id* est une valeur identifiant de façon unique le nœud physique et *type* est un élément de l'ensemble {"Mobile", "Fixe"} qui différencie les deux types de nœuds. L'impact sur la recherche d'homomorphisme est le suivant : la mise en correspondance de deux sommets et/ou arêtes doit maintenant prendre en compte la longueur des n-uplets, et les valeurs d'étiquettes apparaissant à chaque position du n-uplet.

Enfin, nous avons aussi besoin d'introduire des variables et des métacaractères d'étiquette dans le graphe motif. Les graphes motifs deviennent des graphes avec des n-uplets d'étiquettes qui pouvent être des variables, des constantes ou des métacaractères ayant une interprétation particulière (ex : le joker '*', indiquant que la valeur de l'étiquette

correspondante est non spécifiée). Une étiquette joker peut être ignorée lors de la recherche d'homomorphisme, car elle signifie une absence de contrainte. Par contre, l'introduction de variables nécessite une notion d'*unification* des étiquettes. Par exemple, dans une description de scénario, supposons qu'un nœud soit étiqueté <n1, "Mobile"> : il devrait être possible de détecter une correspondance par un nœud physique <"140.93.130.95", "Mobile"> avec la substitution de variable *n1 :* = "140.93.130.95". Comme on peut le voir dans cet exemple, l'introduction de variables nécessite de revisiter la construction d'homomorphismes de graphe. Il faut exhiber non seulement une fonction *f* qui apparie les noeuds, mais aussi une valuation *Val* qui unifie d'une manière cohérente les étiquettes des graphes motifs et des configurations concrètes. Un homomorphisme de graphe est ainsi un couple *(f, Val)*.

Pour *GraphSeq*, nous avons donc besoin d'une fonctionnalité de base qui permet de comparer deux graphes avec des n-uplets d'étiquettes et des variables d'étiquettes. Nous avons utilisé un outil existant qui sera présenté dans le paragraphe suivant.

2.4. Outil retenu

Pour satisfaire nos besoins, *GraphSeq* utilise un outil développé par des collègues du LAAS-CNRS [Guennoun 06]. Leurs travaux portent sur la spécification d'architectures reconfigurables dynamiquement [Guennoun et Drira 06]. Comme dans notre cas, cela induit des problèmes d'appariement de graphes possédant des étiquettes multiples et symboliques. L'outil correspondant est implémenté en C++ et peut être réutilisé dans le cadre de nos travaux. Il nous fournit des fonctionnalités permettant d'encoder et de comparer des graphes. Pour définir des structures de graphe, l'outil offre les possibilités suivantes :

- Les nœuds peuvent avoir au plus 3 étiquettes, avec un 3-uplet de type STRING × INT × INT,
- Les arêtes ont au plus une étiquette de type INT,
- Les variables d'étiquette sont supportées pour les sommets (dans la version actuelle de l'outil, il n'y a pas de variable d'étiquette pour les arêtes du graphe motif).

L'outil n'a pas de notion de joker sur les étiquettes. Cependant, comme expliqué précédemment, les jokers au niveau des arêtes peuvent être encodés en supprimant l'arête (il n'y aura alors pas de contrainte sur la connexion des nœuds correspondants lors de la recherche d'homomorphisme). Nous avons de plus légèrement étendu l'outil pour permettre de prendre en compte les jokers sur des étiquettes de nœud. Notons que les variables et les jokers sont traités différemment par *GraphSeq*. Lorsque l'outil tente de mettre en correspondance des séquences de graphes, une valeur unique est attribuée à chaque variable, tandis que les jokers dénotent des attributs pouvant varier arbitrairement. Nous donnerons des exemples illustratifs par la suite.

Les homomorphismes retournés par l'outil sont de la forme *(f, Val)*, où *f* est une injection sur les nœuds, et *Val* est une valuation des variables d'étiquettes. Nous rappelons ci-dessous quelques définitions qui sont utilisées dans [Guennoun 06] pour la construction de *Val*, avec une très légère extension introduisant la notion de joker au niveau des étiquettes.

Définition (unification de étiquettes). Deux étiquettes, l_1 appartenant à un nœud du graphe motif et l_2 appartenant à un nœud du graphe de configuration concrète, sont unifiables si et seulement si :

- soit l_1 et l_2 sont deux constantes de même type et de même valeur,
- soit l_1 est une variable de même type que la constante l_2,
- soit l_1 est un joker.

L'unification des étiquettes est utilisée pour définir la notion d'unification des nœuds.

Définition (unification de nœuds). Deux nœuds n_1 et n_2 sont unifiables si et seulement si les trois conditions suivantes sont vérifiées :

- les deux nœuds possèdent le même nombre d'étiquettes,
- ces étiquettes sont, deux à deux, unifiables en tenant compte de l'ordre de leur occurrence,
- le résultat de toutes les unifications d'étiquettes est cohérent.

Dans la troisième condition, la cohérence des unifications d'étiquettes doit être maintenue pour empêcher qu'une variable, présente par exemple dans deux étiquettes, reçoive deux valeurs différentes. Par exemple, un nœud n_1 <x, y, x> n'est pas unifiable avec le nœud n_2 <1, 2, 3> (même si la variable *x* est unifiable à la fois avec la constante 1 et la constante 3). Par contre, ce nœud n_1 est unifiable avec le nœud n_2' <1, 2, 1> et produit comme résultat une valuation {(x, 1), (y, 2)}.

La cohérence des valuations est définie comme suit.

Définition (valuations cohérentes). Deux valuations Val_1 et Val_2 sont cohérentes si et seulement si, pour chaque paire (x_1, $value_1$) appartenant à Val_1 et chaque paire (x_2, $value_2$) appartenant à Val_2, une des deux conditions suivantes est vérifiée :

- x_1 et x_2 sont deux variables différentes
- x_1 et x_2 correspondent à la même variable et les valeurs associées sont les mêmes.

Les valuations sont construites de manière progressive avec les itérations d'appariement des nœuds. La valuation *Val* retournée par l'outil est le résultat de l'union de valuations partielles cohérentes, construites lors de l'unification des différents nœuds.

2.5. Comparaison de deux graphes avec l'outil

Conformément à la syntaxe des graphes de l'outil retenu, les graphes motifs dans *GraphSeq* sont supposés avoir la forme illustrée par la Figure 30. Les sommets ont au plus trois étiquettes. La première est obligatoire, c'est un identifiant symbolique pouvant être associé à un identifiant concret d'un nœud physique. Les deux autres étiquettes peuvent être utilisées pour représenter des attributs supplémentaires de type "entier" (des entiers, ou des types énumérés). Leur forme peut être :

- Une valeur constante du type. Dans la Figure 30, les attributs du nœud *id2* ont pour valeurs constantes *1* et *2*.
- Un nom de variable qui dénote une valeur du type. Par exemple, les premiers attributs optionnels des nœuds *id1* et *id3* doivent être identiques, mais la valeur précise est laissée non spécifiée (variable *v1*). Cette valeur est destinée à rester stable dans la configuration. De plus, si un scénario implique plusieurs graphes

motifs qui contiennent la variable d'étiquette *v1*, elle doit être remplacée par une valeur unique.

- Un joker ('*') qui indique une valeur non significative, voir par exemple le dernier attribut du nœud *id1*. Les valeurs non significatives n'ont pas besoin de rester stable dans la configuration.

Les arêtes peuvent être étiquetées par des valeurs constantes ou des jokers. Dans la Figure 30, le type de la connexion est {*Safe*, *RangeNotSafe*} (ces exemples de relations sont inspirés par le cas d'étude GMP, présenté dans le chapitre 2). Les nœuds *id1* et *id2* ont une connexion *Safe*, les nœuds *id2* et *id3* sont déconnectés. Nous ne prenons pas en compte la connexion entre les nœuds *id1* et *id3*, ils peuvent présenter des connexions/déconnexions instables durant la configuration.

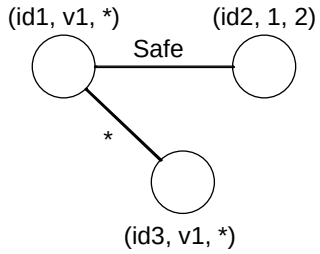


Figure 30. Un graphe avec différentes étiquettes

La syntaxe des graphes de configurations concrètes est plus simple, car ils contiennent uniquement des étiquettes constantes. Les identifiants concrets de nœuds sont des chaînes de caractères, toutes les autres étiquettes sont des constantes entières.

Les détails de la syntaxe des graphes utilisés dans *GraphSeq* (ex : longueur et type des n-uplets d'étiquettes, variables d'étiquettes sur les nœuds et/ou les arêtes) sont liés à l'outil retenu pour la recherche d'homomorphismes. En pratique, nous pourrions étendre cette syntaxe pour avoir une représentation plus riche des graphes, par exemple : n étiquettes pour les sommets avec n>3 ; n-uplets avec des variables pour les arêtes… Cependant, nous n'avons pas introduit ces extensions, qui auraient nécessité de modifier le code de l'outil. Rappelons que notre travail se concentre sur l'appariement de séquences de graphes : l'outil de nos collègues est pris à titre d'exemple, pour réaliser la fonctionnalité de base de comparaison de deux graphes.

En fait, le principe général du raisonnement sur les séquences de graphes est relativement indépendant des détails de l'outil. Les algorithmes que nous allons présenter par la suite pourraient être implémentés en se basant sur d'autres outils de recherche d'homomorphismes, s'accommodant d'une syntaxe de graphe différente. Du point de vue de nos algorithmes, la seule contrainte est qu'une des étiquettes de nœud puisse être interprétée comme un identifiant, apparaissant sous forme symbolique dans les motifs, et sous forme valuée dans les graphes de configurations concrètes. Nos algorithmes exploitent le fait qu'on peut retrouver l'identifiant d'un nœud dans une structure de graphe, et se basent sur trois fonctions supposées offertes par l'outil de recherche d'homomorphisme :

- MATCHGRAPHS (G_1: Graph, G_2: Graph) qui retourne la liste de tous les homomorphismes (*f*, *val*) du graphe G_1 vers le graphe G_2.
- VALUATEGRAPH (G: Graph, V: Valuation) qui retourne une copie du graphe *G* avec les étiquettes symboliques réécrites selon la valuation *V*.

- MERGE (V_1: Valuation, V_2: Valuation) qui retourne une valuation $V_1 \cup V_2$ si ses valuations en entrée sont cohérentes. Si elles ne sont pas cohérentes (elles attribuent des valeurs différentes à une variable d'étiquette), la fonction retourne NULL.

A titre d'exemple, les fonctions correspondantes de l'outil présenté dans [Guennoun 06] sont : MATCHGRAPHS, VALUATEVERTEX et MERGE, respectivement. Les nom de la deuxième fonction étant liée au fait que cet outil spécifique ne gère les variables d'étiquette que pour les nœuds (*vertex*), nous utiliserons le nom plus général VALUATEGRAPH dans la présentation des algorithmes de *GraphSeq*.

3. Principe de *GraphSeq*

Alors que la comparaison de deux graphes a été très étudiée dans la littérature, il y a eu comparativement peu de travaux sur l'appariement de séquences de graphes (voir [Conte et al. 04] pour une synthèse des travaux sur les problèmes d'appariement de graphes). Nous n'avons pas pu trouver de travaux existants directement réutilisables pour notre propos. Dans certains cas, les travaux existants traitent de problèmes plus simples. Par exemple, on recherche une sous-séquence de graphes "*g1 g2*" dans la séquence "*g0 g1 g2 g3*" : ceci peut se résoudre en combinant des algorithmes de recherche d'isomorphisme de graphe et de recherche de chaînes de caractères. Dans d'autre cas, les problèmes d'appariement visent des objectifs très différents des nôtres, comme l'inférence de prédictions à partir de l'analyse d'une série temporelle de graphes. Des exemples de telles inférences appliquées à l'analyse de réseaux dynamiques peuvent être trouvés dans [Bunke et al. 07].

A notre connaissance, les algorithmes qui se rapprochent le plus des nôtres se situent dans le domaine de l'analyse d'images vidéo. Dans [Shearer et al. 01], les auteurs recherchent des séquences de motifs (appelées requêtes graphiques, ou *pictorial queries*) dans une séquence de graphes extraits d'images vidéo. Une différence avec nous est cependant que les motifs ne contiennent pas de variables d'étiquettes, et qu'un nœud du motif correspond à au plus un objet dans une image. Dans notre cas, il peut y avoir plusieurs instances d'un motif P_i dans un graphe de configuration concrète C_j. En particulier, les instances du premier motif P_0 peuvent produire plusieurs possibilités pour le reste de la séquence, avec des valuations alternatives pour les variables qui sont nouvelles dans P_1, …, P_n (les variables qui ne sont pas nouvelles doivent garder leur valeur précédente). Le but de la recherche réalisée par *GraphSeq* est d'explorer toutes les possibilités, en retenant des valuations cohérentes tout au long de la séquence, et en identifiant les transitions d'un motif à un autre.

Pour expliquer le principe du raisonnement séquentiel, nous présentons d'abord la forme des résultats retournés par la recherche. Ensuite, nous donnons les propriétés attendues de ces résultats.

3.1. Forme des solutions recherchées par *GraphSeq*

GraphSeq prend deux séquences de graphes en entrée :

- Une séquence $P_0, \ldots, P_{m-1}$ de m graphes motifs,
- Une séquence $C_0, \ldots, C_{n-1}$ de n graphes de configurations concrètes.

Il calcule l'ensemble de tous les appariements de séquences, où un appariement identifie un sous-ensemble des nœuds concrets qui exhibent la succession de motifs attendus pour un certain intervalle de temps. La Figure 31 donne une intuition visuelle de cela. Le motif P_0 apparaît comme un sous-graphe du système dans les configurations concrètes C_1, C_2. Ensuite, il y a un changement de configuration système qui produit P_1. Et enfin, P_2 apparaît jusqu'au moment où le système passe dans la configuration C_7. Il est important de noter qu'un motif peut persister durant plusieurs configurations concrètes successives. La fenêtre temporelle calculée par *GraphSeq* sera utilisée ultérieurement quand on traitera la vue événementielle des scénarios. Par exemple, supposons qu'une description de scénario inclue un message de communication *msg* du nœud *n1* vers le nœud *n2*, lorsque le système est dans une configuration spatiale exhibant le motif P_2 de la Figure 31. Nous savons que ce message *msg* est à rechercher dans la sous-trace d'exécution qui commence à la date de l'événement de changement de configuration $C_3 \rightarrow C_4$, et qui termine à la date de $C_6 \rightarrow C_7$. L'appariement nous indique également quels nœuds concrets sont supposés être l'émetteur *n1* et le récepteur *n2* de *msg*.

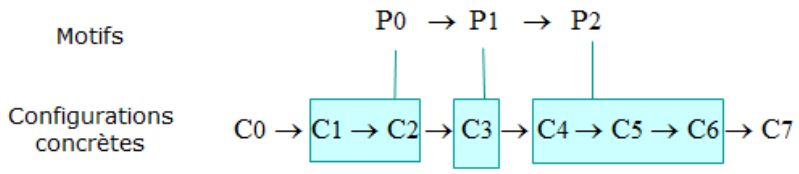


Figure 31. Appariement des séquences de graphes

Un appariement retourné par *GraphSeq* correspond à la structure de donnée suivante :

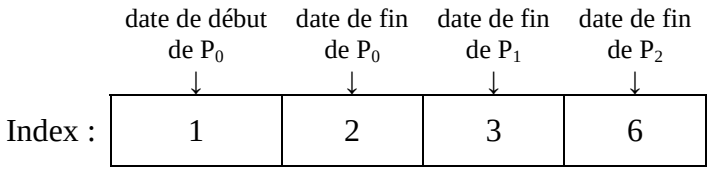
```
Structure Match
   Valuation val
   int index[0..m]
End Structure
```

La valuation *val* d'un match attribue une valeur concrète à toutes les variables qui apparaissent dans la séquence de graphes motifs. C'est un ensemble de couples *(NomVariable, Valeur)*, tel que chaque nom de variable apparaît dans exactement un couple. Notons que tous les sommets dans un graphe motif ont au moins une étiquette qui est une variable : c'est celle qui correspond à l'identifiant symbolique du nœud. Dans le prototype réalisé, l'identifiant a le type STRING. Grâce à cet identifiant, un nœud du motif est apparié à un unique nœud concret du système, et ce nœud peut être suivi d'un graphe à un autre.

Le tableau *index* d'un match donne une fenêtre temporelle pour chaque motif. Il est défini comme suit :

- *index[0]* est la date de début de l'instance de P_0 trouvée ; cela veut dire que la correspondance entre les séquences commence à partir du graphe $C_{index[0]}$.
- Pour $i > 0$, *index[i]* est la date de fin de P_{i-1} ; cela veut dire que le motif P_{i-1} persiste de $C_{index[i-1]+1}$ à $C_{index[i]}$.

Par exemple, la durée des motifs dans la Figure 31 est encodée comme suit :



L'encodage signifie que P_0 commence à apparier quand le système exhibe la configuration C_1 et finit quand le système quitte C_2. La date de début pour le motif P_1 est implicite : elle est définie comme *1 + date de fin de* P_0, c'est-à-dire *3* dans cet exemple. La date de fin de P_1 est, ensuite, donné explicitement. Ceci produit la fenêtre temporelle [3, 3] pour P_1. D'une manière similaire, la fenêtre temporelle pour P_2 est [4, 6].

Nous allons maintenant énoncer les propriétés que doivent satisfaire les matchs retournés par la recherche.

3.2. Propriétés attendues

Supposons que *M* soit un match retourné par *GraphSeq*, et que $[s_i, e_i]$ soit la fenêtre temporelle pour chaque motif P_i comme indiqué dans *M.index*. De façon évidente, si le match est correct, chaque configuration concrète C_{si},…, C_{ei} doit contenir l'instance de P_i déterminée par la valuation *M.val*.

Propriété 1. $\forall i \in [0, m-1]$, $\forall k \in [s_i, e_i]$, il existe une fonction f telle que:

$$\text{MATCHGRAPHS}(\text{VALUATEGRAPH}(P_i, M.val), C_k) = \{(f, \varnothing)\}$$

Notons que, pour chaque paire (*i*, *k*), nous comparons deux graphes sans variable d'étiquette, d'où la valuation vide de l'homomorphisme résultant. De plus, si la fonction *f* existe alors elle est nécessairement unique, car la valuation des identifiants de nœuds du motif ne laisse qu'une possibilité pour les apparier aux nœuds de la configuration concrète.

Une valuation cohérente des variables, et plus spécifiquement des identifiants des nœuds symboliques, doit également tenir compte des nœuds qui apparaissent et disparaissent. La Figure 32 montre un exemple. Pour faciliter la discussion, nous ne faisons figurer que les étiquettes correspondant aux identifiants de nœuds, et nous omettons toutes les autres étiquettes.

