Rôle de la géométrie en informatique diffuse : programmation des applications et navigation contextuelle

Julien Pauty

N° d'ordre: 3266

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Julien Pauty

Équipe d'accueil : ACES - IRISA
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Rôle de la géométrie en informatique diffuse : programmation des applications et navigation contextuelle

soutenue le 24 février 2006 devant la commission d'examen

M. :	Daniel	Herman	Président
MM. :	Yolande	Berbers	Rapporteurs
	David	Powell	
MM. :	Paul	Couderc	Examinateurs
	Michel	Banâtre	

Remerciements

Je remercie Daniel Herman, professeur, qui me fait l'honneur de présider ce jury.

Je remercie Yolande Berbers, professeur, et David Powell, directeur de recherche d'avoir accepter la charge de rapporteur.

Je remercie Michel pour m'avoir proposé cette thèse et pour l'avoir dirigée. Je le remercie également pour son soutien, ses conseils et son intégrité.

Je remercie Paul avec qui j'ai passé trois années enthousiasmantes. Merci pour toutes ces discusions et échanges d'idées endiablés, les conseils ainsi que le soutient.

Je remercie Gilbert pour m'avoir introduit dans le monde de la recherche et sans qui je n'aurais jamais fait de thèse.

Je remercie Claude pour avoir pris en charge les nombreuses étapes administratives précédant la soutenance. Je remercie Carole et Arnaud pour leurs travaux de relecture. Finalement, je salue chaleureusement l'ensemble de l'équipe ACES pour tous ces bons moments passés ensemble.

Table des matières

Table des matières 1

Introduction 7

I État de l'art 11

1 Informatique diffuse 13
1.1 Introduction . . . 13
1.1.1 Principes . . . 13
1.1.2 Exemples d'applications . . . 14
1.1.3 Interactions implicites . . . 15
1.2 Applications contextuelles . . . 15
1.2.1 Contexte et informatique diffuse . . . 16
1.2.2 Classification des applications contextuelles . . . 16
1.2.3 Capture du contexte . . . 16
1.3 Réalisation . . . 17
1.3.1 Approche physique . . . 17
1.3.2 Approche virtuelle . . . 18
1.4 Bilan . . . 18

2 Applications purement diffuses 21
2.1 Présentation . . . 21
2.2 Architecture . . . 22
2.2.1 Architecture logicielle . . . 22
2.2.2 Modèle de programmation . . . 23
2.2.3 Architecture matérielle . . . 23
2.3 Partage de données basé sur un espace de tuples . . . 24
2.3.1 Principes de base . . . 25
2.3.2 Lime : un middleware dédié à la mobilité . . . 25
2.3.3 SPREAD . . . 26
2.4 Partage de données basé sur une structure arborescente . . . 27
2.5 Partage de données basé sur des bases de données . . . 28
2.6 Synthèse . . . 29

3 Marquage électronique **31**
3.1 Présentation . . . 31
3.2 Architecture . . . 32
3.2.1 Architecture logicielle . . . 32
3.2.2 Modèle de programmation . . . 32
3.2.3 Support technologique . . . 33
3.2.4 Infrastructure matérielle . . . 34
3.3 Applications . . . 35
3.3.1 Applications d'annotation contextuelle . . . 35
3.3.2 Réalité enrichie . . . 36
3.3.3 Assistant cuisinier . . . 36
3.4 Comparaison avec les applications purement diffuses . . . 36
3.5 Synthèse . . . 37

4 Contribution **39**
4.1 Propriétés géométriques du monde physique . . . 39
4.2 Géométrie et informatique diffuse . . . 40

II Rôle de la géométrie dans la programmation des applications **41**

5 Calcul dans l'espace physique **43**
5.1 Programmation spatiale . . . 43
5.1.1 Mémoire spatiale . . . 44
5.1.2 Synchronisation physique . . . 46
5.1.3 Exemple d'application . . . 47
5.1.4 Bénéfices de la programmation spatiale . . . 48
5.1.5 Le système SPREAD . . . 49
5.2 Programmation spatiale et applications contextuelles . . . 50
5.2.1 Contexte et proximité . . . 51
5.2.2 Mise en œuvre grâce à la programmation spatiale . . . 51
5.2.3 Application . . . 51
5.3 Limitations . . . 52
5.3.1 Adressage et forme des données . . . 52
5.3.2 Détection d'interactions physiques . . . 53
5.4 Bilan . . . 54

6 Programmation géométrique **55**
6.1 Principe . . . 55
6.1.1 Forme des tuples . . . 55
6.1.2 Adressage géométrique . . . 56
6.1.3 Mise en oeuvre . . . 57
6.2 Réalisation . . . 58
6.3 Évaluation . . . 59

6.4 Discussion 60
6.4.1 Travaux apparentés 60
6.4.2 Bilan 61

7 Synchronisation sur les arrivées et départs d'entités 63
7.1 Limitation du modèle de programmation 63
7.2 Les opérations readOnce et lostOne 64
7.2.1 Principe 65
7.2.2 Étiquetage des interactions physiques 66
7.3 Réalisation 66
7.3.1 Opération readOnce 66
7.3.2 Opération lostOne 67
7.3.3 Support technologique 67
7.3.4 Évaluation 68
7.4 Bilan 69

8 Transfert atomique de tuple 71
8.1 Principe de fonctionnement 71
8.1.1 Atomicité 72
8.1.2 L'atomicité ne peut pas être systématiquement garantie 73
8.2 Protocole 74
8.2.1 Le two-phases commit protocol 74
8.2.2 Adaptation du protocole 2PC pour l'opération take 75
8.2.3 Réduction du nombre de défaillances 76
8.3 Évaluation de la contrainte géométrique 77
8.3.1 Présentation de la simulation 77
8.3.2 Premier scénario : les entités s'approchent 79
8.3.3 Second scénario : les entités s'éloignent 81
8.3.4 Bilan 82
8.4 Prise en compte des défaillances au niveau de l'application 83
8.4.1 Détection des défaillances 83
8.4.2 Restauration de la cohérence 83
8.5 Réalisation 84
8.5.1 Implémentation 84
8.5.2 Évaluation de la réactivité 85
8.6 Bilan 85
8.6.1 Travaux apparentés 85
8.6.2 Discussion 86