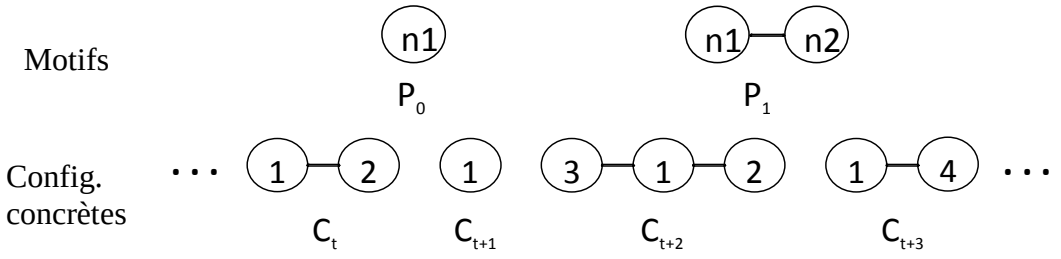
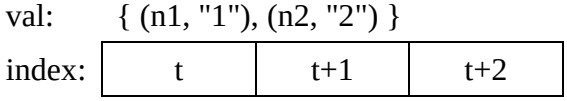


Figure 32. Un exemple avec des noeuds qui apparaissent et disparaissent

Dans cet exemple, le motif P_1 introduit un nouveau nœud d'identifiant symbolique *n2*, qui n'était pas présent dans le motif P_0. Nous exigeons que la valuation de *n2* corresponde à un nœud concret qui n'était jamais apparu dans les configurations concrètes exhibant le motif P_0. Par exemple, le match suivant n'est pas retenu, même s'il satisfait la propriété 1.



Le nœud concret *2* ne peut pas jouer le rôle du nouveau nœud symbolique *n2*, car il était déjà apparu dans la configuration C_t. Cependant, le match suivant est accepté.

val: { (n1, "1"), (n2, "2") }

index:	t+1	t+1	t+2

Au travers de cet exemple, on voit que le raisonnement séquentiel doit prendre en considération le fait que des nœuds puissent être interdits dans certaines configurations. Soit F_Id_i l'ensemble des valeurs concrètes (déterminées par *M.val*) pour les identifiants symboliques qui n'apparaissent pas dans P_i, mais sont présents dans certains P_j pour $j \neq i$.

Propriété 2. $\forall\, i \in [0, m-1]$, $\forall\, k \in [s_i, e_i]$, C_k n'a pas de sommet étiquetté par un identifiant dans F_Id_i.

Enfin, nous exigeons que chaque fenêtre temporelle de motif $[s_i, e_i]$ soit maximale. Cela signifie que s_i-1 ne serait pas une date de début pour P_i avec la valuation *M.val*, et $1+e_i$ ne serait pas une date de fin. Ainsi, en reprenant l'exemple de la Figure 32, le match suivant n'est pas maximal, car la fenêtre temporelle de P_0 peut commencer plus tôt (en *t*).

val: { (n1, "1"), (n2, "3") }

index:	t+1	t+1	t+2

Propriété 3. $\forall\, i \in [0, m-1]$, soit P_i' = VALUATEGRAPH (P_i, *M.val*). La fenêtre temporelle $[s_i, e_i]$ est maximale, c'est-à-dire :

- $s_i = 0$, ou il n'existe pas d'homomorphisme entre P_i' et $C_{(Si)-1}$, ou $C_{(Si)-1}$ contient un sommet avec un identifiant dans F_Id_i.
- $e_i = n-1$, ou il n'existe pas d'homomorphisme entre P'_i et C_{1+ei}, ou C_{1+ei} contient un sommet avec un identifiant dans F_Id_i.

La propriété 3 implique que les dates de début et de fin correspondent réellement à des événements de changements de configurations. Un scénario avec deux motifs successifs identiques ne produira pas de match (la recherche sera infructueuse). La transition de P_i à P_{i+1} doit être provoquée par un changement de connectivité ou d'attributs des nœuds concrets, par l'introduction de nouveaux nœuds, la suppression de nœuds, ou par toute combinaison de ces changements.

Avec l'introduction de nouveaux nœuds, le calcul des dates de début et de fin peut devenir délicat. Par exemple, dans la Figure 32, si le nœud "1" joue le rôle du nœud *n1*, la date de fin de P_0 peut être *t+1* (juste avant l'apparition du nœud "2" ou "3", si l'un de ces noeuds est retenu pour jouer le rôle de *n2*), *t+2* (avant l'apparition du nœud "4") ou plus tard. De plus, selon le nœud concret qui sera retenu pour jouer le rôle de *n2*, une date candidate pour le début de P_0 (par exemple, *t+1*) pourra produire – ou non – une fenêtre temporelle maximale.

L'objectif de *GraphSeq* est d'explorer tous les choix alternatifs, et de construire l'ensemble de matchs qui satisfont les propriétés 1, 2 et 3.

Nous allons, dans les paragraphes suivants, présenter cet algorithme de construction de matchs. Pour faciliter la compréhension, nous commencerons par une version simplifiée de l'algorithme, qui traite le cas où tous les motifs impliquent le même ensemble de nœuds. Ensuite, nous expliquerons comment l'algorithme est étendu pour prendre en compte les

nœuds qui apparaissent et disparaissent. Dans chaque cas, nous prendrons un exemple pour illustrer les différentes étapes de l'algorithme.

4. Algorithme de *GraphSeq* pour un ensemble fixe de nœuds dans les motifs

Dans cette version simplifiée de l'algorithme, nous supposons que la séquence de motifs ne décrit pas d'apparition ou de disparition de nœuds. Les conséquences sont les suivantes : (i) la propriété 2 va être trivialement satisfaite, car l'ensemble *F_id* des nœuds interdits est vide, (ii) dans la propriété 3, les conditions que doivent satisfaire les dates de début et de fin des motifs sont plus simples. Nous donnons ci-dessous une reformulation de la propriété 3.

Propriété 3 pour un ensemble fixe de nœuds dans les motifs. $\forall i \in [0, m-1]$, soit P_i' = ValuateGraph (P_i, *M.val*). La fenêtre temporelle $[s_i, e_i]$ est maximale, c'est-à-dire :

- $s_i = 0$, ou MATCHGRAPHS(P_i', $C_{(Si)-1}$) = {}.
- $e_i = n-1$, ou MATCHGRAPHS(P_i', C_{1+ei}) = {}.

Notre présentation de l'algorithme commence par une vue générale de la structure de contrôle. Puis, nous en détaillons chaque étape, en illustrant le fonctionnement sur un exemple de séquences de graphes.

4.1. Structure de contrôle

Afin de construire progressivement des matchs, *GraphSeq* utilise une structure de données intermédiaire dénommée *PartialMatch* :

```
Structure PartialMatch
   Valuation val
   int index[0..m]
    int depth
End Structure
```

La structure de données *PartialMatch*, utilisée pour représenter un match partiel, est comme la structure *Match* présentée précédemment, mais avec un champ additionnel *depth*. La valeur contenue dans ce nouveau champ représente le nombre de motifs qui ont été appariés avec succès avec des graphes de configuration. Nous appellerons cette valeur la profondeur du match partiel. Par exemple, une profondeur de valeur *i* indique que nous avons pu trouver une sous-séquence de motifs $P_0,\ldots, P_{i-1}$ dans les graphes de configuration, mais qu'il nous reste les motifs $P_i,\ldots, P_{m-1}$ à traiter. Si nous reprenons l'exemple de la Figure 32, un match partiel trouvé par *GraphSeq* est :

val:	{ (n1, "1")}		
index:	t+1	t+1	-1
depth:	1		

Dans le champ *index*, la valeur non significative *-1* signifie que les dates de fin des motifs qui restent à traiter sont encore indéterminées.

Un match partiel de profondeur *i* est étendu par le traitement du motif suivant P_i, qui permet d'obtenir un match partiel de profondeur *i+1*. Par exemple, une extension incrémentale du match partiel précédent est :

val:	{ (n1, "1"), (n2, "2") }		
index:	t+1	t+1	t+2
depth:	2		

Quand un match partiel a été étendu jusqu'à une profondeur *m*, alors un match complet a été trouvé.

GraphSeq utilise un algorithme de parcours en profondeur (*depth-first search* - DFS) pour étendre les matchs partiels. C'est-à-dire que s'il existe plusieurs extensions possibles à un match partiel, alors l'outil explorera une branche aussi loin que possible avant de faire machine arrière pour explorer les autres. La structure de contrôle de DFS est présentée dans la Figure 33. Elle utilise une pile *L* de type LIFO (*Last In First Out* - dernier arrivé, premier sorti) pour stocker les matchs partiels à traiter. Notons que la boucle de plus haut niveau de l'algorithme, représentée par la structure de contrôle *for()*, impose que la date de commencement du motif P_0 soit inférieure à *n-m*. La raison est que si elle est supérieure à cette valeur, le nombre de configurations concrètes restant à examiner sera inférieur au nombre de motifs restant à traiter.

```
Let L be an empty stack of PartialMatch elements
For (i=0; i≤n-m; i++)
   build all partial matches of depth 1 with start date index[0] = i,
   push each of them in L
   While L is not empty
      Let pm = pop (L)
      If (pm.depth < m)
         Build all one step extensions of pm,
         push each of them in L
      Else   // found
         Write pm.val and pm.index in output file
      Endif
   End While
End For
```

Figure 33. La structure de contrôle DFS

Les différentes étapes de l'algorithme de la Figure 33 sont indiquées en gras. Elles concernent la création de matchs partiels de profondeur 1, et l'extension des matchs partiels. Nous allons maintenant les détailler.

4.2. Détail des étapes de l'algorithme

4.2.1. Exemple illustratif

Nous allons utiliser l'exemple suivant pour illustrer les étapes de l'algorithme. La Figure 34 montre une séquence de deux motifs, ainsi qu'un début de séquence de configurations concrètes. Rappelons que nous sommes dans le cas où l'ensemble de nœuds dans les motifs est fixe (ici, c'est l'ensemble {*id1*, *id2*}). Il n'y a donc pas de nœuds interdits dans les configurations, et nous devons garantir la propriété 1 et la version simplifiée de la propriété 3 donnée plus haut.

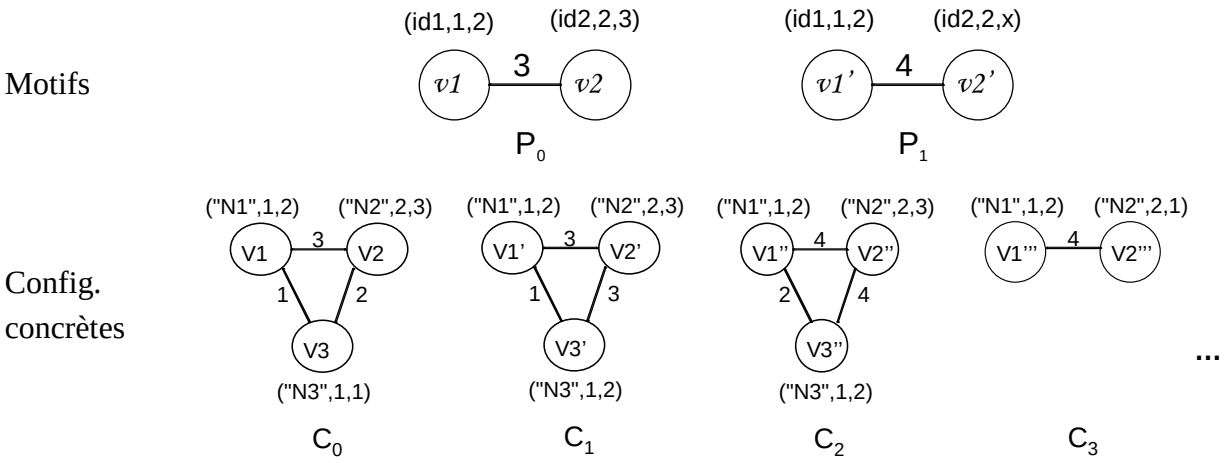


Figure 34. Exemple de graphes avec un match commençant à la date 0 et finissant à la date 2

Un des matchs à trouver est :

val: { (id1, "N1"), (id2, "N2"), (x, 3) }

index:	0	1	2

La construction de ce match sera expliquée en suivant l'algorithme détaillé.

4.2.2. Création de matchs partiels de profondeur 1

Étant donné une valeur candidate *i* pour une date de début, la recherche est initiée en regardant toutes les possibilités de matchs partiels pour P_0. Ainsi, cette procédure produit un ensemble de matchs partiels de profondeur 1. L'algorithme correspondant est décrit dans la Figure 35a et les fonctions auxiliaires dans la Figure 35b.

Déroulons cet algorithme avec l'exemple de la Figure 34. Nous commençons avec la valeur candidate 0 comme date de début. D'abord, les graphes P_0 et C_0 sont comparés (appel à MATCHGRAPHS( )). L'outil trouve un homomorphisme (f_0, val_0) avec :

f_0: *v1* → V1 val_0 = { (id1, "N1"), (id2, "N2") }

v2 → V2

La fonction d'association f_0 utilise les références de sommets *v1*, *v2*, *V1*, *V2* qui sont spécifiques à l'encodage interne des graphes. Ici, l'information intéressante est la valuation fournie pour construire les homomorphismes : dorénavant, le nœud concret *N1* jouera le rôle du nœud abstrait *id1* et le nœud *N2* jouera le rôle du nœud abstrait *id2*. Nous devrons continuer la recherche avec ces choix de valuation. Soit *P'* l'instance de P_0 avec cette valuation, construite par appel à VALUATEGRAPH ( ).

Maintenant, nous devons déterminer la fenêtre temporelle de cette instance *P'* du motif.

La vérification de la date de début, au sens de la propriété 3, s'effectue dans la conditionnelle *If (i>0) … Endif*. Comme la date de début est *0*, elle satisfait trivialement la condition de maximalité. Pour une date $i > 0$, il faudrait vérifier que l'instance de motif trouvée n'était pas déjà contenue dans la configuration concrète C_{i-1}.

```
// build all partial matches of depth 1
// with start date i and push each of them in L
Let H = MatchGraphs (P0, Ci)
While H is not empty
   Extract h from H
   Let P' = ValuateGraph (P0, h.val)
   // Is i a start date for P0?
   Let start_OK = true
   If (i > 0)
      Let H' = MatchGraphs (P', Ci-1)
      If (H' is not empty)
         // previous config did match --> not a start date
         start_OK = false
      Endif
   Endif
   // Now, check the end date
   If (start_OK = true)
      Let end = ComputeEndDate(P', i)
      If (end ≤ n-m)
         // end date is not too late
         let pm = CreatePartialMatchD1 (h.val, i, end)
         push pm in L
      Endif
   Endif
End While
```

(a) Algorithme principal

```
int ComputeEndDate (Graph G, int start)
  Let i = 1
  Repeat
    Let H = MatchGraphs(G,Cstart+i)
      If (H is not empty) then
         i = i+1
      Endif
  Until (H is empty or start+i = n)
  Return (start+i-1)
End ComputeEndDate
```

```
PartialMatch CreatePartialMatchD1
(Valuation v, int start, int end)
   Let pm be a Partial match
   pm.val = v
   pm.index[0] = start
   pm.index[1] = end
   For (i=2; i≤m; i++)
      pm.index[i] = -1
   End For
   pm.depth = 1
   return (pm)

End CreatePartialMatchD1
```

(b) Fonctions auxiliaires

Figure 35. *Matchs* partiels de profondeur 1, commençant à la date i

Une fois la date de début vérifiée avec succès, nous devons calculer la date de fin. Ceci s'effectue dans la conditionnelle *If (start_OK = true) … Endif*. L'appel à la fonction ComputeEndDate( ) renvoie une date de fin maximale au sens de la propriété 3, c'est-à-dire que : soit la date de fin est *n-1*, soit c'est une date *end* telle que C_{1+end} ne contient pas *P'* comme sous-graphe. Dans notre exemple, ComputeEndDate( )va faire deux appels successifs à MatchGraphs (). Un premier appel va comparer *P'* et C_1 : un homomorphisme est trouvé, ce ce qui signifie que l'instance du motif est toujours présente en C_1. Un deuxième appel infructueux va montrer que *P'* n'est plus dans C_2. La date de fin est donc *1*. Dans l'algorithme principal, on vérifie alors que cette date de fin n'est pas trop tardive compte tenu du nombre

de motifs qu'il reste à traiter *(end ≤ n-m)*. Comme ici elle n'est pas trop tardive, le match partiel suivant est créé :

val:	{ (id1, "N1"), (id2, "N2") }		
index:	0	1	-1
depth	1		

En résumé, la création de matchs partiels *pm* de profondeur 1 présente les caractéristiques suivantes :

- Les différents appels à `MATCHGRAPHS()` assurent que toutes les configurations concrètes entre la date de début *i = pm.index[0]* et la date de fin *end = pm.index[1]* contiennent l'instance *P'* de P_0 déterminée par *pm.val* (propriété 1).
- Les dates de début et de fin sont maximales au sens de la propriété 3.
- La date de fin n'est pas trop tardive vis-à-vis du nombre de motifs qu'il reste à traiter.

4.2.3. Extension des matchs partiels

L'étape suivante de l'algorithme est l'extension d'un match partiel de profondeur *d*, pour produire des matchs partiels de profondeur *d+1*, décrite dans la Figure 36. Ici, la date de début (*start*) est fixée : elle vient juste après la date de fin du motif précédent. Le motif courant est P_d. Comme il faut garder les choix de valuation déjà effectués lors du traitement des motifs P_0, … P_{d-1}, on va créer une version partiellement valuée *P'* du motif courant, et chercher si *P'* est contenu dans C_{start} (appel à `MATCHGRAPHS`). Ceci induit éventuellement de nouveaux choix de valuation, pour les variables restantes dans *P'*. Pour chaque extension candidate, la date de fin doit être calculée et vérifiée. L'appel à `COMPUTEENDDATE()` utilise une version complètement valuée *P''* du motif, après fusion des valuations précédentes (*pm.val*) et nouvelles (*h.val*).

Reprenons l'exemple. Nous avons construit un match partiel de profondeur 1. Nous continuons avec le motif suivant (P_1) de la séquence de motifs. Nous avons besoin de retenir les valeurs attribuées aux nœuds à l'étape précédente. Nous cherchons donc un homomorphisme du motif *VALUATEGRAPH(P_1, pm.val)* en C_2. On trouve un homomorphisme *h* avec la valuation *h.val = {(x, 3)}*. Ensuite, nous fusionnons les valuations qui ont été produites jusqu'à maintenant. Cette fusion donne la valuation *v = {(id1, "N1"), (id2, "N2"), (x, 3)}*. La date de fin d'appariement du motif P_1 est *2*, car il n'existe pas d'homomorphisme de *P'' = VALUATEGRAPH(P_1,v)* en C_3 : la dernière étiquette du sommet *V2'''* n'a pas la valeur attendue *3*. Le résultat final est le match attendu.

Au travers de cette section, il est évident que le problème principal est de retenir des choix de valuation cohérents au travers de la séquence. Cela peut devenir très complexe quand les nœuds apparaissent, disparaissent et réapparaissent de manière dynamique dans les motifs. Le paragraphe suivant discutera ce cas.

```
// Build all one step extensions of pm,
// push each of them in L
// Here, pm.depth < m

Let start = 1+pm.index[pm.depth]
Let P' = ValuateGraph (P_depth, pm.val)
H = MatchGraphs (P', C_start)

While H is not empty
   Extract h from H
   Let v = Merge (pm.val, h.val)
   If (v!=NULL) // compatible valuations
      Let P'' = ValuateGraph (P_depth, v)
      Let end = ComputeEndDate(P'', start)
      If (end ≤ n-m+depth)
         // end date is not too late
         let pm = CreateExtendedMatch (pm, end, v)
         push pm in L
      Endif
   Endif
End While
```

(a) Algorithme principal pour étendre un match partiel

```
PartialMatch CreateExtendedMatch (PartialMatch father, int end, Valuation v)
   Let pm be a PartialMatch
   pm.val = v
   For (i=0; i≤father.depth; i++)
      pm.index[i] = father.index[i]
   End For
   pm.index[depth+1] = end
   For (i= depth+2; i≤m; i++)
      pm.index[i] = -1
   End For
   pm.depth = 1+ father.depth
   return (pm)
End CreateExtendedMatch
```

(b) Fonction auxiliaire

Figure 36. Etendre un match partiel *pm*

5. Algorithme avec les créations/disparitions de nœuds

Maintenant, nous allons considérer le cas où les nœuds varient dans les graphes de motifs. Le principe de recherche de match est le même que dans l'algorithme simplifié précédent. À chaque extension de match partiel, nous retenons des choix de valuation cohérents vis-à-vis de ceux déjà effectués. En plus, nous devons faire attention aux nouveaux nœuds, et aux nœuds qui disparaissent, pour garantir la propriété 2. Le calcul des dates de début et de fin pour les instances de motif (propriété 3) est également plus complexe.

5.1. Structure de contrôle

Sur le principe, la structure de contrôle DFS ne change pas. Cependant, il y a des impacts sur quelques parties (indiquées en gras dans la Figure 37) :

- Une étape de prétraitement est effectuée avant d'entrer dans la recherche des matchs. Les graphes motifs sont analysés pour identifier les nœuds qui apparaissent ou disparaissent. Cette étape a pour objectif de préparer la vérification des propriétés 2 et 3.
- Les informations extraites à partir de l'étape de prétraitement ont un impact à la fois sur le calcul des matchs partiels de profondeur 1, et sur les extensions successives des matchs partiels.
- Une vérification finale est ajoutée avant de retenir un candidat de match complet. Nous verrons que cette vérification est liée à la propriété 3.

```
Preprocessing of pattern data
Let L be an empty stack of PartialMatch elements
For (i=0; i≤n-m; i++)
   build all partial matches of depth 1 with start date i,
   push each of them in L
   While L is not empty
      Let pm = pop (L)
      If (pm.depth < m)
         Build all one step extensions of pm,
         push each of them in L
      Else   // found
         If (FINALCHECK(pm))
            Write pm.val and pm.index in output file
         Endif
      Endif
   End While
End For
```

Figure 37. Modification de la structure de contrôle DFS

5.2. Détail des étapes de l'algorithme

5.2.1. Exemple illustratif

La Figure 38 montre un exemple avec la création et la disparition de nœuds dans la séquence de motifs.

Dans cet exemple, le nœud avec l'identifiant symbolique *id2*, présent dans le motif P_0, disparaît ensuite dans P_1, P_2. Le motif P_2 introduit un nouveau nœud avec l'identifiant *id3*, qui ne doit pas être présent dans les configurations concrètes exhibant P_0, P_1.

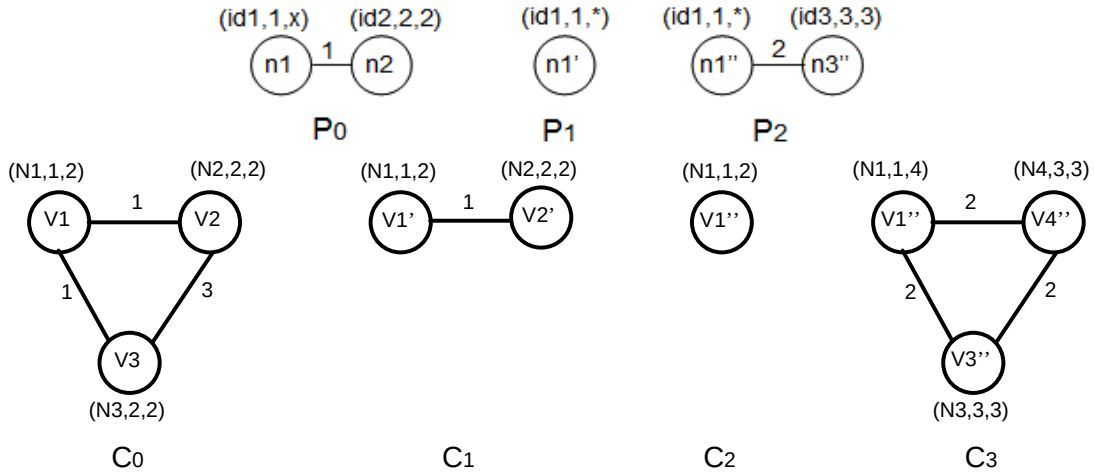


Figure 38. Exemple avec les créations, disparitions des nœuds

5.2.2. Prétraitement des graphes motifs

Cette étape extrait, pour chaque motif P_j, l'ensemble des identifiants symboliques de nœuds qui étiquettent les sommets. Ensuite, elle calcule les informations suivantes, stockées dans une structure globale de données :

- Pour $1 \leq j \leq m-1$, *NewNodes(j)* est l'ensemble des identifiants symboliques de nœuds qui sont présents dans P_j, mais qui ne l'étaient pas dans P_0, .., P_{j-1}.
- Pour $1 \leq j \leq m-1$, *ForbiddenNodes(j)* est l'ensemble des identifiants symboliques de nœuds qui sont apparus au moins une fois dans P_0, ..., P_{j-1}, mais qui ne sont pas présents dans P_j.
- *ForbiddenNodes(0)* est l'ensemble des identifiants symboliques de nœuds qui sont apparus au moins une fois dans P_1, ..., P_{m-1}, mais qui ne sont pas présents dans P_0. Sa définition est, donc, différente de celle de *ForbiddenNodes(j)* où $j > 0$.
- Pour $0 \leq j \leq m-2$, *StopBefore(j)* est une valeur booléenne qui indique si la transition de P_j à P_{j+1} peut s'effectuer avant que P_j ne cesse d'être exhibé par une configuration concrète. Tel est le cas quand P_{j+1} ne diffère de P_j que par la présence de nœuds supplémentaires (soit nouveaux, soit issus d'une réapparition). Nous mettons *StopBefore(m-1) = false*.

Les ensembles *NewNodes* et *ForbiddenNodes* sont liés aux ensembles *F_Id* mentionnés dans les propriétés 2 et 3. Pour $j=0$, F_Id_j est l'ensemble des nœuds concrets appariés aux nouds symboliques de *ForbiddenNodes(0)*. Pour $1 \leq j \leq m-1$, notre algorithme distingue deux sous-ensembles de nœuds interdits : ceux dont l'identifiant concret est connu quand on traite le motif P_j, c'est-à-dire *ForbiddenNodes(j)*, dont on pourra vérifier l'absence à ce moment ; ceux dont l'identifiant concret n'est pas encore connu quand on traite le motif P_j, c'est-à-dire $\bigcup_{k\in[j+1,m-1]} NewNodes(k)$, dont l'absence ne sera vérifiée que lorsqu'ils auront reçus une valuation concrète.

Avec l'exemple de la Figure 38, le prétraitement calcule les ensembles suivants :

- NewNodes(1) = Ø, NewNodes(2) = {id3}
- ForbiddenNodes(1) = {id2}, ForbiddenNodes(2) = {id2}
- ForbiddenNodes(0) = {id3}
- StopBefore(0) = false, StopBefore(1) = true, StopBefore(2) = false

```
// Auxiliary computation AllNodes to be used for NewNodes and ForbiddenNodes
Let AllNodes(0) = N0
For (j=1; j≤m-1; j++)
   Let AllNodes(j) = Nj ∪ AllNodes(j-1)
End For

// New nodes
NewNodes(0) = ∅
For (j=1; j≤m-1; j++)
 NewNodes(j)= Nj \ AllNodes(j-1)
End For

// Forbidden Nodes
For (j=1; j≤m-1; j++)
   ForbiddenNodes (j)= AllNodes(j-1) \ Nj
End For
ForbiddenNodes (0) = AllNodes(m-1) \ N0

// Stop before?
For (j=0; j≤m-2; j++)
   If (Nj+1 ⊆ Nj)
      // No node expected to appear
      StopBefore(j) = false
   Else If (Nj ∩ ForbiddenNodes(j+1) ≠ ∅)
      // At least one node expected to disappear
      StopBefore(j) = false
   Else
      if there exists vertices v1 from Pj and v2 from Pj+1 such that
      v1 and v2 share the same symbolic ids, and one of the vertex integer label
      is constant in both cases, but with different values
      // At least one vertex label is expected to change
      StopBefore(j) = false
   Else
      if there exists edges e1 from Pj and e2 from Pj+1 such that
      they connect pairs of vertices with the same symbolic ids, and the edge label
      is constant in both cases, but with different values
      // At least one edge label is expected to change
      StopBefore(j) = false
   Else // transition from Pj to Pj+1 may occur before Pj ceases to be matched
      StopBefore(j) = true
   End if
End For
StopBefore(m-1) = false
```

Figure 39. Calcul de prétraitement

La Figure 39 décrit les calculs correspondants. Dans les calculs, N_j dénote l'ensemble des identifiants de nœuds dans le motif P_j, et *AllNodes(j)* l'ensemble des identifiants de nœuds dans les motifs $P_0 \ldots P_j$. L'extraction initiale de N_j, ainsi que les comparaisons des étiquettes dans le calcul de *StopBefore(j)* ne sont pas détaillées ici, car elles dépendent de l'encodage précis des graphes dans l'outil.

L'information calculée ci-dessus est utilisée pour implémenter des vérifications additionnelles pendant la recherche de matchs, comme montré de la Figure 40 à la Figure 42.

Dans ces figures, les nouvelles parties sont indiquées en gras. Bien évidemment, s'il n'y a pas de création ou de disparition de nœuds dans les motifs, l'algorithme est équivalent à celui présenté dans le paragraphe précédent (Figure 35 à Figure 36). Notons que, pour un ensemble fixe des nœuds dans les motifs, les ensembles *NewNodes(j)* et *ForbiddenNodes(j)* sont vides, et *StopBefore(j)* est faux pour tout *j*.

5.2.3. Impact sur la construction des matchs partiels de profondeur 1

L'impact sur la recherche des matchs partiels de profondeur 1 est montré dans la Figure 40. Il y a deux modifications majeures qui correspondent aux vérifications additionnelles des dates de début et fin, respectivement. La vérification de la date de début dépend maintenant de *ForbiddenNodes(0)* (voir la condition du premier *If*). Si cet ensemble est vide, la décision reste inchangée par rapport à la version simplifiée de l'algorithme. Elle est identique à celle qui est dans la Figure 35 : nous ne retenons pas la date candidate *i* si C_{i-1} contenait déjà l'instance de P_0 trouvée dans C_i. Par contre, si cet ensemble n'est pas vide, nous retenons toujours la date candidate *i*. Cette décision peut être expliquée en se référant à l'exemple de la Figure 38, et en prenant *N1* et *N2* pour jouer le rôle de *id1* et *id2*. Dans cet exemple, la date de début *1* est retenue car *ForbiddenNodes(0)* n'est pas vide (il contient le nœud symbolique *n3*). De fait, *1* est bien la date de début d'un match complet :

val:	{ (id1, "N1"), (x, 2), (id2, "N2"), (id3, "N3") }			
Index:	1	1	2	3

A la profondeur 1, on ne peut pas encore déterminer si *i* est réellement une date de début au sens de la propriété 3, car il faudrait avoir la valuation de tous les identifiants symboliques dans *ForbiddenNode(0)*. La décision sera prise par la vérification finale (`FINALCHECK()` dans la structure de contrôle dans la Figure 37) que nous allons décrire plus tard.

En ce qui concerne la date de fin, la nouvelle partie de la Figure 40 correspond au cas où *StopBefore(0)* est vrai. L'algorithme prend en compte le fait que la transition du motif P_0 au motif P_1 peut avoir lieu avant que P_0 ne cesse d'être exhibé. Tel n'est pas le cas dans l'exemple de la Figure 38, car la transition est provoquée par la disparition du nœud symbolique *id2* : ainsi, P_0 cessera nécessairement d'être exhibé. Mais dans le cas où *StopBefore(0)* serait vrai, nous devrions retenir plusieurs dates de fin candidates : toutes les valeurs possibles entre la date de début *i* et *min(end, n-m)*. Un match partiel serait créé pour chaque valeur. Nous reviendrons sur cette possibilité d'avoir plusieurs dates de fin possibles quand nous commenterons la transition entre les motifs P_1 et P_2 de l'exemple, dans le paragraphe suivant.

Une autre différence se situe dans la fonction *COMPUTEENDDATE()* qui détecte une fin de match partiel quand il devient impossible de trouver un homomorphisme. La fonction a maintenant deux paramètres supplémentaires. En fait, ces deux derniers paramètres permettraient la détection de fin de match dans le cas suivant : un graphe de configuration concrète contient des nœuds interdits. À la profondeur 1, les valeurs ∅ et NULL sont passées à la fonction, car aucun identifiant concret de nœud interdit n'est connu à cette étape de l'algorithme. Dans ce cas, le comportement de l'algorithme reste le même que dans la version simplifiée qui ne comportait pas les deux derniers paramètres. Nous verrons leur utilité aux profondeurs supérieures à 1, lors de l'extension d'un match partiel.

```
// build all partial matches of depth 1 with start date i,
// push each of them in L
Let H = MATCHGRAPHS (P0, Ci)
While H is not empty
   Extract h from H
   Let P' = VALUATEGRAPH (P0, h.val)
   // Is i a start date for P0?
   Let start_OK = true
   If (i > 0 && ForbiddenNodes (0)= ∅)
      Let H' = MATCHGRAPHS (P', Ci-1)
      If (H' is not empty)
         // previous config did match --> not a start date
         start_OK = false
      Endif
   Endif
   // Now, check the end date
   If (start_OK = true)
      Let end = COMPUTEENDDATE(P', i, ∅,NULL)
      If (StopBefore(0))
      // can stop at an intermediate date
         For (j=i; j<=min(end,n-m); j++)
            let pm = CREATEPARTIALMATCHD1(h.val, i, j)
            push pm in L
         End for
      Else If (end ≤ n-m)
         // end date is not too late
         let pm = CREATEPARTIALMATCHD1 (h.val, i, end)
         push pm in L
      Endif
   Endif
End While
```

(a) Algorithme principal

```
ListOfHomomorphisms NEWMATCHGRAPHS
(Graph G1, Graph G2,SetOfStrings
ForbidIds, Valuation v)
   // None of the forbidden
   // ids in G2?
   For each f in ForbidIds
      Let concreteId be v(f)
      If (concreteId exists in G2)
         Return (empty list)
      End if
   End For
 Return (MATCHGRAPHS(G1, G2))
End NEWMATCHGRAPHS
```

```
int COMPUTEENDDATE (Graph G, int
start, SetOfStrings ForbidIds,
Valuation v)
   Let i = 1
   Repeat
      Let H = NEWMATCHGRAPHS (G,
Cstart+I, ForbidIds,v)
      If (H is not empty) then
         i = i+1
      Endif
   until (H is empty or start+i = n)
   return (start+i-1)
End COMPUTEENDDATE


PartialMatch CREATEPARTIALMATCHD1 ()
   Same as previously
   See Figure 35.b
End CreatePartialMatchD1
```

(b) Fonctions auxiliaires

Figure 40. Modification de la construction de *Matchs* partiels de profondeur 1, commençant à la date i

En résumé, si on déroule l'algorithme de création de matchs partiels de profondeur 1 sur l'exemple de la Figure 38, on trouve les résultats suivants.

Pour la date de début *0*, l'algorithme trouve deux possibilités, *pm1* et *pm2*.

pm1 :

val: { (id1, "N1"), (x, 2), (id2, "N2") }

index:	0	1	-1	-1

Depth 1

pm2 :

val: { (id1, "N1"), (x, 2), (id2, "N3") }

index:	0	0	-1	-1

Depth 1

Pour la date de début *1*, l'algorithme trouve une possibilité, *pm3*.

pm3 :

val: { (id1, "N1"), (x, 2), (id2, "N2") }

index:	1	1	-1	-1

Depth 1

Les dates *2* et *3* sont trop tardives et ne sont pas explorées.

Ces matchs partiels présentent les caractéristiques suivantes :

- Toutes les configurations concrètes entre la date de début *pmk.index[0]* et la date de fin *pmk.index[1]* contiennent l'instance *P'* de P_0 déterminée par *pmk.val* (propriété 1).
- Les dates de début *0* sont maximales au sens de la propriété 3. Pour la date de début *1*, sa maximalité reste à déterminer lors de la vérification finale, si *pm3* peut être étendu jusqu'à la profondeur *3*.
- Les dates de fin sont maximales : puisque *StopBefore(0) = false*, on a retenu une date de fin telle que la configuration concrète suivante (ici, C_2) ne contienne plus l'instance *P'* de P_0.
- On ne peut encore rien dire sur la satisfaction de la propriété 2, car on ne connaît pas l'identifiant concret du nœud interdit *id3*.

5.2.4. Impact sur l'extension d'un match partiel

La nouvelle version de l'extension des matchs partiels est montrée dans la Figure 41. Notons que la recherche des homomorphismes maintenant doit prendre en compte les ensembles *ForbiddenNodes* (lors de l'appel aux fonctions *NEWMATCHGRAPHS* et *COMPUTEENDDATE()*). Rappelons que pour $1 \leq j \leq m-1$, *ForbiddenNodes(j)* est l'ensemble des identifiants symboliques de nœuds qui sont apparus au moins une fois dans P_0, .., P_{j-1}, mais qui ne sont pas dans P_j. Des valeurs concrètes ont déjà été attribuées à ces identifiants dans le

match partiel *pm*. Nous connaissons ainsi des identifiants des nœuds qui ne doivent pas être présents dans les configurations concrètes qui s'apparient avec P_j. Nous utilisons cette information pour rejeter des extensions de match partiel (voir la fonction *NEWMATCHGRAPHS* de la Figure 40b). Dans l'exemple, le nœud symbolique *id2* est absent des motifs P_1 et P_2. Les extensions des matchs partiels *pm1* et *pm3* devront correspondre à des configurations concrètes ne contenant pas *N2*, tandis que pour *pm2* le nœud concret *N3* est interdit.