III Géométrie et systèmes d'information 87

9 Géométrie et contexte 89
9.1 Contexte et espace physique 89

9.1.1 Notion de contexte . . . 90
9.1.2 Contexte physique . . . 91
9.2 Navigation contextuelle . . . 92
9.2.1 Contexte dans un système d'information . . . 92
9.2.2 Contexte et notion de dimension . . . 93
9.2.3 Navigation contextuelle dans un système d'information . . . 93
9.2.4 Navigation et contexte multidimensionnel . . . 95
9.3 Définition formelle . . . 96
9.3.1 Introduction . . . 96
9.3.2 Définition . . . 96
9.3.3 Contexte dans l'espace physique . . . 97
9.3.4 Mise en œuvre . . . 98
9.4 Discussion . . . 100
9.4.1 Travaux apparentés . . . 100
9.4.2 Bilan . . . 101

10 Navigateur de photos **103**
10.1 Présentation . . . 103
10.1.1 Collection de photos . . . 103
10.1.2 Scénario . . . 104
10.1.3 Fonctionnalités offertes par le navigateur . . . 105
10.1.4 Interactions . . . 106
10.2 Implémentation . . . 106
10.2.1 Architecture Logicielle . . . 107
10.2.2 Interface . . . 108
10.2.3 Construction du contexte . . . 110
10.3 Expérimentations . . . 111
10.4 Bilan . . . 112

Conclusion et perspectives **113**

A Évaluation des opérations readOnce et lostOne **117**
A.1 Évaluation de l'opération readOnce . . . 117
A.1.1 Évaluation du temps de réaction . . . 117
A.1.2 Limitation de la perte des tuples . . . 121
A.1.3 Analyse des résultats . . . 122
A.2 Évaluation de la réactivité de l'opération lostOne . . . 122
A.2.1 Évaluation du temps de réaction . . . 122
A.2.2 Limitation de la perte des tuples . . . 123
A.2.3 Analyse des résultats . . . 124
A.3 Bilan . . . 125

B Similink **127**
B.1 Présentation 127
B.1.1 Principe de fonctionnement 127
B.1.2 Scénarios d'utilisation 128
B.1.3 Schéma de navigation 129
B.2 Réalisation 129
B.3 Bilan 131

Bibliographie **133**

Table des figures **139**

Publications **143**

Introduction

Contexte de l'étude

Les terminaux mobiles tels que les assistants personnels (PDA) ou téléphones mobiles permettent à leurs utilisateurs d'accéder à des services informatiques directement depuis l'environnement physique. Actuellement, le service le plus répandu est la téléphonie mobile. Ce service est global, car accessible quelle que soit la position de l'utilisateur. Dans le cas de la téléphonie, les terminaux sont reliés par un réseau cellulaire global.

Avec le développement des technologies de communication sans fil à courte portée, nous pouvons envisager de nouveaux types de services, non plus globaux mais locaux, car liés à l'environnement physique proche de l'utilisateur [56]. Dans ce cas là, l'environnement physique est peuplé de calculateurs sans fil à courte portée, chacun fournissant des services aux terminaux avec lesquels il peut communiquer. Les services locaux présentent l'intérêt d'être adaptés à la situation courante de l'utilisateur [6]. Par exemple, lorsque l'utilisateur fait la queue devant un cinéma, son terminal peut se connecter au service local offert par le cinéma et lui proposer les bandes annonces des films à l'affiche.

L'informatique diffuse [66] embarque des calculateurs sans fil directement sur les objets de tous les jours [18, 59]. Ces objets sont alors à même de fournir des services aux utilisateurs situés à portée de communication. Les calculateurs embarqués sur les objets peuvent également communiquer ensemble. Nous pouvons ainsi envisager des services composés non seulement d'interactions entre l'utilisateur et les objets proches de lui, mais également entre les objets eux mêmes. Par exemple, considérons que chaque article présent dans un supermarché embarque un micro-calculateur sans fil qui contient les informations qui lui sont relatives, telles que sa compositionăet son prix. Les chariots embarquent également un calculateur sans fil. Ceux-ci peuvent ainsi interroger les articles qu'ils contiennent, afin de calculer leur prix au fur et à mesure des commissions [53].

Dans le cadre de l'informatique diffuse, les calculateurs embarqués sur les objets contiennent les données relatives aux objets correspondant. Nous considérons que les objets portent les données qui leur sont relatives. Nous distinguons deux grandes classes d'applications : i) les applications basées sur des échanges de données entre calculateurs, telles que l'application de calcul du prix d'un chariot ; ii) les applications qui accèdent aux données portées par les objets et les présentent à l'utilisateur sur l'écran de son terminal.

Contribution

Une application d'informatique diffuse est non seulement fonction des données portées par les objets, mais également de l'organisation géométrique de ces objets. En effet, une application d'informatique diffuse délivre des services qui sont fonction de paramètres physiques tels que la position de l'utilisateur ou la proximité d'objets particuliers. Ainsi, l'organisation géométrique de l'espace physique joue un rôle direct dans le fonctionnement des applications d'informatique diffuse.

Le plus souvent, les applications se reposent sur des informations géométriques basées sur la connectivité du réseau. Lorsqu'un objet peut communiquer avec un autre objet, il en déduit que ce second objet se trouve dans sa zone de communication. Par exemple, les interfaces de communication de type Bluetooth ont une portée de communication de 10 mètres. Un objet équipé d'une telle interface peut donc déduire que les objets avec lesquels il peut communiquer sont situés à l'intérieur d'un cercle théorique de 10 mètres de rayon, centré sur sa position courante. Ce type d'information n'est pas suffisamment précis pour réaliser des applications nécessitant un couplage fin avec le monde physique [3]. Par exemple, dans le cas d'une application basée sur des échanges de données entre voitures, une voiture ne peut pas distinguer les données portées par les voitures situées sur sa droite, des données portées par les voitures situées sur sa gauche. Dans le cas d'une application qui présente à l'utilisateur les informations portées par les objets, celle-ci ne peut pas distinguer les informations situées devant l'utilisateur des informations situées derrière lui.

Dans la seconde partie de cette thèse nous présentons un modèle de programmation qui nous permet de prendre en compte précisément l'organisation géométrique de l'espace physique lors du développement des applications. Ce modèle de programmation nous permet notamment de sélectionner les données relativement à l'orientation de l'utilisateur.

Dans la troisième partie de cette thèse nous étudions l'accès par l'utilisateur aux données portées par les objets. Lorsque la densité physique des données est importante, nous verrons qu'il est nécessaire de filtrer les données accessibles afin de ne pas surcharger l'utilisateur par un excès d'information. Dans cette troisième partie, nous nous reposons sur l'organisation géométrique de l'espace physique pour filtrer les données. Pour cela, nous ne présentons à l'utilisateur que les données liées aux objets les plus proches de lui. Nous nous reposons également sur l'organisation géométrique de l'espace physique pour organiser l'affichage des données. Par exemple, nous pouvons afficher les données liées aux objets situés devant l'utilisateur en haut de son écran et les données liées aux objets situés derrière lui en bas de l'écran. Ainsi, l'utilisateur peut relier plus facilement les données affichées à son environnement physique proche.

Organisation du document

Ce document comporte trois parties. Dans la première partie nous introduisons l'informatique diffuse et présentons deux grandes approches pour réaliser une application.

Ces deux approches permettent de construire des applications reposant sur des échanges de données entre objets, ainsi qu'entre objets et utilisateurs.

Notre contribution est présentée dans la seconde et la troisième partie. Dans la seconde partie nous étudions le rôle de la géométrie dans la construction d'applications d'informatique diffuse. Nous présentons tout d'abord un modèle de programmation qui nous permet de prendre en compte, directement au niveau du code des programmes, l'organisation géométrique de l'espace physique. Ce modèle de programmation nous permet notamment de synchroniser les applications sur la détection de configurations géométriques particulières. Dans cette seconde partie, nous présentons également un mécanisme de transfert atomique de données entre objets. Nous verrons que nous pouvons améliorer la fiabilité de ce mécanisme en prenant en compte la configuration géométrique des objets au niveau du protocole de transfert.

Dans la troisième partie nous étudions le rôle de la géométrie au niveau de la construction, de l'accès et de la navigation dans un système d'information. Les informations portées par les objets de l'espace physique sont considérées comme un système d'information particulier. Nous étudions tout d'abord la notion de contexte dans un système d'information et son application à la navigation à travers le système. Nous verrons qu'un système d'information est structuré selon une ou plusieurs dimensions et que le contexte de l'utilisateur est représenté par les données qui sont proches de lui, selon ces dimensions. Finalement, nous présentons une application qui permet à l'utilisateur de naviguer, grâce à son contexte, à travers un système d'information composé de photos numériques.

Première partie

État de l'art

Chapitre 1

Informatique diffuse

Dans ce chapitre nous commençons par introduire l'informatique diffuse. Nous poursuivons en présentant les applications contextuelles, qui représentent une classe d'applications importantes de l'informatique diffuse. Nous terminons en présentant les deux grandes approches pour réaliser une application d'informatique diffuse.

1.1 Introduction

Dans cette partie nous commençons par présenter les principes de l'informatique diffuse. Nous continuons en présentant plusieurs applications et terminons en présentant la notion d'interaction implicite, qui est une notion importante de l'informatique diffuse.

1.1.1 Principes

L'informatique diffuse (*ubiquitous computing*) a pour objectif d'assister implicitement l'utilisateur dans ses tâches quotidiennes [66]. Contrairement à une application d'informatique classique, une application d'informatique diffuse est utilisée dans des situations où l'utilisateur n'est pas entièrement focalisé sur l'utilisation d'un ordinateur. Il peut être en train de marcher, de faire ses courses, de visiter un musée... Dans de telles situations, la capacité de l'utilisateur à interagir avec une application informatique est donc réduite. Une application d'informatique diffuse doit donc fonctionner avec le minimum d'interactions entre l'utilisateur et la machine.

Une application d'informatique diffuse délivre un service local, adapté à l'activité courante de l'utilisateur. Par exemple, dans un magasin, lorsque l'utilisateur tient dans sa main un DVD, la distribution du film s'affiche sur son assistant personnel. Lorsqu'un visiteur de musée se trouve devant la Joconde, le navigateur Web de son téléphone affiche automatiquement une page Web qui décrit la vie de Léonard de Vinci. Une approche commune pour réduire les interactions entre l'homme et la machine est de détecter automatiquement le contexte de l'utilisateur et d'y adapter le service fourni par l'application. Le contexte de l'utilisateur peut être représenté par sa position ou bien par les personnes qui l'entourent. Dans l'exemple précédent, la position de l'utilisateur est

surveillée en permanence pour lui fournir automatiquement des informations pertinentes tout au long de sa visite.

Une application d'informatique diffuse est couplée au monde physique. Les modifications du monde physique ont une influence directe sur l'application. Par exemple, dans un musée lorsque l'utilisateur se déplace d'une œuvre à l'autre, son téléphone lui propose automatiquement les informations liées à l'œuvre la plus proche. Les déplacements de l'utilisateur contrôlent le déroulement de l'application. L'application doit donc disposer de capacités de perception de l'environnement physique pour détecter les changements qui y ont lieu et adapter ses services à ces changements. Les grandeurs physiques telles que la position [65] de l'utilisateur ou la température ambiante sont détectées à l'aide de capteurs spécialisés. L'informatique diffuse embarque également des micro-calculateurs sur les objets physiques [18, 59]. Par exemple, dans un musée nous pouvons embarquer sur chaque œuvre un calculateur qui contient les informations relatives à cette œuvre. Ce calculateur est équipé d'une interface de communication à courte portée. Lorsque l'utilisateur est dans la zone de communication de l'œuvre, son terminal personnel accède aux informations et les lui présente. Dans ce cas là, le terminal utilisateur perçoit l'environnement physique via son interface de communication sans fil.

1.1.2 Exemples d'applications

De par le lien direct de l'informatique diffuse au activités humaines, chaque activité ou processus physique représente une application potentielle de l'informatique diffuse. Nous pouvons distinguer plusieurs classes d'applications : les applications d'annotation contextuelle qui associent des informations aux objets physiques ; les applications impliquant des interactions entre plusieurs objets ou personnes ; les applications agissant sur l'environnement.

Les guides virtuels [1, 7] représentent l'application d'annotation contextuelle la plus connue. Un guide virtuel associe aux œuvres et monuments des informations. Par la suite, en fonction de la position de l'utilisateur, le guide détermine les œuvres et monuments proches de lui et lui fournit les informations correspondantes. Un guide virtuel peut aussi localiser l'utilisateur sur une carte. Le plus souvent, un guide se présente sous la forme d'un assistant personnel numérique (PDA). Dans un supermarché électronique [53] les produits sont associés à des informations telles que leur prix ou leurs ingrédients. Les chariots perçoivent leur contenu et affichent le prix total au fur et à mesure des commissions. En fonction de son contenu, le chariot présente des promotions au client.

Les applications d'informatique diffuse incluent également des applications afin d'aider les mal voyants dans les transports en commun [2, 21]. Par exemple, dans [21] les auteurs présentent un système qui guide un malvoyant dans une gare afin de l'aider à rejoindre le bon quai. Ce système guide le malvoyant au fur et à mesure de ses déplacements grâce à des messages vocaux diffusés dans un casque.