Une vérification spéciale pour les nouveaux nœuds est également introduite (*CHECKNEWNODES*). Des valeurs concrètes sont attribuées à leurs identifiants symboliques à l'étape courante (dans *h.val*). Il faut faire attention que ces valeurs correspondent réellement à des nouveaux nœuds. C'est-à-dire qu'il faut vérifier que les nœuds concrets ne sont jamais apparus dans $C_{pm.index[0]}$, …, $C_{pm.index[pm.depth]}$. Dans l'exemple, les extensions de *pm1* ne pourront pas donner le match complet suivant :

val: { (id1, "N1"), (x, 2), (id2, "N2"), (id3, "N3") }

Index:	0	1	2	3

En effet, l'identifiant *id3* représente un nouveau nœud dans le motif P_2, et le nœud *N3* était déjà présent dans la configuration C_0. Donc, il ne peut pas prendre la valeur *N3* sans violer la propriété 2 de l'algorithme.

```
// Build all one step extensions of pm,
// push each of them in L
// Here, pm.depth < m

Let start = 1+pm.index[pm.depth]
Let P' = ValuateGraph (P_depth, pm.val)
H = NewMatchgraphs (P', C_start, ForbiddenNodes(pm.depth), pm.val)

While H is not empty
   Extract h from H
   Let v = Merge (pm.val, h.val)
   If (v!=NULL&&CheckNewNodes(depth,pm.index[0],pm.index[pm.depth],h.val))
   // compatible valuations
   // and new nodes are really new
      Let P'' = ValuateGraph (P_depth, v)
      Let end = ComputeEndDate(P'', start, ForbiddenNodes(pm.depth), v)
      If (StopBefore(pm.depth))
      // can stop at an intermediate date
         For (j=i; j<=min(end,n-m+depth); j++)
            let pm = CreateExtendedMatch (pm, j, v)
            push pm in L
         End for
      Else If (end ≤ n-m+depth)
         // end date is not too late
         let pm = CreateExtendedMatch (pm, end, v)
         push pm in L
      Endif
Endif
End While
```

(a) Algorithme principal pour étendre un match

```
Boolean CHECKNEWNODES (int i, int start,
                       int end, Valuation v)
   For each f in ForbiddenNodes(i)
      Let concreteId be v(f)
      // Is concreteId new?
      For (j=start, j≤end, j++)
        If (concreteId exists in C_j)
           Return (false)
        End if
      End for
   End For
   Return (true)
End CHECKNEWNODES
```

```
ListOfHomomorphisms
NEWMATCHGRAPHS
(Graph G1, Graph
G2,SetOfStrings ForbidIds,
Valuation v)
   Same as previously
   See Figure 40.b
End NEWMATCHGRAPHS


int COMPUTEENDDATE (Graph G,
int start, SetOfStrings
ForbidIds, Valuation v)
   Same as previously
   See Figure 40.b
End COMPUTEENDDATE

PartialMatch
CREATEEXTENDEDMATCH ()
   Same as previously
   See Figure 36.b

End CREATEEXTENDEDMATCH
```

(b) Fonctions auxiliaires

Figure 41. Modification pour l'extension d'un match partiel *pm*

Pour chaque extension candidate de match partiel, la date de fin est calculée et vérifiée. La vérification utilise une version valuée *P''* du motif, après fusion des valuations passées et nouvelles. Notons que les appels à la fonction *COMPUTEENDDATE()* prennent en compte les nœuds interdits. Enfin, le traitement de la date de fin dépend de la valeur booléenne *StopBefore*, comme cela a déjà été expliqué pour les matchs partiels de profondeur *1*. Lorsque la valeur booléenne est vraie, plusieurs dates de fin pourront être considérées, chacune donnant lieu à un match partiel.

Pour illustrer toutes ces notions, nous allons reprendre l'exemple et étendre les matchs partiels *pm1*, *pm2*, *pm3* selon l'algorithme.

Commençons par expliquer l'extension de *pm2*, qui nous permet d'illustrer la possibilité d'avoir plusieurs dates de fin candidates. Lors de l'extension à la profondeur *2*, on regarde si les configurations concrètes à partir de la date *1* contiennent le motif P_1 sans contenir le nœud concret interdit *N3*. La fonction *COMPUTEENDDATE()* retourne la date *2*. Comme le booléen *StopBefore* est vrai à cette étape, il y a en fait deux dates de fin possibles *1* et *2*.

pm2' :

val: { (id1, "N1"), (x, 2), (id2, "N3") }

index:	0	0	1	-1

depth 2

pm2" :

val: { (id1, "N1"), (x, 2), (id2, "N3") }

index:	0	0	2	-1

depth 2

Pour chacun de ces matchs partiels, la suite de la recherche va échouer. Lors de l'extension de *pm2'*, on ne trouve pas d'instance du motif P_2 dans C_2. Lors de l'extension de *pm2''*, on voit que C_3 contient le nœud interdit *N3*, de sorte que l'appel à NEWMATCHGRAPHS() échoue là encore. Il n'y a donc pas d'extension de *pm2* qui satisfasse les propriétés requises.

Les extensions de *pm1* et *pm3* sont plus fructueuses. Pour ces extensions, le nœud concret interdit est *N2*. Une instance de P_1 est trouvée jusqu'en C_3, mais *3* est une date de fin trop tardive car il reste encore le motif P_2 à traiter. Les dates de fin possibles sont entre 2 et min(3,2), c'est-à-dire qu'il y a en fait une seule possibilité.

Pour *pm1*, on crée donc l'extension suivante :

val: { (id1, "N1"), (x, 2), (id2, "N2") }

index:	0	1	2	-1

depth 2

qui pourra être encore étendue jusqu'à la profondeur 3.

pm1' :

val: { (id1, "N1"), (x, 2), (id2, "N2"), (id3, "N4") }

index:	0	1	2	3

depth 3

Notons qu'ici, l'algorithme considère aussi une extension alternative avec la valuation partielle *{(id3, "N3")}*, mais celle-ci est rejetée par la vérification CHECKNEWNODES().

De façon similaire, les extensions de *pm3* donnent lieu à deux matchs de profondeur 3.

pm3' :

val: { (id1, "N1"), (x, 2), (id2, "N2"), (id3, "N3") }

index:	1	1	2	3

depth 3

(Comme la date de début pour P_0 est *1*, *N3* est ici considéré comme un nouveau nœud.)

pm3" :

val: { (id1, "N1"), (x, 2), (id2, "N2"), (id3, "N4") }

index:	1	1	2	3

depth 3

En résumé, le résultat des extensions successives jusqu'à la profondeur complète présente les caractéristiques suivantes :

- La propriété 1 est garantie par les appels à la fonction de recherche d'homomorphisme.

- La propriété 2 est vérifiée graduellement lors de la recherche, par les appels à NEWMATCHGRAPHS (pour les nœuds interdits déjà valués) et à CHECKNEWNODES (pour les nœuds interdits nouvellement valués).
- La date de fin de chaque motif (et donc aussi la date de début du motif suivant) est maximale au sens de la propriété 3. Lorsque *StopBefore* est faux, la date de fin est telle que le motif n'est plus contenu dans la configuration concrète suivante, ou qu'un nœud interdit apparaît dans cette configuration concrète. Lorsque *StopBefore* est vrai, les dates de fin retenues avec succès correspondent à l'apparition d'un nouveau nœud, non présent dans les configurations passées.

Il reste juste à vérifier que la date de début pour P_0 est maximale, ce qui fait l'objet de la vérification finale.

5.2.4.1. Vérification finale

La vérification finale est présentée dans la Figure 42. Rappelons que dans le cas où l'ensemble *ForbiddenNodes(0)* n'est pas vide, nous avons retenu des dates de début potentielles *i* même si C_{i-1} contenait déjà l'instance trouvée de P_0. Nous devons maintenant décider si oui ou non la valeur *i* est réellement une date de début, au sens de la propriété 3. La décision est triviale pour *i=0*. Pour *i>0*, la décision sera négative si C_{i-1} peut être apparié avec P_0 et ne contient aucun nœud interdit.

```
Boolean FINALCHECK (PartialMatch pm)
   If (pm.index[0]=0 || ForbiddenNodes(0) = ∅)
      return (true)
   Else
      Let P' = VALUATEGRAPH (P0, pm.val)
      Let H= NEWMATCHGRAPHS (P', Cpm.index[0]-1, ForbiddenNodes(0), pm.val)
      If (H is empty)
         Return (true)
      Else return (false)
      Endif
   Endif
End FINALCHECK
```

Figure 42. Vérification finale avant de produire les résultats

Revenons à l'exemple de la Figure 38. Le match *pm1'* est trivialement accepté. Pour *pm3'* et *pm3''*, la vérification acceptera le premier :

val:	{ (id1, "N1"), (x, 2), (id2, "N2"), (id3, "N3") }			
index:	1	1	2	3

mais pas le second avec *id3 := "N4"*, car la date de début pourrait être plus tôt, en *0* (ce qui donnerait d'ailleurs *pm1'*).

On aura donc au final deux matchs complets, issus de *pm1'* et *pm3'*.

6. Conclusion

Ce chapitre a présenté les algorithmes d'appariement de graphes de *GraphSeq*, un outil développé pour traiter la vue spatiale des scénarios d'interaction dans le cadre des systèmes mobiles. La vue spatiale représente explicitement la dynamicité de la topologie des nœuds dans le système, avec des connexions et déconnexions induites par la mobilité, ainsi que des créations et des disparitions dynamiques des nœuds.

GraphSeq prend en entrée deux séquences de graphes étiquetés. Ces séquences proviennent, respectivement, d'un scénario et d'une trace. *GraphSeq* génère l'ensemble de tous les matchs. L'algorithme implémenté dans *GraphSeq* utilise des fonctionnalités déjà existantes pour la recherche d'homomorphismes de graphes, développées par des collègues du LAAS-CNRS. La nouveauté de notre outil consiste à raisonner sur des séquences de configurations spatiales. Le problème principal est de retenir des valuations cohérentes tout au long d'une séquence de graphes. En particulier, la prise en compte des scénarios où des nœuds apparaissent et disparaissent dynamiquement s'est avérée délicate, car il faut tenir compte du fait que certains nœuds sont interdits dans certaines configurations.

L'étude de la complexité de la recherche d'appariements, ainsi que la validation et l'application de *GraphSeq* à des exemples expérimentaux seront présentées dans le chapitre suivant.

CHAPITRE 5

GRAPHSEQ : COMPLEXITE, VALIDATION ET EXPERIMENTATION

1. Introduction

Dans le chapitre précédent, nous avons présenté l'algorithme détaillé de *GraphSeq*. Nous avons noté que *GraphSeq* pouvait être utilisé avec les langages de scénario qui incluent la vue spatiale et ont un événement de changement de configuration globale dans la vue événementielle. Dans ce chapitre seront présentées l'étude de complexité, la validation et l'application de notre outil *GraphSeq*.

Dans un premier temps, nous allons aborder l'étude de complexité de l'algorithme de *GraphSeq*, en se basant sur la complexité de l'algorithme d'appariement de graphes qui a été présenté dans [Guennoun 06]. *GraphSeq* sera analysé dans le meilleur et le pire cas, qui correspondent aux cas de calcul minimal et maximal.

Ensuite, nous allons présenter la validation de *GraphSeq*. Nous commençons par des petits exemples qui illustrent les problèmes spécifiques de l'algorithme (les problèmes de création, de disparition, de réintégration et de la vérification finale). Après, *GraphSeq* est validé par des centaines de séquences de graphes qui ont été générées aléatoirement par un outil automatisé. Nous l'avons utilisé dans les deux cas : avec un nombre fixe de nœuds puis avec des créations/disparitions de nœuds.

Nous allons montrer l'utilité de *GraphSeq* pour extraire des motifs de mobilité au travers de résultats d'expérimentations. Il s'agit des analyses effectuées sur des sorties d'un simulateur de mobilité et des traces de test du cas d'étude GMP.

2. Analyse de complexité de *GraphSeq*

Cette section présente une analyse de complexité de l'outil *GraphSeq*. Nous étudions la complexité de l'algorithme pour le meilleur et le pire cas. Dans le meilleur cas, les entrées de l'algorithme se situent dans un contexte favorable impliquant un minimum de calcul. Dans le pire cas, *GraphSeq* réalise un nombre maximum d'opérations. Pour ces analyses, nous allons d'abord résumer la complexité de l'algorithme de recherche d'homomorphismes de graphes,

car nous l'utilisons comme une fonctionnalité de base pour notre outil. Ensuite, nous allons présenter la complexité introduite par la déduction sur des séquences de graphes.

2.1. Analyse de complexité de l'algorithme de recherche d'homomorphismes

Guennoun a présenté l'analyse de son algorithme de recherche d'homomorphisme dans [Guennoun 06]. Le résultat de cette analyse est résumé dans les paragraphes 2.1.1 et 2.1.2. Les paramètres pris en compte sont :

- N_{Pi} : le nombre de nœuds dans le graphe motif P_i
- N_{Cj} : le nombre de nœuds dans le graphe de configuration concrète C_j

Notons que nous ne prenons pas en compte le nombre d'étiquettes d'un nœud comme un paramètre car, dans *GraphSeq*, il est constant (sa valeur est de 3).

2.1.1. Le meilleur cas

Le meilleur cas correspond à la situation où les étiquettes sont constantes : chaque nœud du graphe motif est étiqueté par une liste unique et correspond à un et un seul nœud dans le graphe de configuration. Dans ce cas, la complexité de l'algorithme est égale à :

$$C1 = O(N_{Pi}.N_{Cj} + N_{Pi}{}^{2})$$

Nous constatons que la complexité, dans le meilleur cas, est polynomiale avec le nombre de nœuds du graphe motif et de la configuration.

2.1.2. Le pire cas

Le pire cas correspond à la situation où : les étiquettes sont variables (autant de variables différentes que d'étiquettes pour chaque nœud), tous les nœuds du graphe motif et ceux du graphe de configuration sont unifiables deux à deux, tous les graphes sont complets, et tous leurs arcs sont étiquetées par la même étiquette. Dans ce cas, la complexité de l'algorithme est égale à :

$$C2 = O(N_{Cj}{}^{N_{Pi}})$$

Nous voyons que la complexité, dans le pire cas, est exponentielle avec le nombre de nœuds du graphe motif.

Nous allons appliquer les résultats ci-dessus pour *GraphSeq*. Par raison de simplicité, nous appelons *H* la complexité de recherche d'homomorphismes. *H* varie donc entre la complexité pour le meilleur cas et celle pour le pire cas.

2.2. Analyse de complexité de *GraphSeq*

2.2.1. Complexité des fonctions auxiliaires

Nous allons maintenant présenter les analyses de complexité des fonctions auxiliaires des figures [Figure 35.b, Figure 36.b, Figure 39, Figure 40b, Figure 41].

- *ValuateGraph()* : Pour cette fonction, nous parcourons le graphe G pour valuer tous les sommets si la variable est présente dans la valuation. La complexité est

polynomiale avec le nombre de sommets du graphe et le cardinal de la valuation : $O(N_{Pi}.card(V))$ où V est la valuation passée en paramètre de la fonction.

- *ComputeEndDate()* : Nous effectuons la recherche d'homomorphismes entre le graphe motif et les graphes de configuration jusqu'à trouver une configuration où nous ne trouvons plus d'homomorphisme. La complexité de cette fonction dépend donc de la longueur d'appariements.
- *CreatePartialMatchD1()* et *CreateExtendedMatch()* : La complexité de cette fonction est linéaire avec le cardinal du match partiel et de la valuation.
- *NewMatchGraphs()* : Dans cette fonction, parallèlement à la recherche d'homomorphismes, nous devons vérifier les nœuds qui sont interdits. Cette vérification est polynomiale avec le nombre de nœuds dans le graphe motif et le cardinal de la valuation. Le calcul de la complexité de cette fonction résulte de la combinaison entre la complexité de la recherche d'homomorphismes et celle de la vérification.
- L'extraction de graphes motif : Rappelons que, cette étape extrait, pour chaque motif P_j, l'ensemble des identifiants symboliques de nœuds qui étiquettent les sommets. Ensuite, elle calcule les informations des nouveaux nœuds et des nœuds interdits pour être stockées dans une structure de données globale, *NewNodes* et *ForbiddenNodes*. Elle est linéaire avec le nombre de nœuds des graphes motif.
- La vérification d'existence de transition *StopBefore()* : Cette fonction a l'objectif de vérifier si une transition de P_j à P_{j+1} peut survenir. Cette fonction compare pour cela les sommets et les arêtes entre deux graphes motif. La complexité est polynomiale avec le nombre de nœuds de ces deux graphes : $O(N_{Pj}.N_{Pj+1})$.
- *CheckNewNodes()* : Cette fonction prend en compte de nouveaux nœuds qui ne doivent pas être présents dans des graphes de configuration. Ce calcul est polynomial avec le cardinal de l'ensemble *NewNodes()* et le nombre de nœuds des graphes de configuration : $O(N_{Cj}.card(NewNodes()))$

2.2.2. La complexité de l'algorithme de GraphSeq

Nous allons considérer la complexité de *GraphSeq* pour le meilleur et le pire cas. Etant donné que *GraphSeq* travaille sur des séquences de graphes, nous devons prendre en compte deux autres paramètres suivants :

- m : la longueur des motifs
- n : la longueur des configurations

2.2.2.1. Pour le meilleur cas

Le meilleur cas correspond à la situation suivante :

- Nous avons un nombre fixe de nœuds, c'est-à-dire qu'il n'y a ni création, ni disparition de nœud.
- Chaque graphe motif P_i est apparié avec un seul graphe de configuration C_i, et la date de début de match est 0,

- Seule l'étiquette qui est utilisée pour l'identifiant de nœud est variable, les autres étiquettes de sommet sont constantes,
- Pour chaque paire d'appariement entre P_i et C_i, chaque nœud du motif correspond à un seul nœud dans la configuration. Ça veut dire que nous avons trouvé un homomorphisme entre deux graphes et chaque match partiel contient un élément.

Dans le meilleur cas, la pile qui stocke les matchs partiels contient toujours un élément. De plus, les identifiants de nœuds sont valués après le match de P_0.

Le coût total est donc dominé par le coût de comparaison du motif P_0 à chaque configuration C_s. Par conséquent, la complexité est :

$$O(nH)$$

2.2.2.2. Pour le pire cas

Le pire cas correspond à la situation suivante :

- Tous les nœuds diffèrent d'un motif à l'autre, ça veut dire aussi que *NewNodes(i)* = N_{Pi},
- Toutes les variables (dans tous les nœuds) sont différentes,
- Le nombre d'homomorphismes trouvés entre un motif et une configuration est maximal. C'est le cas où, pour chaque paire d'appariement entre deux graphes, tous les nœuds dans le motif sont unifiables avec tous les nœuds dans la configuration et toutes les arêtes ont une même valeur,

Nous pouvons constater que, pour le pire cas, la complexité des fonctions auxiliaires et de l'étape de prétraitement est négligeable par rapport à la complexité de la recherche d'homomorphismes. Par conséquent, pour le pire cas, nous ne prenons en compte que la complexité de la recherche d'homomorphismes. Ensuite, le nombre de matchs partiels dans la pile est toujours maximal et exponentiel avec le nombre de motifs traités.

Quand nous cherchons à apparier le motif P_{i+1}, nous devons chercher une valuation pour les identifiants de nœuds qui sont nouveaux par rapport aux motifs précédents P_0, …, P_i. Cela nous amène à plusieurs suites possibles au match partiel en cours. Avec les hypothèses ci-dessus, les ensembles de nœuds des différents motifs sont disjoints, nous avons une complexité :

$$O(nH^m)$$

Quand la transition $P_i \rightarrow P_{i+1}$ n'est provoquée que par la création de nouveaux nœuds, il y a plusieurs dates possibles de fin pour le match de P_i, chacune donnant lieu à une recherche d'homomorphismes pour traiter les nouveaux nœuds. Une estimation pessimiste du pire cas est alors de supposer que toutes les transitions sont activées par l'apparition de nœuds, ce qui donne :

$$O(n^m H^m)$$

L'impact de la création de nœuds est donc une augmentation polynomiale du temps de calcul en *n*, et une augmentation exponentielle en *m*.

2.3. Discussion

Le résultat de complexité de *GraphSeq* montre que, l'application de *GraphSeq* pour l'analyse des traces de mobilité peut être limitée. En effet, la complexité exponentielle pour le pire cas ne nous permet pas d'analyser les systèmes à grande échelle. C'est la raison pour laquelle nous avons choisi d'effectuer nos expérimentations avec des systèmes de quelques dizaines de nœuds, qui seront présentées par la suite. Cependant, nous constatons que *GraphSeq* est utilisable vis-à-vis de l'objectif pour lequel il a été conçu : l'analyse des traces de test à partir de scénarios graphiques.

Les descriptions de scénario impliquent généralement que quelques nœuds et on se concentre sur leurs interactions. Ainsi, il est attendu que le nombre de motifs successifs dans les scénarios reste petit (par exemple, deux ou trois motifs dans un scénario de GMP).

La taille et le nombre de graphes de configuration sont moins critiques par rapport à l'analyse de complexité. Nous trouvons, dans la littérature, des travaux d'évaluation de protocoles de routage qui ne concernent pas un nombre de nœuds et une longueur d'expérimentation importants. Par exemple, dans [Johnson et al. 01], [Vic et Medid 04], [Johnson et Maltz 96], [Devarapalli et al. 01] et [Johnson 05] entre 5-50 nœuds sont impliqués et une durée de traitement comprise entre 28 secondes et 5 minutes. Nos expérimentations montrent que de telles configurations sont à la portée de *GraphSeq*.

3. Validation de *GraphSeq*

Dans cette section, nous allons présenter la validation de notre outil *GraphSeq*. Pour cela, nous avons traité plusieurs exemples pour pouvoir, au préalable, analyser les problèmes de création et de disparition de nœuds. Ensuite, *GraphSeq* a été validé par des centaines de séquences de graphes qui ont été générées aléatoirement.

3.1. Analyse des exemples illustratifs

Dans certaines situations, de petits exemples s'avèrent utiles pour mettre en évidence des problèmes spécifiques à un algorithme. C'est dans cet esprit que nous avons traité et analysé des exemples pour se rendre compte des problèmes de création, de disparition, de réapparition et de vérification finale de l'algorithme de *GraphSeq*. Ils sont présentés par la suite.

Le problème de la création de nœuds

Considérons la Figure 43. Supposons que le motif P_0 s'apparie avec C_0 et C_1, ce qui produit le match partiel suivant :

val:	{(id1,N1)}		
Index:	0	1	-1

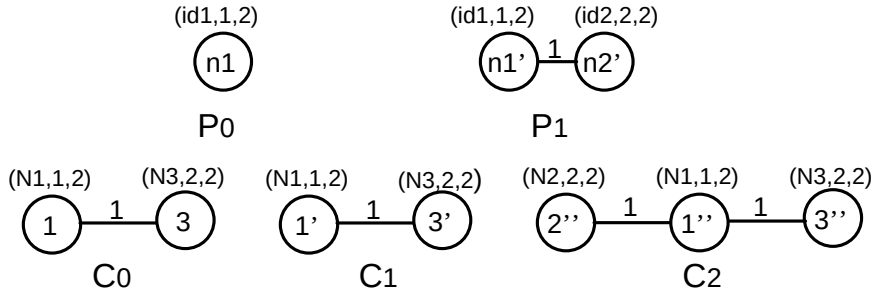


Figure 43. Exemple de vérification de problème de création de nœuds

Nous continuons à apparier le motif P_1 avec C_2. Le nœud symbolique *id2* peut correspondre au nœud *N2* ou *N3* dans C_2. Cependant, *id2* ne peut pas être *N3*, car il est un nouveau nœud dans la séquence des motifs et le nœud *N3* est déjà dans les configurations C_0 et C_1. Donc, nous avons le match complet suivant :

val:	{(id1,N1), (id2,N2)}		
Index:	0	1	2

Le problème de la disparition de nœuds

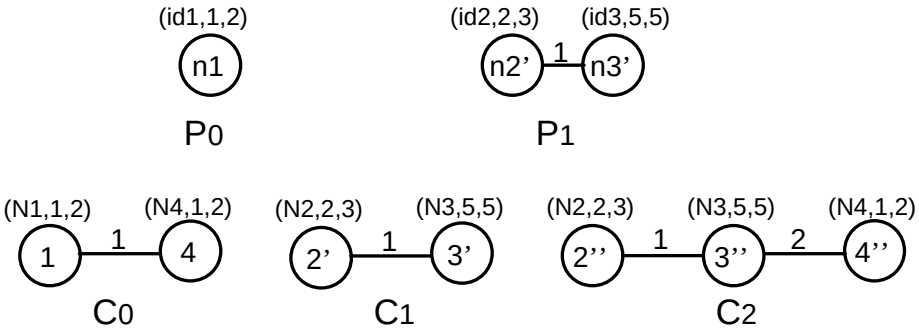


Figure 44. Exemple de vérification de problème de disparition de nœuds

Considérons la Figure 44. Dans cet exemple, nous n'acceptons pas le match suivant :

val:	{(id1, N4), (id2, N2), (id3, N3)}		
Index:	0	0	2

Comme le nœud symbolique *id1* disparaît dans le graphe motif P_1, le nœud correspondant (le nœud concret *N4*) dans les configurations ne doit pas être présent dans les graphes C_1 et C_2. C'est pour cette raison que nous ne pouvons pas retenir le match du tableau ci-dessus.

La transition de match partiel quand il y a un nouveau nœud

Considérons la Figure 45. Dans cet exemple, le graphe motif P_1 introduit un nouveau nœud symbolique *id2* par rapport à P_0. Ensuite, nous voyons que le motif P_0 peut être apparié de C_0 à C_2. Par conséquent, la transition de P_0 à P_1 peut être détecté en C_1 ou C_2, où apparaissent de nouveaux nœuds concrets (*N2* ou *N3*).

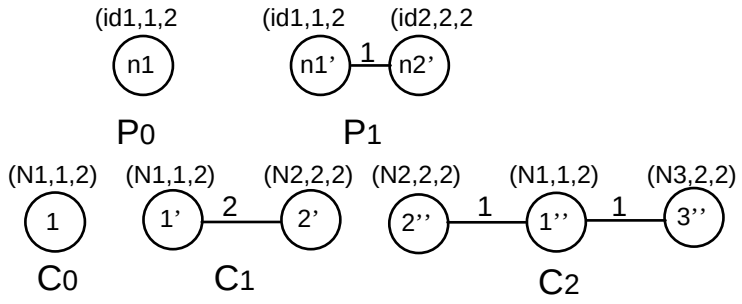


Figure 45. Exemple de vérification de transition de motifs

La vérification finale

D'après l'algorithme, dans le cas où l'ensemble *ForbiddenNodes(0)* n'est pas vide, nous retenons les dates de début « potentielles » *i* même si C_{i-1} a déjà été apparié avec P_0. Ensuite, nous devons décider si oui ou non la valeur *i* est une date de début « réelle ». La décision sera négative si C_{i-1} est apparié avec P_0 et ne contient aucun nœud interdit. Un exemple simple est celui de la Figure 46 où P_0 est apparié avec C_1 et P_1 avec C_2 car si P_0 avait été apparié dès C_0 alors N2 n'aurait pas été nouveau en C_2 car déjà là en C_0 et l'appariement P_1 avec C_2 aurait été impossible.

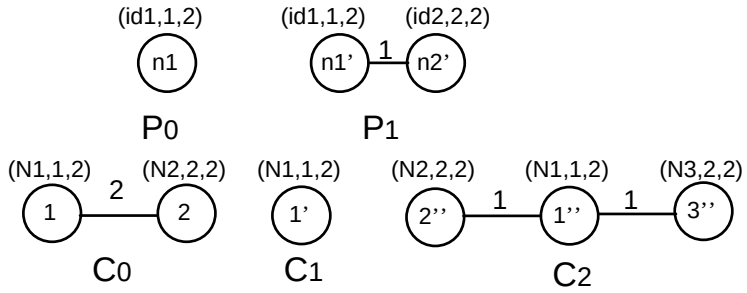


Figure 46. Exemple de vérification finale

Avec la vérification finale, nous allons accepter le match suivant :

val: {(id1, N1), (id2, N2)}

Index:	1	1	2

Notons que les problèmes ci-dessus peuvent être combinés pour provoquer des problèmes plus complexes. La Figure 47 montre un exemple de mélange de problèmes :

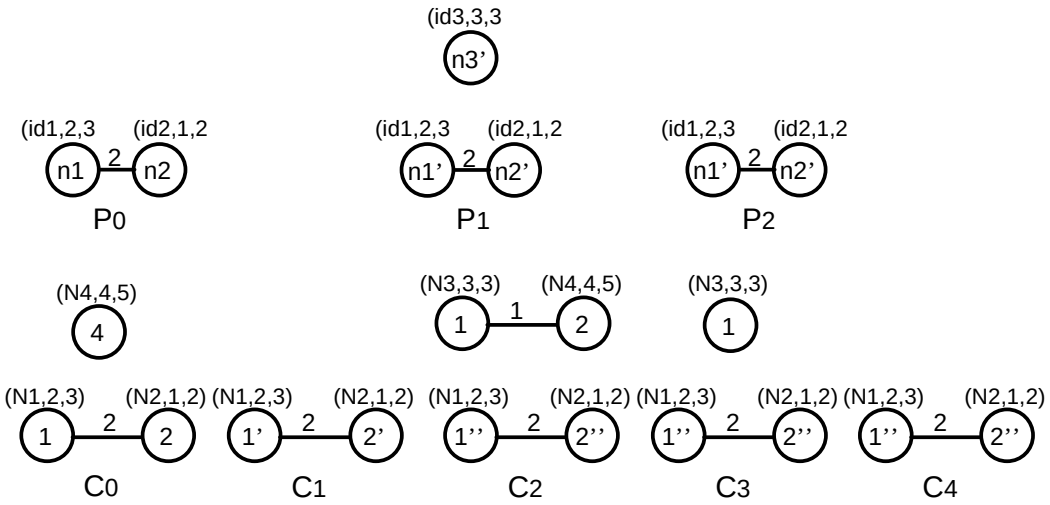


Figure 47. Exemple de vérification de problèmes mélangés

Dans cet exemple, le nœud symbolique *id3* ne peut pas être retenu avec la valeur *N4* (en C_2), car il est le nouveau nœud dans le monde de motif et le nœud *N4* est déjà présent en C_0. De plus, l'apparition de *n3* peut conduire la transition (de P_0 à P_1) à C_2 ou C_3. Ainsi, la terminaison du match peut être en C_3 ou C_4. De telles combinaisons peuvent provoquer une explosion combinatoire à l'exécution de l'algorithme.

3.2. Validation par un outil

Nous avons commencé l'étape de validation avec de petits exemples qui ont été produits manuellement (cf. section 3.1) mais nous nous sommes rapidement rendus compte que nous aurions besoin d'une solution automatisée pour à la fois générer les graphes et analyser les résultats. C'est dans ce sens que nous avons développé un outil qui génère des séquences aléatoires de graphes P_i et C_j tels que, par construction, un graphe C_j contienne au moins un match. Les résultats de *GraphSeq* peuvent alors être analysés automatiquement et un verdict de défaillance est déclenché si le match attendu n'a pas été trouvé par l'outil. Notons que l'outil *GraphSeq* peut trouver d'autres matchs mais le verdict du test ne concerne que celui qui est connu par construction.

La génération aléatoire des graphes dans l'outil peut être paramétrée. Les paramètres et ses liens peuvent être résumés comme suite : la longueur maximale d'une séquence de motifs, le nombre de nœuds maximal pour chaque motif, le nombre de nœuds maximal pour une configuration (supérieur au nombre de nœuds du motif), le nombre d'étiquettes dans les motifs, la différence entre deux motifs successifs, la durée de match entre les motifs et les configurations.

Nous avons produit environ 900 cas de test qui ont exhibé des caractéristiques différentes. Ces cas de test ont été appliqués pour les deux versions de l'algorithme : le cas avec un nombre fixe de nœuds et le cas avec des créations/disparitions de nœuds.

- Le nombre de motifs pour chaque séquence varie de 1 à 5,
- Le nombre de configurations concrètes varie de 1 à 100,

- Le nombre de nœuds dans les motifs varie de 1 à 5,
- Le nombre de nœuds dans les configurations concrètes varie de 1 à 25,
- Pour chaque motif individuel P_i, la durée d'un match du motif dans les configurations concrètes varie de 1 à 20,
- Les transitions de P_i à P_{i+1} peuvent entraîner un changement au niveau d'une étiquette d'un nœud, d'une arête, d'un nœud qui apparaît ou disparaît, ou encore n'importe quelle combinaison (le nombre maximal de changements étant de 5).

L'outil de test a prouvé son utilité pour déboguer *GraphSeq* et pour effectuer des vérifications de non régression chaque fois que des changements ont été apportés dans le code source en C++.

4. Application de *GraphSeq*

Nous allons présenter, dans cette section, l'utilité de *GraphSeq* pour extraire des motifs de mobilité au travers de résultats d'expérimentations. Il s'agit d'analyses effectuées sur des sorties d'un simulateur de mobilité et des traces de test du cas d'étude du protocole d'appartenance de groupe (GMP).

4.1. Connexion à un simulateur de mobilité

4.1.1. Principe

Les simulateurs de mobilité gèrent les positions relatives des nœuds selon des modèles de mobilité et produisent des données contextuelles (par exemple, les données basées sur la localisation). Ce sont des informations requises par une application dans le cadre de la mobilité physique. Des exemples de plates-formes de test avec un simulateur de mobilité ont été présentés dans le chapitre 1. Les données produites à chaque étape de simulation peuvent être utilisées afin d'identifier les configurations du système testé. Pour démontrer le concept, nous avons fait l'expérimentation en connectant *GraphSeq* avec un outil de simulation.

Nous avons choisi un simulateur de mobilité développé à l'Université de Californie du Sud (USC), aux États-Unis. Ce simulateur a été développé dans le cadre du projet IMPORTANT (*Impact of Mobility Motifs On Routing proTocol in the mobile Ad hoc NeTworks*). Cet outil est distribué sous licence libre[1]. Il offre un ensemble riche de modèles de mobilité paramétrables, avec entre autres les plus connus comme *Reference Point Group*, *Freeway* et *Manhattan*. Les traces générées par cet outil sont compatibles avec le simulateur de réseau ns-2.

4.1.2. Modèle des expérimentations

La connexion de *GraphSeq* à un tel simulateur de mobilité exige le développement des composants d'interface (voir la Figure 48) :

- un traducteur de format,

[1] http://nile.usc.edu/important/

- un outil d'optimisation de graphes de configuration.

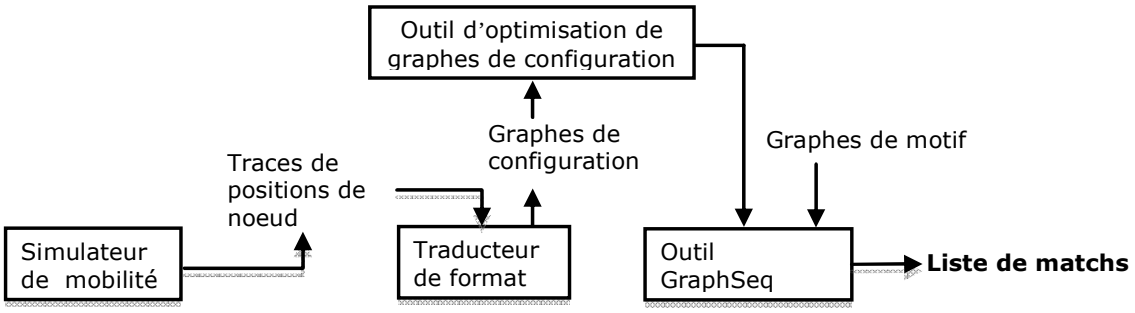


Figure 48. Connexion du *GraphSeq* à un simulateur de mobilité

Le traducteur de format abstrait les données brutes de la simulation en une séquence de graphes de configuration. Dans nos expérimentations, les données brutes sont des positions de nœuds à chaque étape de simulation. Ensuite, l'abstraction consiste à définir des étiquettes pour les arêtes en fonction de la distance entre les nœuds (reliés ou non). Par exemple, dans le cas d'étude du GMP, la distance entre les nœuds peut être définie par les relations *Safe* (dans la distance de sécurité), *RangeNotSafe* (dans la portée de transmission mais pas dans la distance de sécurité) ou déconnecté.

L'outil d'optimisation de graphes de configuration a pour but de réduire le nombre de graphes de configuration. Nous avons constaté que les graphes de configuration doivent parfois rester stables. Par exemple, considérons la situation suivante : la distance actuelle entre deux nœuds est 15m, leur vitesse maximale est 10m/s, les arêtes entre deux nœuds quelconques changent de 1 à 2 si la distance *d* correspondante passe de d ≤ 100m à d >100m, et les traces sont enregistrés toutes les secondes. L'étiquette de l'arête reliant ces deux nœuds ne change pas pendant au moins 4 secondes (ou 4 étapes). A partir de ce principe, le fonctionnement de l'outil d'optimisation de graphes de configuration est le suivant : si des graphes successifs sont identiques, l'information retenue sera un graphe représentatif et sa longueur. Cela veut dire que nous allons stocker les configurations seulement si elles changent de structure par rapport à la configuration précédente.

Le bloc d'optimisation a prouvé son utilité lors de la réduction du nombre de graphes de configuration à traiter. Nous avons mené quelques expériences pour estimer le gain avec l'optimisation :

$$Gain = \frac{Nombre\ Réduit\ De\ Graphes}{Nombre\ Brut\ De\ Graphes}$$

Les expérimentations ont été effectuées avec différents modèles existant du simulateur. Nous avons fait varié certains paramètres, comme le nombre de véhicules, la vitesse, le modèle de mobilité. Nous avons adopté la définition suivante pour les valeurs des arêtes de graphes :

- $0 < d < 140$m : arête = 1,
- $140 \leq d \leq 300$m : arête = 2,
- $d > 300$m : les nœuds sont déconnectés.

  (avec *d* la distance entre deux nœuds mobiles)

Nous avons conduit un nombre important d'expérimentations. Le gain est calculé à partir de la moyenne des gains de chaque expérimentation. Des exemples de résultats sont fournis dans les Tableau 5 et Tableau 6.

Tableau 5. Expérimentation avec le modèle Random Waypoint

Nombre de nœuds	**Gain** (avec vitesse de $10m.s^{-1}$)	**Gain** (avec vitesse de $5m.s^{-1}$)
6	0,245	0,102
8	0,399	0,180
10	0,543	0,244
15	0,790	0,438
20	0,875	0,595
25	0,901	0,674

Tableau 6. Expérimentation avec le modèle FreeWay

Nombre de véhicules	**Gain**
6	0,095
8	0,179
10	0,276
15	0,425

Revenons à l'application *GraphSeq*. Nous avons effectué une série d'essais avec divers modèles de mobilité, des paramétrages variés des modèles et différents motifs à rechercher. Nous allons décrire deux exemples d'exécution : l'un utilise le modèle *Freeway* et l'autre le modèle *Reference Point Group*.

4.1.3. Expérimentation avec le modèle Freeway

Le modèle *Freeway* émule le comportement du mouvement des véhicules sur les autoroutes. Il possède les caractéristiques suivantes :

- Chaque véhicule (ou chaque nœud mobile) est cantonné à sa voie sur les autoroutes.
- La vitesse de chaque nœud dépend temporellement de sa vitesse précédente.
- Si deux noeuds sur une même voie d'autoroute sont à une distance inférieure ou égale à *SD* l'un de l'autre, alors la vitesse du noeud qui suit ne peut dépasser celle du premier.