Les espaces intelligents [28, 4] entrent dans la catégorie des applications qui agissent sur l'environnement. Les espaces intelligents s'adaptent aux préférences des utilisateurs présents dans la pièce. Une application d'espace intelligent ajuste par exemple l'am-

biance lumineuse et sonore de la pièce. Elle peut également éteindre la lumière automatiquement lorsque la dernière personne quitte la pièce.

1.1.3 Interactions implicites

Une application d'informatique diffuse doit fonctionner le plus implicitement possible, afin de ne pas perturber ni détourner l'utilisateur de son activité courante. Une interaction implicite correspond à une interaction entre l'utilisateur et son environnement physique qui a également des répercutions sur l'application [57]. Les déplacements de l'utilisateur sont des interactions implicites souvent utilisées en informatique diffuse. De même, le fait de se rapprocher d'un objet ou de le saisir peut être vu comme une sélection implicite.

Les interactions implicites présentent l'avantage d'être intuitives pour l'utilisateur. Il n'a pas à apprendre à se servir de l'application pour interagir avec elle. Il se contente d'agir comme à son habitude.

Les interactions implicites ne sont possibles que si l'application est couplée à l'espace physique : des changements dans l'état du monde physique ont des répercutions directes sur l'application, sans que l'utilisateur n'ait à interagir explicitement avec un ordinateur. Dans la partie suivante, nous présentons les applications contextuelles, qui sont couplées à l'espace physique en détectant et en adaptant automatiquement l'application au contexte de l'utilisateur.

1.2 Applications contextuelles

Les applications contextuelles [56, 6] représentent une catégorie d'applications très importantes de l'informatique diffuse. Au sein d'une application contextuelle, le contexte de l'utilisateur peut être représenté par divers paramètres tels que sa position, son rythme cardiaque ou la luminosité ambiante. L'application détecte automatiquement ce contexte et s'y adapte en conséquence.

Les guides virtuels représentent l'application contextuelle type. Le contexte de l'utilisateur est représenté par sa position et le guide lui fournit des informations en fonction de cette position. La détection automatique du contexte permet de réduire les interactions explicites avec l'ordinateur. Dans le cas d'un guide virtuel, si la contexte de l'utilisateur n'était pas détecté automatiquement, l'utilisateur devrait renseigner manuellement l'application sur sa situation courante. Il devrait par exemple rentrer un code numérique affiché à côté du tableau qu'il regarde, afin d'obtenir les informations correspondantes.

Dans cette partie nous commençons par présenter la notion de contexte. Dans la seconde partie nous présentons quelques techniques de capture du contexte. Nous terminons dans la troisième partie avec une classification des applications contextuelles.

1.2.1 Contexte et informatique diffuse

Dans le cadre de l'informatique diffuse, plusieurs définitions ont été proposées pour la notion de contexte. Chen et Kotz [6] définissent le contexte comme étant l'ensemble des propriétés physiques qui, soit déterminent le comportement de l'application, soit sont pertinentes pour l'utilisateur et lui sont présentées lorsqu'elles évoluent. La définition de Dey et Abowd [13] est la suivante : le contexte est représenté par toute information qui peut être utilisée pour caractériser la situation d'une entité, une entité étant une personne, un lieu ou un objet en rapport avec les interactions entre l'utilisateur et l'application. Nous obtenons une notion de contexte très large pouvant englober toute information concernant l'environnement ou l'utilisateur.

Plus récemment de nouvelles définitions de contexte ont été proposées. Tout comme les définitions précédentes, l'objectif de ces définitions est de caractériser la situation courante de l'utilisateur. Padovitz et al [36] représentent le contexte de l'utilisateur par sa position au sein d'un espace multidimensionnel. Par exemple, si les dimensions considérées sont le rythme cardiaque et la vitesse, la position de l'utilisateur dans cet espace permet de distinguer un utilisateur dans un bus (rythme cardiaque normal et vitesse élevée) d'un utilisateur qui court (rythme cardiaque élevé et vitesse élevée). Wang et al [63] décrivent le contexte de l'utilisateur en se basant sur une ontologie, qui est une hiérarchisation des différents éléments intervenant dans la description du contexte. Les auteurs s'appuient sur un exemple de maison intelligente. Dans cette maison, les éléments intervenant dans la description du contexte sont les personnes, les lieux, les éléments informatiques. Ces éléments peuvent être mis en relation pour décrire une situation, par exemple : John est dans la chambre. Utiliser une telle ontologie permet, via des règles prédéfinies, de déduire des situations de plus haut niveau, par exemple : John est allongé dans la chambre qui a ses rideaux fermés implique que John dort.

1.2.2 Classification des applications contextuelles

Pascoe a proposé de classifier les applications contextuelles en quatre catégories [37] :
- l'annotation contextuelle qui consiste à enrichir l'environnement par des notes électroniques diffusées à l'utilisateur lorsqu'il en est proche ;
- la navigation contextuelle qui situe l'utilisateur sur une carte et lui propose des services en fonction de sa position ;
- l'enrichissement contextuel qui ajoute des informations contextuelles aux données saisies par l'utilisateur, telles que des notes électroniques qui sont enrichies par la position de l'utilisateur au moment de la saisie ;
- l'adaptation contextuelle qui adapte le service délivré par l'application au contexte de l'utilisateur.

1.2.3 Capture du contexte

Pour s'adapter automatiquement au contexte de l'utilisateur, l'application doit donc capturer ce contexte. Capturer le contexte de l'utilisateur revient à extraire de l'environnement physique les paramètres qui caractérisent la situation courante de l'utilisateur.

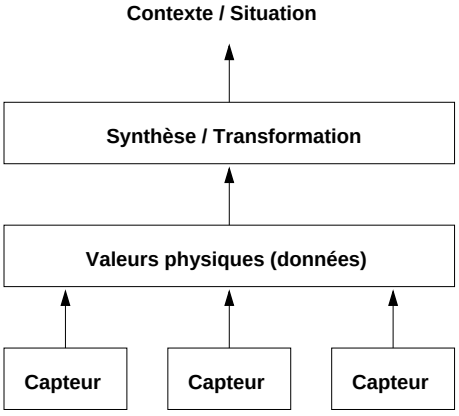

FIG. 1.1 – Capture du contexte depuis l'environnement physique. Les données captées sont d'abord regroupées puis analysées pour déduire le contexte de l'utilisateur

Si nous considérons les données physiques telles que la position ou la température, la capture est réalisée directement par des capteurs adaptés. La position étant l'information la plus utilisée, plusieurs solutions ont été développées, la plus connue étant le GPS. Hightower et Borriello recensent et classent les systèmes de localisation dans un document de synthèse [25]. Des capteurs dédiés à d'autres informations, telles que l'accélération, la température ou le cap sont également disponibles, leur utilisation dans l'informatique contextuelle étant par ailleurs présentée dans [58]. En plus des capteurs dédiés, la connectivité du réseau permet également de caractériser le contexte de l'utilisateur. Par exemple, si nous considérons que toutes les personnes sont équipées d'un terminal sans fil courte portée, le contexte d'une personne peut être représenté par l'ensemble des terminaux qui sont à portée de communication.