Le Tableau 7 donne les paramètres utilisés pour un exemple d'exécution. La carte a été construite en utilisant des fragments prédéfinis des autoroutes et des voies disponibles dans l'environnement de simulation.

Tableau 7. Paramètres pour l'exécution de la simulation (modèle Freeway)

Paramètres pour le modèle de mobilité	**Valeurs**
Durée de Simulation (1 pas de simulation / seconde)	15 minutes

Nombre de nœuds	15
Accélération	4 $m.s^{-2}$
Vitesse maximale	40 $m.s^{-1}$
Carte: Nombre d'autoroutes Nombre de voies	 2 6
Distance *SD*	40 m

Nous avons extrait 345 graphes différents de configurations à partir de la trace de simulation. Les identifiants concrets des nœuds mobiles sont étiquetés 'N0', 'N1', 'N2',… Les arêtes représentant les relations spatiales entre deux nœuds mobiles dépendent de la distance *d* qui les sépare. Ils sont définis comme suit :

- $0 < d < 140$m: arête = 1,
- $140 \leq d \leq 300$m: arête = 2,
- $d > 300$m: les nœuds sont déconnectés (ce qui signifie que 300 m est la portée maximale de transmission).

Les graphes motifs sont représentés Figure 49, avec *Safe* = 1 et *RangeNotSafe* = 2.

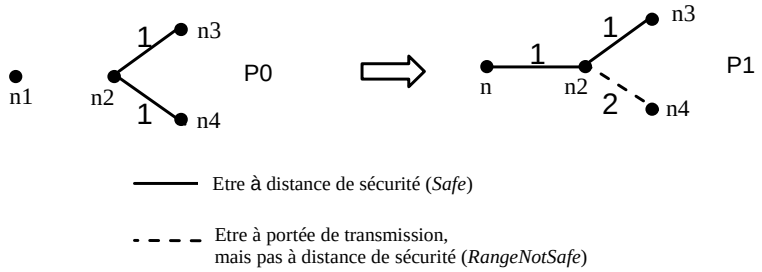


Figure 49. Les motifs utilisés dans l'expérimentation de modèle Freeway

GraphSeq cherche la séquence de motifs dans 345 configurations concrètes, et retourne 11 matchs dont un est illustré ici :

Val {(n1,"N0"),(n2,"N13"),(n3,"N11"),(n4,"N6")}

Index	76	83	85

4.1.4. Expérimentation avec le modèle RPGM

Le modèle *Reference Point Group (RPGM)* émule le mouvement d'un groupe avec les caractéristiques suivantes :

- Chaque groupe a un centre logique (un leader de groupe) qui détermine le déplacement du groupe,
- Au début, chaque membre de groupe est réparti d'une manière uniforme dans le voisinage du leader,
- A chaque instant, tous les nœuds ont une vitesse et une direction qui sont dérivées aléatoirement du leader.

Le Tableau 8 donne les paramètres utilisés pour un exemple d'exécution avec ce modèle. La carte a été construite en utilisant des fragments prédéfinis et disponibles dans l'environnement de simulation.

Tableau 8. Paramètres pour l'exécution de la simulation (modèle RPGM)

Paramètres pour le modèle de mobilité	**Valeurs**
Durée de Simulation (1 pas de simulation / seconde)	15 minutes
Nombre de nœuds Nombre de groupes Nombre de nœuds par groupe	12 3 4
Déviation de la vitesse	0,4
Vitesse maximale	40 $m.s^{-1}$
Vitesse minimale	10 $m.s^{-1}$

A partir de la trace de simulation, 297 graphes différents de configurations ont été extraits. Les identifiants concrets des nœuds et leurs arêtes sont définis comme pour l'exemple précédent (Freeway).

Les graphes motifs sont définis d'une manière aléatoire comme illustré sur la Figure 50.

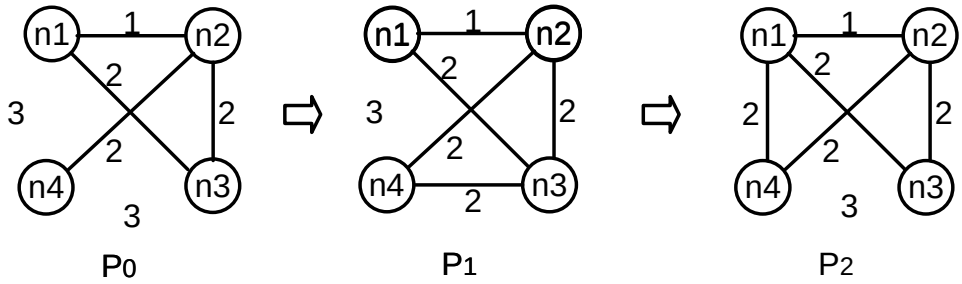


Figure 50. Motifs utilisés dans l'expérimentation du modèle RPGM

GraphSeq cherche la séquence de motifs dans les configurations concrètes, et retourne 7 matchs dont un est présenté ici :

Val {(n1,"N14"), (n2, "N8"), (n3, "N3"), (n4, "N5")}

Index	128	136	141	153

4.1.5. Discussion

L'extraction de tels matchs, comme nous l'avons montré dans le chapitre 3, peut être un support à l'implémentation des cas de test. En effet, la production manuelle des données contextuelles, qui instancient une évolution attendue des configurations, est fastidieuse voire irréaliste. La proposition est alors d'avoir une phase préliminaire de production de données contextuelles, basée sur des exécutions du simulateur de mobilité seul. La liste des matchs trouvés fournit un ensemble de données qui peut être rejoué pour une exécution d'un cas de test ciblé. Par exemple, le match ci-dessus est adéquat pour tester une application de type « véhicule à véhicule » dans le cas où les trajectoires des véhicules exhibent des motifs à chercher.

Comme nous l'avons expliqué, une fonctionnalité attendue de *GraphSeq* est d'identifier toutes les occurrences d'un scénario dans des traces de test. La partie d'appariement de graphes correspond au traitement de la vue spatiale. Elle est à compléter par une analyse des événements qui apparaissent dans la fenêtre temporelle du match pour les nœuds identifiés. Cette fonctionnalité sera mieux illustrée dans la section suivante, en utilisant le cas d'étude du GMP.

4.2. Application de *GraphSeq* au cas d'étude GMP

Le cas d'étude du GMP est présenté dans le chapitre 2. Rappelons que, nos expérimentations de test ont montré des violations diverses des propriétés de l'application (ces propriétés sont définies dans le Tableau 1). Il était impossible d'examiner manuellement tous les cas, mais un échantillon de 164 scénarios a été extrait pour une analyse détaillée. Le scénario de la Figure 15 est un des scénarios analysés.

Utilisons maintenant *GraphSeq* pour examiner une variante de scénario qui devrait produire exactement la même défaillance. Cette variante diffère dans la vue spatiale (la Figure 20). Le nœud *n1* n'existe pas dans la configuration initiale. Ensuite, il se réveille et est à distance de sécurité avec le nœud *n2*. En même temps, le nœud *n2* s'éloigne du nœud *n4*. Après l'événement de changement de configuration, les événements de communications sont comme dans la Figure 20.b.

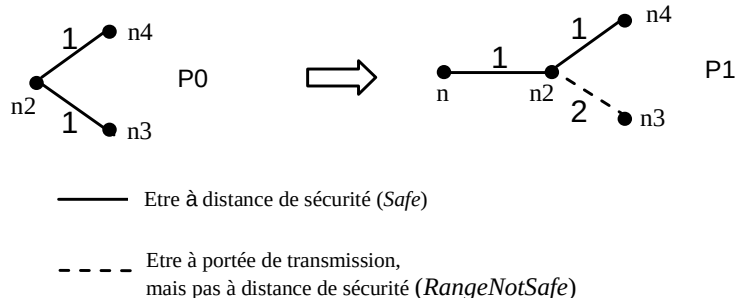


Figure 51. Vue spatiale de la variante du scénario

Les expérimentations incluent le test de l'implémentation du GMP, l'enregistrement des traces d'exécution et la recherche des occurrences de scénario pour déterminer si le mode de défaillance attendu est observé. Un nombre total de 100 exécutions a été généré avec les paramètres du Tableau 9.

Tableau 9. Paramètres pour les exécutions du GMP

Paramètres	Valeur
Nombre de nœuds	Aléatoire dans [6..15]
Date de début d'un nœud *StartDate*	Aléatoire dans [0..300] s
Date de fin d'un nœud *EndDate*	Aléatoire dans [StartDate..300] s
Portée de transmission	300 m
Distance de sécurité	140 m
Vitesse maximale des nœuds	10 $m.s^{-1}$
Modèle de mobilité	Mouvement aléatoire à la vitesse maximale
Temps d'une execution	5 minutes

GraphSeq a renvoyé 75 matchs entre la vue spatiale et les traces d'exécution. Il nous a permis d'extraire des sous-traces d'événements de communication qui se produisent durant les matchs. Notons que, l'analyse des événements de communication n'est pas encore automatisée. Nous avons donc dû comparer manuellement les événements dans le scénario et les sous-traces. Dans les 48 cas, les événements de communications n'ont pas matché. Le reste des 27 matchs correspondaient bien aux occurrences de scénario. Nous pouvons confirmer que le GMP exhibe la défaillance attendue pour chacun de ces cas.

Avec les mêmes traces, nous pouvons examiner le scénario où des fusions se font de manière concurrente. Rappelons la vue spatiale de ce scénario dans la Figure 52. Au début, les trois nœuds sont à portée de transmission de l'un et l'autre, mais pas à distance de sécurité. Ensuite, ils s'approchent et entrent dans la distance de sécurité.

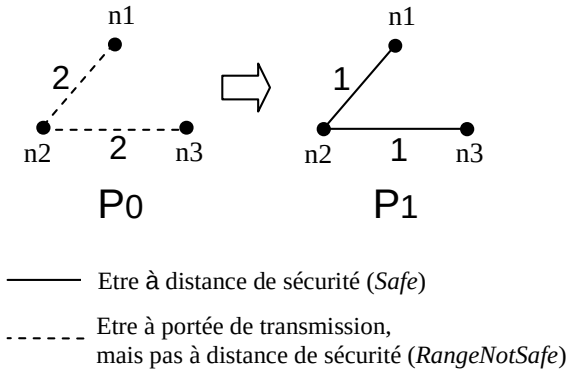


Figure 52. Vue spatiale du scénario de fusions concurrentes

GraphSeq a renvoyé 84 matchs entre la vue spatiale et les traces d'exécution. L'analyse manuelle a donné 31 cas de matchs des événements de communication.

Rappelons que les étapes d'analyse préliminaire du GMP (Chapitre 2, Section 3) ont montré des scénarios qui peuvent provoquer des violations. Nous n'avons pas pu les observer dans nos expérimentations de test à cause des limitations de la plate-forme utilisée. Nous constatons que ces limitations concernent l'ordre des événements de communication. De toute façon, nous pouvons toujours utiliser *GraphSeq* pour déterminer si la vue spatiale est présente dans les traces enregistrées. Si *GraphSeq* renvoie des matchs pour les motifs, l'utilisation de la plate-forme comme dans la Figure 16 pourra mettre en évidence des motifs de violations de communication en contrôlant le délai des messages.

Prenons l'exemple de l'ordonnancement des messages pour cette expérimentation. La vue spatiale est donnée dans la Figure 53. Au début, les trois nœuds *n1*, *n2*, *n3* sont dans un même groupe avec le leader *n3*. Ensuite, les nœuds *n1* et *n2* s'éloignent de leur leader. Enfin, le nœud *n1* s'éloigne le nœud *n2*.

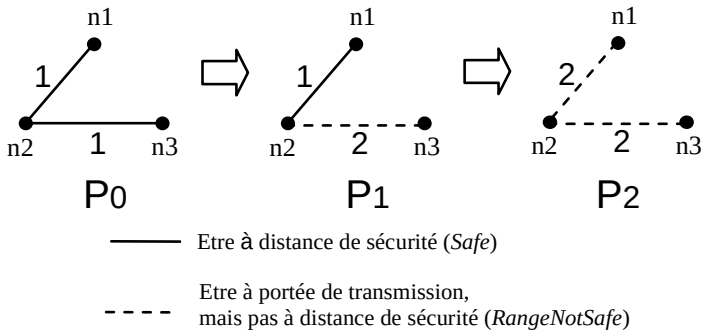


Figure 53. Vue spatiale du scénario d'ordonnancement de messages

Avec les mêmes traces précédentes, *GraphSeq* renvoie 114 matchs.

5. Conclusion

Dans ce chapitre, nous avons présenté l'étude de complexité, la validation et l'expérimentation de l'outil *GraphSeq*.

L'étude de complexité a montré que, pour le pire cas, les opérations de *GraphSeq* sont exponentielles avec le nombre de nœuds dans les motifs ainsi qu'avec leurs longueurs. Cela peut poser des problèmes de performance pour *GraphSeq*. L'utilisation de *GraphSeq* est, éventuellement, limitée à des dizaines de nœuds. Toutefois, cela reste comparable à des expérimentations sur des protocoles de routage ou des protocoles d'appartenance de groupe pour les systèmes mobiles.

Nous avons validé rigoureusement *GraphSeq* par des centaines de séquences de graphes qui ont été générées aléatoirement par un outil automatisé. Le nombre important des expérimentations garantit le bon fonctionnement de *GraphSeq*.

Nous avons également commencé des expérimentations afin de démontrer l'utilité de *GraphSeq*. Il s'agit d'expérimentations effectuées en connexion avec un simulateur de mobilité avec le cas d'étude (GMP) qui est représentatif du domaine de l'informatique mobile. Ces expérimentations ont montré l'application de *GraphSeq*. Il peut être utilisé pour rejouer une exécution d'un cas de test ciblé ou pour identifier toutes les occurrences d'un scénario dans des traces de test.

CONCLUSION

Les avancées technologiques du monde du « sans fil » ont conduit au développement d'applications dédiées à l'informatique mobile. Dans un contexte de mobilité physique, les caractéristiques propres de ces applications (dynamicité de la structure du système, communication avec des partenaires inconnus dans le voisinage, dépendance vis-à-vis du contexte) posent de nouveaux défis à leur vérification. Ce mémoire traite de la technologie de test. Nous allons résumer, ci-dessous, le corps des travaux et des contributions présentés dans le cadre de cette thèse.

Dans un premier temps, nous avons passé en revue l'état de l'art et étudié la littérature concernant les domaines de recherche dans lesquels s'inscrivent nos travaux : le test de systèmes mobiles. Nous avons poursuivi avec une étude de cas, un protocole d'appartenance de groupe dans le domaine des réseaux ad hoc, qui nous a permis de mieux comprendre les problèmes liés au test des applications mobiles. Ces étapes nous ont aidé à définir le noyau de nos travaux et nos contributions.

Nous avons présenté une approche de test qui se base sur les descriptions de scénarios. Afin d'appliquer cette approche dans les systèmes mobiles, il est nécessaire d'introduire de nouveaux éléments dans les langages de scénarios classiques pour couvrir les spécificités des systèmes mobiles. Nous avons proposé les trois extensions suivantes :

- Les relations spatiales entre nœuds doivent être considérées comme des concepts de première classe. Il incombe à l'utilisateur de déterminer quelles relations sont les plus adéquates pour représenter de façon abstraite la configuration spatiale de l'application ciblée. Nous utilisons les graphes étiquetés pour modéliser de telles relations et nous pouvons retenir le principe d'utiliser un formalisme visuel pour représenter les configurations spatiales successives.
- Les scénarios d'interactions sont décomposés en fragments où chaque fragment se déroule dans une des configurations spatiales précédemment définies. Les changements de configuration sont représentés par une synchronisation globale (ou de façon équivalente, nous avons une composition séquentielle forte des fragments). De cette manière, les dépendances causales entre les changements de configuration et les interactions entre nœuds sont rendues explicites. Il est également indiqué explicitement quel événement se produit dans quelle configuration.
- La communication par diffusion est introduite. Il y a un unique événement d'envoi suivi par un ensemble d'événements concurrents de réception. La notation doit également nous permettre de spécifier le sous-ensemble de nœuds récepteurs en faisant référence aux relations spatiales dans la configuration sous-jacente.

Ces trois extensions répondent à des besoins récurrents lorsque l'on cherche à modéliser des scénarios pour une application évoluant dans un contexte mobile. Un exemple

d'utilisation de ces extensions est le langage d'exigences TERMOS, qui a été développé dans le cadre du projet HIDENETS.

En effet, les descriptions de scénarios ne sont pas utilisées que pour la documentation. Il est attendu qu'elles seront compilées dans les programmes qui analysent automatiquement les traces d'exécution. Les traces d'exécution sont récoltées à partir de plates-formes génériques de test comportant les trois composants suivants : le support d'exécution de l'application, le simulateur du contexte et le simulateur du réseau.

Nous avons argumenté que les traitements liés aux descriptions engendrent un problème d'appariement de graphes, au moins pour certaines parties. Cela est dû au besoin de déterminer si un nœud physique apparaissant dans une trace peut être associé à des nœuds abstraits dans la vue spatiale. Pour cette raison, nous avons créé *GraphSeq* qui est un outil développé pour traiter l'aspect spatial dans les descriptions de scénarios. La nouveauté de *GraphSeq* se situe dans la déduction sur les séquences de graphes pour traiter les configurations spatiales, ce qui n'est pas encore exploité dans la littérature. L'application de *GraphSeq* pour les analyses des traces d'exécution dans les systèmes mobiles met aussi en évidence une contribution de nos travaux. L'outil peut être utilisé pour rejouer une exécution d'un cas de test ciblé ou pour identifier toutes les occurrences d'un scénario dans les traces de test. Nous avons montré son utilité pour extraire des motifs de mobilité au travers de résultats d'expérimentations.

Pour les perspectives, nous voyons des pistes suivantes.

Nos travaux pourront, d'abord, s'orienter vers le traitement intégré des vues spatiales et événementielles des scénarios. Des travaux coopératifs avec l'Université de Technologie et d'Economie de Budapest ont permis de définir un profil d'UML 2.0 pour les diagrammes de séquences spécifiques aux systèmes mobiles avec sa sémantique associée. Cela nous permettrait de compléter les analyses automatisées des traces de test avec les *matchs*, trouvés par *GraphSeq*, des événements de communication. La partie d'appariement de graphes correspond au traitement de la vue spatiale. Elle est à compléter par une analyse des événements qui apparaissent dans la fenêtre temporelle du match pour les nœuds identifiés.

Une autre piste est liée à la méthode de test. Comment sélectionner un (petit) ensemble de scénarios efficaces permettant de révéler les éventuelles fautes de conception de système ? Dans le cas des systèmes mobiles, une difficulté provient du nombre et de la nature des paramètres d'entrée à prendre en compte : les paramètres liés au contexte mobile sont aussi importants que les paramètres liés à la logique des applications. La combinaison efficace de ces deux types de paramètres, contextuels et applicatifs, reste une question ouverte.

Il serait aussi intéressant de s'attacher à résoudre les problèmes d'ordre technologique. Afin d'instancier les données de descriptions de scénarios et réaliser des cas de test, il est nécessaire de disposer de plates-formes expérimentales permettant l'émulation de l'environnement ainsi que le pilotage (commande et observation) des scénarios retenus. Comme nous l'avons abordé dans le premier chapitre, il existe des modèles qui ont été développés pour des objectifs d'évaluation. Nous pourrions développer une plate-forme, basée sur le modèle à trois composants, qui serait capable de s'adapter à tous types d'applications mobiles dépendantes du contexte.