Les données issues des capteurs sont le plus souvent transformées afin d'obtenir des informations de plus haut niveau. Nous pouvons par exemple transformer des coordonnées géographiques en un nom de lieu. La figure 1.1 illustre les différentes étapes de la capture du contexte de l'utilisateur.

1.3 Réalisation

Nous distinguons deux grandes approches pour réaliser une application d'informatique diffuse : l'approche physique et l'approche virtuelle. Dans cette partie nous introduisons ces deux approches.

1.3.1 Approche physique

L'approche physique extrait directement depuis l'environnement physique les informations nécessaires à l'application. Pour cela, elle embarque directement des données sur les objets physiques qui participent à l'application et recourt à un mécanisme de

perception pour détecter les données à proximité. Nous distinguons deux approches pour embarquer des données sur un objet physique : avec une étiquette électronique ; avec un calculateur embarqué. Une étiquette électronique est un dispositif électronique contenant des données et lisible à distance grâce à un lecteur d'étiquettes dédié. Le terminal de l'utilisateur peut accéder aux étiquettes qui sont incluses dans la zone de lecture de son lecteur. Lorsque les objets physiques embarquent un calculateur sans fil, le terminal de l'utilisateur est lui aussi équipé d'une interface de communication à courte portée. Le terminal accède aux données stockées dans les calculateurs inclus dans sa zone de communication.

Nous considérons que l'approche physique repose sur un mode d'adressage physique. Le terminal utilisateur accède aux données incluses dans sa zone d'adressage. La zone d'adressage du terminal corresponds soit à sa zone de communication, soit à sa zone de lecture dans le cas d'étiquettes électroniques.

Pour réaliser un guide virtuel avec l'approche physique, nous embarquons sur chaque œuvre les informations qui lui sont relatives. Au cours de la visite, le terminal de l'utilisateur propose à l'utilisateur les informations qu'il découvre dans sa zone d'adressage.

1.3.2 Approche virtuelle

L'approche virtuelle couple les applications à l'environnement physique grâce à un modèle logique du monde physique ; elle n'embarque ni étiquette ni calculateurs sur les objets physiques. Ce modèle représente l'état du monde physique et inclus notamment la position des utilisateurs. Il doit être régulièrement mis à jour en fonction des changements apportés au monde physique. L'application analyse régulièrement le modèle et réagit en fonction des modifications apportées au modèle, telles que des changements de position.

Pour conserver la cohérence entre le modèle et le monde physique, l'approche virtuelle utilise un système de localisation afin de surveiller la position des objets mobiles et des utilisateurs. L'approche virtuelle repose sur un serveur distant qui stocke le modèle du monde physique et délivre les services aux utilisateurs. Pour réaliser un guide virtuel pour musée avec cette approche [34] le terminal de l'utilisateur doit transmettre régulièrement sa position au serveur. Le serveur renvoie à l'utilisateur des informations en fonction de sa position. Le serveur est relié aux terminaux utilisateurs par un réseau global sans fil.

1.4 Bilan

Dans ce chapitre nous avons introduit d'informatique diffuse. Une application d'informatique diffuse est utilisée dans un contexte où l'utilisateur a déjà une activité, elle doit donc fonctionner implicitement pour ne pas détourner son attention. Nous avons présenté les applications contextuelles qui sont un type d'application diffuse très important. Une application contextuelle s'adapte automatiquement à la situation de l'utilisateur.

Dans la dernière partie de ce chapitre, nous avons présenté deux grandes approches pour réaliser une application d'informatique diffuse : l'approche physique et l'approche virtuelle. Dans cette thèse nous nous intéressons à l'approche physique et particulièrement à la programmation d'applications physiques. La suite de cet état de l'art est donc dédiée à l'approche physique. L'approche virtuelle est détaillée dans ce document de synthèse [43]. Nous avons vu que l'approche physique embarque sur les objets physiques des données, soit via des calculateurs embarqués, soit via des étiquettes électroniques. Dans le chapitre 2 nous présentons les système physiques qui reposent sur des calculateurs embarqués et dans le chapitre 3 nous présentons les systèmes qui reposent sur les étiquettes électroniques.

Chapitre 2

Applications purement diffuses

Dans cette partie, nous présentons les systèmes physiques qui reposent sur des calculateurs embarqués. Ces systèmes sont qualifiés de purement diffus car ils sont complètement intégrés dans l'espace physiques et ne nécessitent ni infrastructure de communication ni serveur d'exécution. Dans ce chapitre, nous introduisons tout d'abord les systèmes purement diffus et présentons ensuite leur architecture. La suite de ce chapitre est dédiée à la présentation de plusieurs systèmes supportant la création d'applications purement diffuses.

2.1 Présentation

Dans la suite de cette thèse nous appelons entité physique tout élément de l'espace physique qui participe à une application d'informatique diffuse. Une entité physique peut donc être un lieu, une objet, une personne, un bâtiment...

Une application purement diffuse est couplée au monde physique en embarquant sur les entités un calculateur équipé d'une interface de communication à courte portée. Les entités avec lesquelles une entité peut communiquer sont appelées entités voisines. Les données associées à une entité sont stockées dans son calculateur. Ainsi, les données se déplacent avec les entités. De cette manière, l'organisation physique des entités représente l'organisation physique des données. L'espace physique est donc utilisé à la fois pour organiser et stocker les données. Il devient un système d'information spatial [9, 10].

Une application purement diffuse est par essence distribuée. Ainsi, chaque entité exécute un programme et interagit avec les autres entités. Les entités interagissent entre elles par l'échange de données via des communications sans fil de proximité. Ces données peuvent être aussi bien des données destinées à l'utilisateur comme pour un guide virtuel, ou des données de type "jeton" servant à la synchronisation des programmes exécutés par les entités. Les entités ne peuvent communiquer qu'avec les entités voisines, ainsi la découverte des données et les interactions entre entités sont fonction de leurs mouvements.

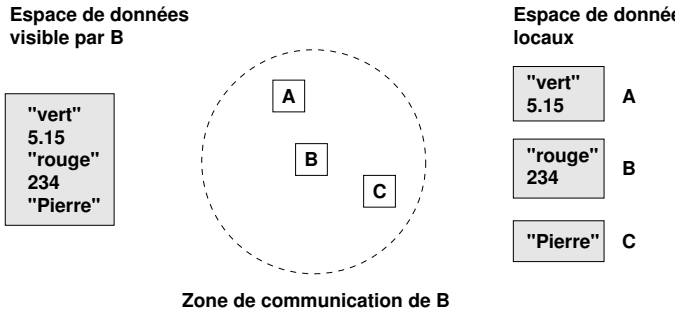

FIG. 2.1 – Un espace de données partagées physiquement

2.2 Architecture

Dans cette partie nous présentons l'architecture logicielle et matérielle d'une application purement diffuse, ainsi que le modèle de programmation associé.

2.2.1 Architecture logicielle

Les systèmes présentés dans ce chapitre permettent aux entités d'échanger des données par le biais d'un mécanisme de partage de données. Le principe de base est d'offrir à chaque entité la vision d'un espace de données contenant les données partagées par les entités voisines. La figure 2.1 illustre ceci. Nous avons trois entités A, B et C. Dans un souci de lisibilité nous ne représentons que la zone de communication B. L'espace de données visible depuis l'entité B est constitué de l'union des données partagées par les trois calculateurs. Nous sommes bien ici en présence d'un mode d'adressage physique : les données accessibles par une entité sont celles qui sont incluses dans sa zone de communication.

Du point de vue de l'espace visible d'une entité, les mouvements des entités sont reflétés par l'apparition et la disparition de données de l'espace visible. L'apparition d'une donnée dans son espace visible correspond à l'arrivée d'une autre entité dans sa zone de communication. La disparition de cette même donnée correspond au départ de cette entité.

Une fois les données partagées, les programmes y accèdent par le biais de différentes opérations. Ces opérations sont fonction du modèle de partage de données. Par exemple, si les données sont stockées dans un espace de tuples, les opérations disponibles sont similaires aux opérations fournies par Linda [17]. Si le système de partage de données est basé sur une base de données, alors les données peuvent être manipulées à l'aide du langage de requêtes SQL.

Pour réaliser un guide virtuel avec cette approche, chaque œuvre partage les informations qui la concernent. L'espace de données visible par le guide est composé de l'ensemble des informations partagées par les œuvres voisines. A partir du moment où la portée de communication du guide est suffisamment réduite, le guide n'affiche que des informations concernant des œuvres proches du visiteur (cf. figure 2.2).

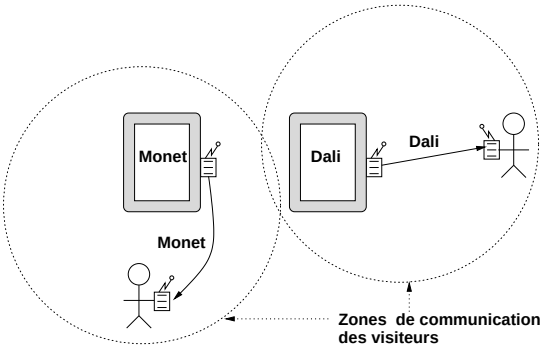

FIG. 2.2 – Un guide virtuel purement diffus

2.2.2 Modèle de programmation

Avec une application purement diffuse, chaque entité qui participe à l'application embarque les données qui lui sont associées, via un calculateur sans fil. Lorsque le programmeur manipule les données détectées dans l'environnement, nous considérons qu'il manipule implicitement les entités associées à ces données.

Le mécanisme de partage de données entre entités permet au programmeur de synchroniser une application sur la rencontre d'une entité. Pour cela, il synchronise simplement l'application sur l'apparition dans son espace visible d'une donnée associée à cette entité. Ce modèle de programmation permet au programmeur de penser directement en terme de déplacements d'entités et de leurs rencontres.

Le mécanisme de synchronisation d'une application sur la découverte d'une donnée est analogue au mécanisme de synchronisation par sémaphore dans un système parallèle. Un sémaphore est une primitive de synchronisation permettant de définir un accès exclusif à une ressource. Lorsqu'un processus P_1 désire accéder à une ressource, il reste bloqué jusqu'à ce qu'elle soit libérée par le processus P_2 qui la mobilise. Le processus P_1 est donc synchronisé sur la libération de la ressource. Dans une application purement diffuse nous pouvons synchroniser le programme exécuté par une entité E_1 sur la découverte d'une donnée portée par une entité E_2. Si E_2 est fixe, sa position géographique représente un point de synchronisation spatial pour E_1. Ce point de synchronisation est l'équivalent physique de l'instant auquel P_2 libère la ressource.

2.2.3 Architecture matérielle

Chaque entité doit embarquer un calculateur. En fonction de la taille de l'entité et de ses besoins en capacité de stockage plusieurs solutions techniques sont disponibles. Nous distinguons deux grandes catégories de calculateurs : les PDA et téléphones mobiles ; les capteurs intelligents. Les PDA et téléphones mobiles peuvent contenir une grande quantité d'informations. Cependant leur taille et leur prix les réservent à une utilisation en tant que terminaux utilisateur. Les capteurs intelligents sont des micro-calculateurs équipés d'une interface de communication à courte porté et de différents

capteurs. Les capteurs intelligents sont moins encombrants et meilleur marché que les téléphones mobiles, mais leur capacité de calcul et de stockage sont inférieures. Les capteurs intelligents peuvent être embarqués sur les objets physiques.

Parmi les différentes architectures de capteurs qui sont actuellement développées, nous trouvons : Smart Dust [29], les Motes [26], les Smart Its [27] et PicoRadio [48]. Smart Dust est le projet le plus ambitieux. Il a pour objectif de fabriquer des capteurs qui auront une taille de l'ordre du grain de poussière. Actuellement, la taille correspond à un cube de deux à trois millimètres de coté incluant une unité de calcul, une interface communication laser, une unité de traitement du signal, une batterie et des capteurs solaires. Les Motes [26] ont été développées parallèlement à Smart Dust. Elles sont plus encombrantes que les nœuds Smart Dust, mais présentent l'avantage d'être d'ores et déjà commercialisées. Elles ont notamment été utilisées dans [19] pour des expérimentations sur la localisation de capteurs. Les Smart Its proposent une architecture proche des Motes. PicoRadio a pour objectif de fournir des capteurs consommant un minimum d'énergie et se concentre sur l'aspect communication.