Un autre problème à étudier, qui est plus d'ordre général toutefois, découle de l'utilisation d'une plate-forme de test basée sur la simuation ? Quelle confiance peut-on accorder à un outil de simulation dans le cadre de la vérification et la validation

d’applications ? En effet, un simulateur utilise des modèles mathématiques, lorsqu’ils existent, pour émuler le fonctionnement d’objets (logiques ou physiques). Quid de la fiabilité du simulateur ? Sur quoi repose-t-elle ? Son implémentation ? Ses modèles ? La question reste ouverte et dans le cadre du modèle de plate-forme que nous souhaitons mettre en œuvre, ce sont des éléments nécessaires à prendre en compte pour pouvoir donner des verdicts de test irréfutables.

BIBLIOGRAPHIES

[Abadi et Gordon 97] M. Abadi et A.D. Gordon. “A calculus for cryptographic protocols: the spi calculus”, dans *Proc. of the Fourth ACM Conference on Computer and Communications Security*, ACM Press, Zurich, Swiss, 1997, pp 36-47.

[Abrial 96] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[Acharya et al. 04] S. Acharya, C. George et H. Mohanty. “Specifying a Mobile Computing Infrastructure and Services”, dans *Proc. of 1st International conference on Distributed Computing & Internet Technology (ICDCIT’04)*, LNCS 3347, Springer-Verlag Berlin Heidelberg, India, Dec. 2004, pp. 244-254.

[Arnold 90] A. Arnold, “MEC: A System for Constructing and Analysing Transition Systems”, dans Joseph Sifakis, editeur, *International Workshop on Automatic Verification Methods for Finite State Systems*, Springer-Verlag, LNCS 407, 1990, pp 117-132.

[Axelsen 04] E.W. Axelsen, E.B. Johnsen, et O. Owe. “Toward reflective application testing in open environments”, dans *Proceedings of the Norwegian Informatics Conference* (*NIK 2004*), Trondheim, Norway, 2004, pp. 192–203

[Baget et Mugnier 02] J. Baget et M. Mugnier. “Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints”, *J. Artificial Intelligence*, Res. (JAIR) 16, 2002, pp. 425-465.

[Bai et al. 03] F. Bai, N. Sadagopan, et A. Helmy. ”Important: a framework to systematically analyze the impact of mobility on performance of routing protocols for ad hoc networks”, dans *Proceedings of IEEE Information Communications Conference (INFOCOM 2003)*, IEEE CS Press San Francisco, Etats-Unis, Apr. 2003.

[Barton et Vijayaragharan 03] J. Barton et V. Vijayaragharan. “Ubiwise: A Simulator for Ubiquitous Computing Systems Design”, rapport technique HPL-2003-93, Hewlett-Packard Labs, 2003.

[Baumeister et al. 03] H. Baumeister *et al*. “UML for Global Computing”, dans *Global Computing: Programming Environments, Languages, Security, and Analyisis of Systems, GC 2003*, LNCS 2874, Springer-Verlag Berlin Heidelberg, 2003, pp. 1-24

[Beizer 90] B. Beizer. *Software Testing Techniques*, Van Nostran Reinhold, New York, 1990.

[Bettstetter 01] C. Bettstetter. “Smooth is Better than Sharp: A Random Mobility Model for Simulation of Wireless Networks”, dans *Proc. ACM Intern. Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM)*, ACM Press Rome, Italy, July 2001.

[Bogza et al. 98] Bogza et al. ”KRONOS: a model-checking tool for real-time systems”, dans *Proc. of the 10th Int. Conf. on Computer Aid Verification (CAV’98)*, Springer-Verlag, LNCS 1427, Canada, pp 546-550.

[Boudol 92] G. Boudol. “Asynchrony and the □-calculus”, *Technical Report 1702, INRIA, Sophia-Antipolis*,1992

[Boyer et Moore 79] R. Boyer et J. Moore. *A computational Logic,* Academic Press, 1979.

[Briand *et al.* 06] L. Briand , Y. Labiche et J. Leduc, "Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software", IEEE Transaction on Software Engineering 32(9), Sept. 2006, pp. 642-663.

[Broch et al. 98] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, et J. Jetcheva. "A performance comparison of multi-hop wireless ad hoc network routing protocols", dans *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking(Mobicom98),* ACM Press, Texas, Etats-Unis, Oct. 1998, pp 85-97.

[C. Schroth *et al. 05*] C. Schroth *et al*, "Simulating the traffic effects of vehicle-to-vehicle messaging systems", dans *Proc. 5th Int. Conf. on ITS Telecommunications (ITST 2005),* France, Jun. 2005.

[Cardelli et Gordon 00a] L. Cardelli et A. D. Gordon. "Mobile Ambients", *Theoretical Computer Science, Special Issue on Coordination,* D. Le Métayer Editor. Vol 240/1, Jun. 2000. pp 177-213.

[Cardelli et Gordon 00b] L. Cardelli and A. D. Gordon. "Anytime, Anywhere, Modal Logics for Mobile Ambients", *Proc. of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM Press, Etats-Unis, 2000, pp. 365-377.

[Castanet et al. 98] R. Castanet, O. Kone et P. Laurencot. "On the Fly Test Generation for Real Time Protocol", dans *Intl. Cof. On Computer Communications and Networks (ICCCN'98),* Etats-Unis, 1998, pp 378-387.

[Cavalli et al. 04] A. Cavalli, C. Grepet, S. Maag et V. Tortajada, "A validation Model for the DSR protocol", dans *Proc. of the 24th Int. Conf. on Distributed Computing Systems Workshops (ICDCSW'04),* IEEE CS Press, Japan, Mar. 2004, pp. 768-773.

[Cavalli et al. 09] A. Cavalli, S. Maag, E. Montes de Oca et W. Jimenez, "A Passive Conformance Testing Approach for a Manet Routing Protocol", dans 24th Annual ACM Symposium on Applied Computing (SAC' 09), ACM Press, Mars 2009, Hawaii, Etats-Unis.

[Cavalli et al. 96] A. Cavalli, B. M. Chin et K. Chon. "Testing methods for SDL systems", *Computer Networks and ISDN Systems* 28(12), 1996, pp 1669-1683.

[Cavin *et al.* 02] D. Cavin, Y. Sasson et A. Schiper. "On the Accuracy of MANET Simulators", dans *Proc. 2nd Int. Workshop of Principles of Mobile Computing (POMC'02),* ACM Press, Toulouse, France, 2002, pp 38–43.

[Chan et al 05] W.K. Chan, T.Y. Chen et H. Lu. "A Metamorphic Approach to Integration Testing of Context-Sensitive Middleware-Based Applications", dans *Proc. of the fifth International Conference on Quality Software (QSIC'05),* IEEE CS Press, Australie, pp 241-249.

[Chan et al. 05] W. K. Chan, T. Y. Chen et H. Lu "A Metamorphic Approach to Integration Testing of Context-Sensitive Middleware-Based Applications", dans *Proceedings of the Fifth International Conference on Quality Software (QSIC'05),* IEEE CS Press, Washington DC, Etats-Unis, 2005, pp 241-249.

[Chandy et Mirsa 98] K.M. Chandy et J. Mirsa. "Parallel Program Design: A Foundation", Addison-Wiley, New York, NY, 1998.

[Chen et al 03] T.Y. Chen, T. H. Tse et Z.Q. Zhou. ”Fault-based testing without the need of oracles”, dans *Information and Software Technology*, 45 (1): 1–9, 2003.

[Choffnes et Bustamante 05] D. R. Choffnes et F. E. Bustamante. “An Integrated Mobility and Traffic Model for Vehicular Wireless Networks”, dans *Proc. of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET)*, ACM Press, Allemagne, Sep. 2005, pp 69-78.

[Chow 78] T. Chow. «Testing software design modeled by finite-state machines», *IEEE Transactions on Software Engineering* 4(3), 1978, pp 178-187.

[Chtcherbina et Franz 03] E. Chtcherbian et M. Franz. “Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment”, dans *Proc. of Int. Conf. on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w)*, L’Aquila, Italy, Jan. 2003.

[Clarke et Emerson 81] E. Clarke et E. Emerson. “Design and synthesis of synchronisation skeletons using branching time temporal logic”, dans *Logics of Programs Workshop*, Springer-Verlag, LNCS 131, Etats-Unis, pp 52-71.

[Colin et Puaut 01] A. Colin et I. Puaut. « Worst-case Execution Time analysis of the RTEMS Real-Time Operating System», dans *13th Euromicro Conference on Real-Time Systems*,IEEE CS Press, Pays-Bas, 2001, pp 191-198.

[COMEeSafety] http://www.comesafety.org/

[Cooltown] http://champignon.net/cooltown.php

[Coopers] http://www.coopers-ip.eu/

[Couderc et Banâtre 03] P. Couderc et M. Banâtre. “Ambient computing applications: an experience with the SPREAD approach”, dans *Proc. Of the 36th Hawaii International Conference on System Sciences (HICSS’03)*, IEEE CS Press, Hawaii, Jan. 2003, 9 pp.

[Cousot et al. 05] P. Cousot et al, “The ASTRÉE analyser”, dans *ESOP 2005 — The European Symposium on Programming*, Lecture Notes in Computer Science 3444, Avril 2005, Edinburgh, pp. 21-30.

[Cristian et Schmuck 95] F. Cristian et F. Schmuck, «Agreeing on Processor Group Membership in Timed Asynchroonous Distributed Systems », Rapport technique CSE95-428, UCSD, 1995.

[CYBERGUIDE] http://www.cc.gatech.edu/fce/cyberguide/

[Damm et Harel 01] W. Damm et D. Harel. ”LSCs: Breathing Life into Message Sequence Charts”, Formal Methods in System Design, 19, 45–80, 2001, Kluwer Academic Publishers.

[de Bruijn 68] N. de Bruijn. “The mathematical language AUTOMATH, its usage, and some of its extensions”, dans *Sym. On Automatic Demonstration*, Springer-Verlag, Lecture Notes on Mathematics 125, France, pp 29-61.

[De Bruin et al. 04] de Bruin D., Kroon, J., van Klaverem R., Nelisse M.. ”Design and test of a cooperative adaptive cruise control system”, dans *Intelligent Vehicles symposium*, IEEE CS Press, Pays-Bas, June 2004, pp.392-396.

[De Nicola et al. 97] R. De Nicola, G.-L. Ferrari et R. Pugliese. “Locality based Linda: programming with explicit Localities”, dans *Proc. TAPSOFT’97*. Lecture Notes in Computer Science 1214, Springer Verlag, 1997, pp 712-726.

[Devarapalli et al. 01] V. Devarapalli, A. A. Selcuk, et D. Sidhu. "MZR: A Multicast Protocol for Mobile Ad Hoc Networks", dans *Proc. of the IEEE Intl. Conf. on Communications (ICC)*, Helsinki, Finland, Juin 2001, pp 886 – 891.

[Di et al. 08] P. DI, Y. Houri et K. Kutzner. “Towards Comparable Network Simulations”, Technique report 2008-9, Dept. of Computer Science, Université de Karlsruhe.

[Eichler et al. 05] S. Eichler, B. Ostermaier, C. Schroth et T. Kosch, “Simulation of Car-to-Car Messaging: Analyzing the Impact on Road Traffic”, dans *Proc. of 13th Int. Sym. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS’05)*, Atlanta, Etats-Unis, 2005, pp 507-510.

[Emerson et Halpern 82] E. Emerson et J. Halpern. ”Decision procedures and expressiveness in the temporal logic of branching time”, dans *14th ACM symposium on Theory of Computing (STOC’82)*, ACM Press, Etat-Unis, pp 169-180.

[Ernst et al. 95] J. Ernst, Y. Tanurhan et K. Muller-Glaser. “Using ObjectCharts to specify and design electronic systems”, dans *Proc. of the 7th Euromicro Workshop on Real-Time Systems (EUROMICRO-RTS'95)*, IEEE CS Press, Odensen, Danemark, June 1995 pp. 169-176.

[Ferdinand et Heckmann 04] C. Ferdinand et R. Heckmann, “aiT: Worst-Case Execution Time Prediction by Static Program Analysis”, dans IFIP International Federation for Information Processing, Vol 156/2004, 2004, pp. 377-383.

[Fernandez et al. 96] J. C. Fernandez et al. « Using On-the-Fly Verification Techniques for the Generation of Test Suites », dans *8th Int. Conf. on Computer-Aided Verification (CAV 96)*, Springer-Verlag, LNCS 1102, 1996, pp 348-359.

[Flynn et al. 02] J. Flynn, H. Tiwari et D. O’Mahony. ”A Real-Time Emulation System for Ad Hoc Networks”, dans *Proc. of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, Jan. 2002.

[Fournet et Gonthier 96] C. Fournet, et G. Gonthier. “The reflexive CHAM and the join calculus**”**, dans *Proc. 23rd Annual ACM Symposium on Principles of Programming Languages*, ACM Press, Florida, Etats-Unis, 1996, pp 372-385.

[Fuggetta et al. 98] A. Fuggetta, G. P. Picco et G. Vigna. “Understanding Code Mobility”, IEEE Transactions on software engineering 27(5), Mai 1998.

[Garland et Guttad 91] S. Garland et J. Buttad. ”A guide to LP the Larch Prover”, rapport technique 82, Digital Equipment Corporations, Systems Research Center, 1991.

[Gaudel 95] M.C. Gaudel. “Testing can be formal, too”, dans *6th Intl. Conf. on Theory and Practice of Software Development (TAPSOFT’95)*, Springer-Verlag, LNCS 915, 1995, pp 82-96.

[Gelernter 85] D. Gelernter. “Generative Communication in Linda”. ACM Transactions on Programming Languages and Systems (TOPLAS) 7(1), ACM Press, Jan. 1985.

[GlomoSim] http://pcl.cs.ucla.edu/projects/glomosim/

[Gordon et al. 79] M. Gordon, R. Milner et C. Wadsworth. ”Endinbourgh LCF: A Mechanised Logic of Computation”, Springer-Verlag, LNCS 78, 1979.

[Gordon et Melham 93] M. Gordon et T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambridge University Press, 1993.

[Grabowski *et al.* 93] J. Grabowski, D. Hogrefe et R. Nahm, “Test Case Generation with Test Purpose Specification by MSCs”, dans *Proc. of the 6th SDL Forum (SDL'93)*, North Holland, Oct. 1993, pp. 253-166.

[Grassi et al. 04] V. Grassi, R. Mirandola et A. Sabetta. “A UML Profile to Model Mobile Sytem”, *UML 2004*, LNCS 3273, Springer-Verlag Berlin Heidelberg, 2004, pp.128-142.

[Guennoun 06] M. K. Guennoun, “Architectures Dynamiques dans le Contexte des Applications à Base des Composants et Orientés Services”, PhD Thesis, University of Toulouse III, France, Dec. 2006.

[Guennoun et Drira 06] K. Guennoun et K. Drira. “Using graph grammars for interaction style description. Application to service-oriented architectures”, *Int. Journal on Computer Systems Science Engineering*, 21(4), Jul. 2006, pp. 293-299.

[GUIDE] http://www.guide.lancs.ac.uk/

[Halpin 01] T. A Halpin. “Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design”. Morgan Kaufman Publishers, San Francisco, 2001.

[Harel et al. 07] David Harel, Asaf Kleinbort et Shahar Maoz. ”S2A: A Compiler for Multi-modal UML Sequence Diagrams”, dans *Proc. of 10th International Conference on Fundamental Approaches to Software Engineering (FASE 2007)*, Springer Berlin, Vol 4422, 2007, pp 121-124.

[Harel et Maoz 08] David Harel et Shahar Maoz. ”Assert and negate revisited: Modal semantics for UML sequence diagrams”, Software and Systems Modeling, 7(2):237–253, May, 2008.

[Harri et Fiore 06] J. Harri et M. Fiore. ”VanetMobiSim – Vehicular Ad hoc Network mobility extension to the CanuMobiSim framework”, rapport technique, Institut Eurécom, 2006.

[Held et al. 02] A. Held, S. Buchholz, et A. Schill. “Modeling of context information for pervasive computing applications”, dans *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI '02)*, Orlando, Etats-Unis, July 2002.

[Henricksen et Indulska 06] K. Henricksen et J. Indulska, ”Developing context-aware pervasive computing applications: Models and approach”, dans *Pervasive and Mobile Computing 2 (2006) 37–64, Elsevier, pp. 37-64*

[Henzinger et al 02] T. A. Henzinger, P. H. Ho et K. Shii. ”Towards the animation of proofs-testing proofs by examples”, *Theoritical Computer Science*, 272(1-2): 177-195, 2002.

[HIDENETS] http://www.hidenets.aau.dk/

[HIDENETS D5.2] H. Waeselynck, Z. Micskei, M.D. Nguyen, N. Rivière , « Preliminary testing framework and methodology. Deliverable n° D5.2 », HIDENETS, Project IST-FP6-STREP-26979, Décembre 2007.

[HIDENETS D5.3] G. Huszerl et al., “Refined design and testing framework, methodology and application results”, Hidenets Deliverable D5.3, Déc. 2008.

[Holzmann 97] G. J. Holzmann. “The model checker SPIN”, *IEEE Transcripts on Software Engineering*, 23(5): 279-295, 1997.

[Hong et al. 99] X. Hong, M. Gerla, G. Pei, et C.-C. Chiang. “A group mobility model for ad hoc wireless networks”, dans *Proc. ACM Intern. Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM)*, ACM Press, Seattle, Etats-Unis, Août 1999, pp 53-60.

[Huang 75] J. C. Huang. “An approach to program testing”, *ACM Computing Surveys* 7(3), 1975, pp 113-128.

[Huang et al. 04] Q. Huang, C. Julien, et G. Roman, “Relying on Safe Distance to Achieve Strong Partitionable Group Membership in Ad Hoc Networks”, IEEE Transactions on Mobile Computing 3(2), Apr. 2004, pp. 192-205.

[IMPOTANT] http://nile.usc.edu/important/

[Indulska et al. 03] J. Indulska, R. Robinsona, R. Rakotonirainy et K. Henricksen, “Experiences in using cc/pp in context-aware systems”. dans *Proceedings of the 4th International Conference onMobile Data Management (MDM2003)*, LNCS 2574 Melbourne/Australia, Jan. 2003.

[ITU 02] ITU-T. Specification and Description Language (SDL). Recommendation Z.100, ITU-T, August 2002.

[ITU 99] Z. 120 ITU-TS Recommendation Z. 120: Message Sequence Chart (MSC), ITU-TS, Geneva, Sept. 1999.

[J. Lessman et al. 08] J. Lessman, P. Janackik, L. Lachev et D. Ofanus. ”Comparative Study of Wireless Nework Simulators”, dans Proc. of 7th Intl. Conf. on Networking (ICN’08), IEEE CS Press, Mexico, Arvil 2008, pp 517-523.

[JistSWANS] http://jist.ece.cornell.edu/

[Johnson 05] D. B. Johnson. “Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks”, dans *Wireless Networks (WINET)*, ACM et Springer, 11(1-2):21-38, Jan. 2005.

[Johnson et al. 01] D. B. Johnson, D. A. Maltz et Josh Broch. “DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks”, dans *Ad Hoc Networking*, Charles E. Perkins, Chapter 5, Addison-Wesley, 2001, pp. 139-172.