Le tableau 2.1 présente les caractéristiques principales de ces différentes technologies. Nous les classons en fonction de leur capacité mémoire, leur source d'énergie, leurs moyens de perception de l'environnement (moyen de communication) et de leur utilisation type.

Dispositif	Mémoire	Énergie	Perception	Portée	Utilisation
Téléphones	2 à 8 Mo	Batterie	Bluetooth, IRDA	10m, 1-5m	Terminal utilisateur
PDA	64 Mo	Batterie	Bluetooth, WiFi	10m, 30-300m	Terminal utilisateur
Motes	512 ko	Batterie	Radio	20m	Embarqués sur les entités
Smart Dust	8 ko	Solaire	Laser	>100m	Embarqués sur les entités

TAB. 2.1 – Comparaison des solutions matérielles pour les systèmes physiques

2.3 Partage de données basé sur un espace de tuples

Dans la partie précédente, nous avons vu que les applications purement diffuses sont basées sur un partage d'information entre entités. Nous présentons maintenant deux systèmes exploitant un espace de tuples pour le partage des données. Nous présentons également dans la suite de cette partie (cf. 2.4) un système reposant sur une structure arborescente et un système utilisant un mécanisme de base de données partagée (cf. 2.5).

2.3.1 Principes de base

Les systèmes présentés dans cette partie utilisent un espace de tuples pour le partage des données[17]. Un tuple est une suite ordonnée d'éléments typés : entiers, flottants et chaînes de caractères. Par exemple : `<10,"Voiture",12.5>`. Initialement, un espace de tuples est destiné aux systèmes distribués, pour lesquels les nœuds sont toujours connectés.

Dans le cadre d'un scénario avec des entités mobiles qui ne sont pas connectées en permanence, nous ne pouvons plus nous reposer sur un espace de tuples unique et accessible en permanence par toutes les entités. Les systèmes présentés dans cette partie font appel à un espace de tuples partagé. Ainsi, chaque entité embarque son propre espace de tuples qu'elle partage avec les entités voisines. L'espace de tuples visible par une entité à un moment donné est constitué de l'union des espaces de tuples accessibles.

Les opérations disponibles sont la lecture, le retrait et l'ajout d'un tuple. Les opérations de lecture et de retraits prennent pour paramètre un motif, un motif étant un tuple dans lequel zéro ou plusieurs champs n'ont que leur type de défini. Par exemple : `<10, string, 12.5>`. Il y a correspondance entre un tuple et un motif lorsqu'ils ont le même nombre d'éléments, les types de chaque élément se correspondent deux à deux et les valeurs des champs sont égales deux à deux. Ce mode de communication est dit anonyme, en effet les entités ne communiquent que par l'intermédiaire des tuples. Lorsque les entités lisent ou retirent un tuple, elles ne peuvent en aucun cas connaître l'identité de l'entité qui l'a publié.

Du point de vue contextuel, une application reposant sur un espace de tuples représente le contexte de l'utilisateur par un sous-ensemble de l'espace de tuples visible. Ce sous-ensemble est représenté par l'ensemble des tuples qui correspondent à un motif donné. Par exemple, si nous voulons représenter le contexte de l'utilisateur par l'ensemble des descriptions de restaurants situés à proximité, nous pouvons utiliser le motif `<"restaurant", string description>`. Les opérations de lecture sur un espace de tuples restent bloquées jusqu'à l'apparition d'un tuple satisfaisant la requête. Il est donc possible de synchroniser l'application sur la détection d'un contexte particulier.

2.3.2 Lime : un middleware dédié à la mobilité

LIME [45] est un middleware développé pour aider à la programmation d'applications distribuées incluant des nœuds fixes et mobiles. Il est fondé sur la notion d'agents mobiles et d'espace de tuples partagé. De notre coté, nous nous intéressons au cas des nœuds mobiles et considérons qu'ils sont équipés d'une interface réseau sans fil.

Un agent mobile est un programme capable d'arrêter son exécution, de migrer sur un autre nœud via le réseau et finalement d'y reprendre son exécution. Il est autonome dans le sens où la décision de migrer lui appartient. Les raisons d'une migration sont diverses, cependant, la plupart du temps un agent migre parce qu'il a besoin d'une ressource ou de données présentes sur un autre nœud. Pour leurs communications, LIME fournit aux agents un mécanisme d'espace de tuples partagé. Ainsi, chacun d'eux possède un espace de tuples qui se déplace avec lui, l'espace de tuples visible est constitué de l'union des

espaces de tuples des agents connectés. Un agent est considéré comme connecté lorsque son nœud hôte l'est.

Aux opérations sur les tuples classiques, LIME rajoute un mécanisme de réactions, une réaction étant composée d'un motif et d'un fragment de code. Lorsqu'un tuple correspondant au motif est ajouté dans l'espace de tuples, le fragment de code est exécuté et parallèle du flot d'exécution principal.

Parmi les applications réalisables avec LIME, les auteurs présentent notamment un jeu consistant en une chasse au drapeau faite par deux équipes [46]. L'application propose un service de navigation contextuelle qui affiche en permanence une vue de l'environnement incluant les objets proches ainsi que la position des coéquipiers.

2.3.3 SPREAD

Dans cette partie nous présentons SPREAD [8] qui est un environnement dédié à la programmation d'applications purement diffuses.

2.3.3.1 Présentation générale

SPREAD propose de transformer l'espace physique en une mémoire. Ainsi, chaque entité embarque les données qui lui sont associées. De plus, chaque donnée est associée à un volume qui définit la zone dans laquelle la donnée est accessible. Ce volume se déplace avec l'objet portant la donnée. Bien que le modèle spécifie que le volume associé à une donnée puisse être quelconque, SPREAD ne permet que de définir des sphères centrées sur l'entité.

Comme LIME, SPREAD met en œuvre un mécanisme d'espace de tuples partagé. Chaque entité embarque un espace de tuples local et son espace de tuples visible correspond à l'union des espaces de tuples des entités à portée de communication.

L'objectif de SPREAD est de construire des applications exploitant directement les interactions physiques entre entités. Par exemple, considérons une application aidant un piéton à trouver un taxi. Pour trouver un taxi, le terminal du piéton publie un tuple signifiant que l'utilisateur souhaite prendre un taxi. Un taxi recherche un piéton avec une opération de type `read<“piéton”>`. Cette opération reste bloquée jusqu'à ce que le taxi soit à l'intérieur du volume d'un piéton. Ainsi, les mouvements physiques des entités physiques contrôlent directement le déroulement de l'application. Ce modèle de programmation est appelé programmation spatiale.