[Johnson et Maltz 96] D. B. Johnson et D. A. Maltz. “Dynamic Source Routing in Ad Hoc Wireless Networks”, dans *Mobile Computing*, Tomasz Imielinski and Hank Korth, Chapter 5, Kluwer Academic Publishers, 1996, , pp 153-181

[Kerbrat et al. 99] A. Kerbrat, T. Jéron et R Groz. ”Automated test generation from SDL specifications”, dans Rachida Dssouli, Gregor von Bochmann, and Yair Lahav, editors, *SDL ’99 The Next Millennium, 9th International SDL Forum,* Elsevier, Montréal, Québec, Canada, pp 135-152.

[Killijian et al. 09] M.O. Killijian, D. Powell, M. Roy et G. Severac. ”Experimental evaluation of ubiquitous systems. Why and how to reduce WiFi communication range”,

dans *2nd International Conference on Distributed Event-Based Systems (DEBS 2008)*, Rome, Italie, Juillet 2008.

[Kock et al. 98]. B. Koch, J. Grabowski, D. Hogrefe et D. Schmitt. ”Autolink - A Tool for Automatic Test Generation from SDL Specifications”, dans *IEEE International Workshop on Industrial Strength Formal Specification Techniques, (WIFT’98)*, Boca Raton, Florida, USA, Oct. 1998, pp 227-244.

[Kruger et al. 04] I. Kruger, W. Prenningers et R. Sandner, ”Broadcast MSCs”, Formal Aspects of Computing (2004) 16: 194–209.

[Kugler *et al.* 07] H. Kugler, M.J. Stern et E.J.A. Hubbard, “Testing Scenario-Based Models”, dans *Proc. Fundamental Approaches to Software Engineering (FASE '07)*, LNCS 4422, Springer-Verlag, Braga, Portugal, 2007, pp. 306-320.

[Ladani *et al.* 05] B.T. Ladani, B. Alcalde et A. Cavalli, “Passive testing – a constrained invariant checking approach”, dans *Proc. 17th IFIP Int. Conf. on Testing of Communicating Systems (TestCom 2005)*, LNCS 3502, Springer-Verlag, Québec, Canada, 2005, pp. 9-22.

[Laprie et al. 96] J. Laprie et al. , “Guide de la sûreté de fonctionnement“, 2ème édition. Cépaduès Editions.

[Larsen et al 97] K. Larsen, P. Petterson et W. Yi. “UPPAAL in a nutshell”, *International Journal of Software Tools for Technology Transfer*, 1(1-2): 134-152, 1997.

[Le Métayer 98] D. Le Métayer. “Describing Software Architecture Styles Using Graph Grammars”, IEEE Transactions on Software Engineering 24,7 Jul. 1998, pp 521-533.

[Lei et Zhang 05] S. T. Lei et K. Zhang, “Mobile Context Modelling using Conceptual Graphs”, dans *Proc. of IEEE Conference on Wireless And Mobile Computing, Networking And Communications (WiMob'2005)*, IEEE CS Press, Canada, Août 2005, pp 131-138.

[Leung et al. 04] K. R.P.H Leung, Joseph K-Y Ng et W.L. Yeung. “Embedded Program Testing in Untestable Mobile Environment: An Experience of Trustworthiness Approach”, dans *Proc. of the 11th Asia-Pacific Software Engineering Conference (APSEC’04)*, IEEE CS Press, Korea, Dec. 2004, pp 430-437.

[Liang et Haas 99] B. Liang et Z. J. Haas. “Predictive Distance-Based Mobility Management for PCS Networks”, dans *Proc. of IEEE Information Communications Conference (INFOCOM 1999)*, IEEE CS Press, New York, Etats-Unis, Apr. 1999, pp 1377-1384.

[LIME] http://lime.sourceforge.net

[Lochert et al. 05] C. Lochert et al. “Multiple simulator interlinking environment for IVC”, dans *Proc. of the 2nd ACM international workshop on Vehicular ad hoc networks VANET’05*, ACM Press, Allemagne, 2005, pp 87-88

[Lyytinen et Yoo] K. Lyytinen, K. et Y. Yoo, “Issues and challenges in ubiquitous computing”, Communications of the ACM, Vol 45(12), ACM Press, Déc. 2002, pp 62-96.

[Marre 95] B. Marre. “LOFT: A Tool for Assisting Selection Test Data Sets from Algebraic Specifications”, dans *TAPSOFT: Theory and Practice of Software Development*, Springer-Verlag, LNCS 915, 1995, pp 799-800.

[McMillan 93] K. McMillan. “Symbolic Model Checking”, Kluwer Academic Publishers, Boston, 1993.

[Messmer et Bunke 00] B. Messmer, et H. Bunke, “Efficient subgraph isomorphism detection: a decomposition approach”, *IEEE Trans. on Knowledge and Data Engineering*, 12(2), 2000, pp. 307-323.

[Meyers 79] G. Meyers. *The Art of Software Testing*, Wiley, New York, 1979.

[Micskei et al. 06] Z. Micskei, H. Waeselynck, M. D. Nguyen, et N. Riviere. “Analysis of a group membership protocol for Ad-hoc networks”, Rapport technique LAAS *no. 06797*, Nov. 2006.

[Milner et al 92] R. Milner, J. Parrow et D. Walker, “A calculus of mobile processes, Parts 1-2”, *Information and Computation*, 100(1), 1-77. 1992.

[Morla et Davies 04] R. Morla et N. Davies, “Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment”, IEEE Pervasive computing, Vol.3, No.2, Jul.-Sep. 2004, pp 48-56.

[MoSAIC] http://www.laas.fr/mosaic/

[Murphy *et al.* 01] A. L. Murphy, G. P. Picco et G-C. Roman. “Lime: A Middleware for Physical and Logical Mobility”, *Proc. of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, IEEE CS Press, USA, Apr. 2001, pp. 524-233.

[Naguib et Coulouris 01] H. Naguib et G. Coulouris. « Location Information Management », dans Proc. of Intl. Sym. on *Ubiquitous Computing* (*Ubicomp'01*), Atlanta, Etats-Unis, Sep. 2001.

[Nickelsen et al. 06] A. Nickelsen, M. N. Jensen, E. V. Matthiesen et H-P. Schwefel, “Scalable emulation of dynamic multi-hop topologies”, dans *Proc. of The Fourth International Conference on Wireless and Mobile Communications (ICWMC)*, IEEE CS Press, Washington DC, Etats-Unis, 2008, pp 268-273.

[Nickerson 05] J. V. Nickerson, “A Concept of Communication Distance and Its Application to Six Situations in Mobile Environment”, IEEE Transactions on Mobile Computing, Vol 4, 5, Sept/Oct 2005, pp 409-420.

[Nielsen et Geogre 98] M. Nielsen et C. George. “The RAISE language, method and tools”, dans *Proc. of the 2nd VDM-Europe Symposium on VDM-The Way Ahead*, Springer-Verlag New York, Inc, Ireland, 1998, pp. 376-405.

[Nguyen 06] M.D. Nguyen, «Testing applications and services in mobiles systems», ReSIST Student Seminar, San Miniato, Italy, Septembre 2006

[Nguyen et al. 08] M.D. Nguyen, H. Waeselynck, N. Rivière, “Testing mobile computing applications : towards a scenario language and tools”, 6th Workshop on Dynamic Analysis (WODA 2008), ACM Press, Washington D.C, USA, Jul. 2008.

[Noudem et Viho 05] F. N. Noudem et C. Viho, “Modeling, Verifying and Testing the Mobility Management in the Mobile Ipv6 Protocol”, dans *Proc. 8th Int. Conf. on Telecommunications (ConTEL 2005)*, IEEE CS Press, Croatia, June 2005, pp 619-626.

[NOW] www.network-on-wheels.de/

[NS2] http://www.isi.edu/nsnam/ns/

[Offutt et al. 03] J. Offutt, S. Liu, A. Abdurazik et P. Ammann. ”Generating Test Data from State-Based Specifications”, *Journal of Software Testing, Verification and Reliability* 13, 2003, pp 25-53.

[OMG 07] Object Management Group. "UML 2.1.2 Superstructure Specification," formal/07-11-02, 2007.

[OpenMobileIS] http://www.openmobileis.org/

[Owre et al. 95] S. Owre, J. Rushby, N. Shankar et F. von Henke. ”Formal verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS”, *IEEE Trans. On Software Engineering*, 21(2), 1995, pp 107-125.

[Park et Shaw 91] C. Y. Park et A. C. Shaw. ”Experiments with a Program Timing Tool Based on Source-Level Timing Schema”, *IEEE Computer*, 24(5): 48-57, 1991.

[Phalippou et Groz 90] M. Phalippou et R. Groz. « Evaluation of an empirical approach for computer-aided test case generation», dans *3rd Int. Workshop on Protocol Test Systems*, Etats-Unis, 1990, pp 131-147.

[Pickin 03] Simon Pickin. « Test des composants logiciels pour les télécommunications », thèse de doctorat, Université de Rennes, France, 2003.

[Pickin et Jézéquel 04] S. Pickin et J-M. Jézéquel. “ Using UML sequence diagrams as the basis for a formal test description language”, dans *Proc. of 4th Int. Conf. on Integrated Formal Methods (IFM2004)*, LNCS 2999, Springer-Verlag, 2004, pp. 481-500.

[Pimont et Rault 79] S. Pimont et J. C. Rault. « An approach towards reliable software», dans *Proc. of the 4th Int. Conf. On Software Engineering*, Allemagne, 1979, pp 220-230.

[PLATONIS] http://www-lor.int-evry.fr/platonis/

[Pnueli 81] A. Pnueli. “The temporal semantics of concurrent programs”, *Theoretical Computer Science*, 13:45-60, 1981.

[PolySpace] http://www.polyspace.com/

[PReVENT] http://www.prevent-ip.org/

[Puschner et Koza 89] P. Puschner et C. Koza. ”Calculating the Maximum Execution Time of Real-Time Programes”, *Real-Time Systems*, 1(2): 159-176, 1898.

[Rapps et Weyuker 85] S. Rapps, E. J. Weyuker, « Selecting software test data using data flow information«, IEEE Trans. On Software Engineering, Vol SE-11, no 4, Avril 1985, pp 367-375.

[Roman et McCann 98] G-C. Roman et P.J. McCann. “An Introduction to Mobile UNITY”, dans *Proc. of the Int. Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA’98)*, LNCS 1388, Springer-Verlag Berlin Heidelberg, 1998, pp. 871-880.

[Ryan 99] N. Ryan. “ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server”, 1999. http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html.

[Sabnani et Dahbura 85] K. K. Sabnani et A. Dahbura. « A new technique for generating protocol tests», dans *Proc. of the 9th Data Communication Symposium*, ACM Press, Canada, 1985, pp 36-43.

[Salber et al. 99] D. Salber, A.K. Dey et G. D. Abowd, "The Context Toolkit: Aiding the development of Context-Enabled Applications", dans *Proceedings of CHI'99, Pittsburgh, PA,* ACM Press, Pennsylvania, Etats-Unis, Mai 1999 pp 434-441.

[Sanmiglingam et Coulouris 02] K. Sanmiglingam et G. Coulouris. "A Generic Location Event Simulator", dans *UbiComp 2002,* LNCS 2498, Springer-Verlag Berlin Heidelberg, 2002, pp. 308-315.

[Satoh 03] I. Satoh. "A Testing Framework for Mobile Computing Software", IEEE Transactions on software Engineering, Vol.29, No.12, IEEE CS Press, 2003, pp. 1112-1121.

[Schilit et al 94] B. Schilit, N. Adams et R. Want. "Context-aware computing applications", dans *IEEE Workshop on Mobile Computing Systems and Applications,* IEEE CS Press, Santa Cruz, CA, Etats-Unis, Déc. 1994.

[Schroth et al. 05] C. Schroth et al. "Simulating the traffic effects of vehicle-to-vehicle messaging systems," Proceedings of ITS Telecommunication, 2005.

[Sengupta et Cleaveland 06] B. Sengupta et R. Cleaveland, "Triggerred Message Sequence Charts", IEEE Trans. on Software Engineering, 32(8), Aug. 2006, pp. 587-607.

[Sengupta et Cleaveland 06] B. Sengupta et R. Cleaveland, "TRIM: A Tool for Triggerred Message Sequence Charts", dans *Proc. of the 15th Computer Aided Verification Conf. (CAV '03)*, Springer-Verlag, LNCS 2725, 2003.

[Sheng et Benatallah 05] Q. Z. Sheng et B. Benatallah, "ContextUML: A UML-Based Modelling Language for Model-Driven Development of Context-Aware Web Services", dans *le Proceedings of the Intl. Conf. on Mobile Bussiness (ICMB'05)*, IEEE CS Press, Australie, July 2005.

[Simons 07] C. Simons, "CMP: A UML Context Modeling Profile for Mobile Distrubuted Systems", *dans Proceedings of the 40th Annual Hawaii Int. Conf. on Systems Sciences (HICSS'07)*, IEEE CS Press, Etats-Unis, Jan. 2007, pp 289-298.

[SUMO] http://sumo.sourceforge.net

[TETware] The Open Group, TETware, URL: http://tetworks.opengroup.org/

[Tian et al. 02] J. Tian, J. Hähner, C. Becker, I. Stepanov, et Kurt Rothermel, "Graph-Based Mobility Model for Mobile Ad Hoc Network Simulation", dans *Proceedings of the 35th Annual Simulation Symposium (SS'02)*, IEEE CS Press, Allemagne, Avril 2002, pp 337-344.

[Tse et al. 04] T. H. Tse et al. "Testing context-sensitive middleware-based software applications", dans *Proceedings of the 28th Annual International Computer Software and Applications Conference* (*COMPSAC2004*), volume 1, IEEE CS Press, Los Alamitos, California, 2004, pp 458-465.

[Tse et al. 04] T.H. Tse, Stephan S. Yau, W.K. Chan et Heng Lu. "Testing Context-Sensitive Middleware-Based Software Applications", dans *Proceedings of the 28th Annual*

International Computer Software and Application Conference (COMPSAC 2004), IEEE CS Press, Etats-Unis, 2004, pp 458-466

[Uchitel et al. 02] S. Uchitel, J. Kramer, et J. Magee, “Negative Scenarios for Implied Scenario Elicitation” dans *Proc. of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, ACM Press, Etats-Unis, 2002, pp 109-118.

[Ullmann 76] J.R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1): 1976, pp. 31-52.

[Ural 92] H. Ural. « Formal methods for test sequence generation», *Computer Communications* 15(5), 1992, pp 311-325.

[Vik et Medid 04] K.-H. Vik et S. Medid. "Quality of Service aware source initiated ad-hoc routing", dans *1st IEEE Int. Conf. on Sensor and Ad hoc Communications and Networks*, USA, Oct. 2004.

[Waeselynck 93] H. Waeselynck. “Vérification de logiciels critiques par le test statistique”, thèse de doctorat, Institut National Polytechnique de Toulouse, 1993.

[Waeselynck et al 07] H. Waeselynck, Z. Micskei, M.D. Nguyen et N. Rivière, “Mobile Systems from a Validation Perspective: a Case study”, *Proc. of the 6th International Symposium on Parallel and Distributed Computing (ISPDC’07)*, IEEE CS Press, Austria, July 2007.

[Wang et al. 07] Z. Wang, S. Elbaum et D. S. Rosenblum, “Automated Generation of Context-Aware Tests”, dans *Proc. of the 29th international conference on Software Engineering*, IEEE CS Press, Washington DC, Etats-Unis, 2007, pp 406-415

[Weiser 93] M. Weiser, “Some computer science issue in ubiquitous computing”, Communication of the ACM, Vol. 36, No. 7, July 1993, pp 75-84.

[Willcock *et al.* 05] C. Willcock *et al.*, An Introduction to TTCN-3, Wiley, 2005.

[Zhang et Li 02] Y. Zhang et W. Li. “An Integrated Environment for Testing Mobile Ad-Hoc Networks “, dans *Proc. of the third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, Lausanne, Switzerland, June 2002, pp 104-111.

[Zheng et Ni 02] P. Zheng et L. M. Ni “EMWIN: emulating a mobile wireless network using a wired network”, dans *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, ACM Press, Atlanta, Georgia, USA, 2002, pp 64-73.

Glossaire

AP	Access Point
DFS	Depth First Searching
GMP	Group Membership Protocol
GraphSeq	Graph matching tool for Sequences of configurations
HMSC	High-level Message Sequence Chart
LSC	Live Sequence Chart
MSC	Message Sequence Chart
MSD	Modal Sequence Chart
PCO	Points of Control and Observation
SDL	Specification and Description Language
TERMOS	Test Requirement language for Mobile Setting
TMSC	Triggered Message Sequence Chart
UML	Unified Modeling Language
UTP	UML Testing Profile
V&V	Validation and Verification

Résumé

Les avancées technologiques du monde du « sans fil » ont conduit au développement d'applications dédiées à l'informatique mobile. Leurs caractéristiques propres posent de nouveaux défis pour leur vérification. Ce mémoire traite de la technologie de test pour les systèmes mobiles. Nous avons débuté notre travail par un état de l'art sur le test des applications mobiles conjointement avec une analyse d'une étude de cas, un protocole d'appartenance de groupe (GMP) dans le domaine des réseaux ad hoc, pour nous permettre de mieux comprendre les problèmes liés au test de ces applications. En conséquence, nous avons décidé d'avoir une approche de test qui se base sur les descriptions de scénarios. Nous avons constaté que les scénarios d'interactions doivent tenir compte des configurations spatiales des nœuds comme un concept majeur. Afin de couvrir les nouvelles spécificités des systèmes mobiles dans les langages de descriptions, nous avons introduit des extensions qui portent essentiellement sur les relations spatio-temporelles entre nœuds et sur la communication par diffusion. Le traitement de l'aspect spatial a conduit au développement de l'outil *GraphSeq*. *GraphSeq* a pour but d'analyser des traces de test afin d'y identifier des occurrences de configurations spatiales successives décrites à partir d'un scénario abstrait. L'application de *GraphSeq* (support à l'implémentation des cas de test, vérification de la couverture des traces) est présentée avec les analyses des sorties d'un simulateur de mobilité et des traces d'exécution de l'étude de cas GMP.

Mots-clés : test, systèmes informatiques mobiles, approche de test basée sur les scénarios, descriptions de scénarios, algorithmes d'appariements de graphes, analyse de traces de test.

Abstract

Advances in wireless networking have yielded the development of mobile computing applications. Their unique characteristics provide new challenges for verification. This dissertation elaborates on the testing technology for mobile systems. As a first step, a review of the state-of-the-art is performed together with a case study - a group membership protocol (GMP) in mobile ad hoc settings - that allowed us to gain insights into testing problems. We present, then, a testing approach which bases on the descriptions of scenarios. We note that the scenario interactions must take into account the spatial configurations of nodes as first class concepts. To cover new specificities of mobile systems in description languages, we introduce some extensions that focus on spatio-temporal relations between nodes and on broadcast communication. The processing of spatial aspect leads to the development of the tool *GraphSeq*. *GraphSeq* aims to analyze test traces in order to identify occurrences of successive spatial patterns described in an abstract scenario. The application of *GraphSeq* (support to implementation of test cases, verification of trace coverage) is shown with the analysis of outputs of a simulator and execution traces of the case study GMP.

Keywords: test, mobile computing systems, scenario-based testing approach, scenario languages, graph homomorphism algorithms, trace analysis.