SPREAD ne propose que des opérations de lecture et d'ajout de tuples. Seule l'entité qui a créé un tuple peut le retirer. SPREAD choisit d'utiliser un espace de tuples parce qu'il permet de partager simplement les données entre entités et de synchroniser les programmes sur les déplacements d'entités.

SPREAD permet bien sûr de programmer des applications contextuelles. Le contexte de l'utilisateur est représenté par l'ensemble des tuples accessibles.

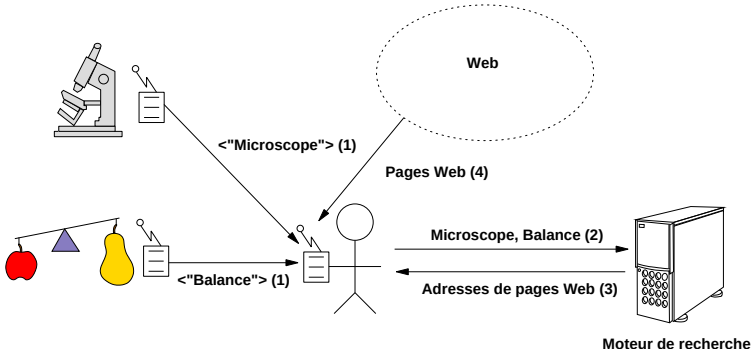

FIG. 2.3 – Capture du contexte dans WebWalker

2.3.3.2 Applications

SPREAD est dédié aux applications physiques. WebWalker [9], un guide pour expositions a notamment été réalisé. WebWalker associe chaque objet exposé à des tuples contenant soit un mot clé le décrivant soit l'adresse d'une page Web (URL) le concernant.

WebWalker représente le contexte de l'utilisateur par l'ensemble des URL et mots clés l'entourant. Les URL lui permettent d'accéder à de plus amples informations sur les objets proches. De leur côté, les mots clés sont utilisés pour effectuer une recherche sur le Web. Grâce aux pages trouvées, l'utilisateur peut également accéder à des informations sur son environnement physique proche.

WebWalker combine à la fois navigation physique et navigation virtuelle. La navigation physique permet à l'utilisateur de découvrir des pages Web par ses déplacements physiques. Depuis les pages trouvées physiquement, l'utilisateur peut continuer sa navigation virtuellement sur le Web, grâce au mécanisme de navigation Web habituel.

La figure 2.3 présente cette application. Les mots sont distribués sur les stands d'exposition via un calculateur sans fil qui peut communiquer avec le guide du visiteur. Le guide détecte tout d'abord les mots, puis fait une requête vers le moteur de recherche et utilise le résultat pour charger les pages Web correspondantes. Par souci de clarté, la figure ne représente pas les URL directement associées aux objets exposés.

2.4 Partage de données basé sur une structure arborescente

PeerWare [11] est un système physique permettant aux entités de partager leurs données via une structure particulière appelée GVDS pour : *Global Virtual Data Structures*. Le principe de partage des données dans un GVDS est similaire au principe d'espace de tuples partagé (cf. 2.2.1) : une entité à la vision d'un espace de données global correspondant à l'union des espaces de données des entités voisines. Un GVDS ne définit pas la structure des données ni les opérations pour y accéder, c'est à l'application de le

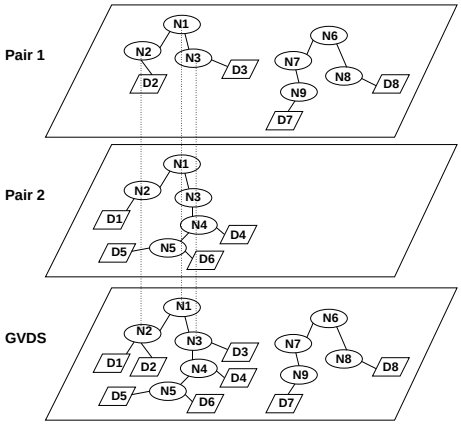

FIG. 2.4 – Partage des informations dans PeerWare

faire.

PeerWare fait le choix de représenter les données par une forêt d'arbres ayant chacun une racine distincte. Les feuilles des arbres sont des documents et correspondent aux données partagées par les entités. Lorsque plusieurs entités sont connectées, les nœuds qui portent le même nom et occupent la même place dans l'arbre sont représentés par un seul nœud dans la structure globale (cf figure 2.4). Avec une telle structure, la sélection d'une donnée ne se fait plus par l'intermédiaire de sa signature comme pour les tuples, mais via sa place dans l'arborescence. Un programme peut par exemple modifier l'ensemble des documents attachés à un nœud, la position du nœud étant passée en paramètre. Cette organisation est similaire à l'organisation d'un système de fichiers, à la différence près qu'un système de fichiers a une racine unique.

PeerWare propose des opérations de création, modification et effacement de nœuds ou documents ainsi qu'un système de réaction comme LIME.

Une des applications envisagées par PeerWare est l'aide aux secouristes en cas de catastrophe naturelle. Dans ce cas là, l'application ne peut pas se reposer sur une infrastructure globale. Le scénario d'un ensemble de nœuds mobiles communicant en mode pair à pair prend alors tout son sens. PeerWare permet aux sauveteurs de partager des informations concernant le lieu d'opération ainsi que d'envoyer et recevoir des notifications à propos d'événements appropriés.

2.5 Partage de données basé sur des bases de données

PERSEND est un système qui permet aux entités de partager leurs données via une base de données [60]. Pour cela, chaque entité embarque une base de données. Lorsqu'elle fait une requête, celle-ci est envoyée aussi bien à la base de données locale qu'aux bases de données des entités voisines. Une entité a donc pour vision une base de données globale composée des bases de données des entités voisines.

PERSEND a pour objectif la prise en compte de la volatilité des données et de leur

aspect dynamique dans les applications purement diffuses. Avec un système de base de données classique, le résultat d'une requête est calculé d'après le contenu de la base au moment de la requête. Si l'on veut suivre l'évolution des données en fonction des mouvements des entités, ce système impose de faire des requêtes régulièrement. Les requêtes continues proposent une solution élégante à ce type de problèmes. Une requête continue n'est pas exécutée une fois pour toute par la base de données, mais reste active après le calcul du résultat pour être réévaluée à chaque ajout ou suppression d'enregistrements. Ainsi, le résultat d'un requête évolue dynamiquement au fur et à mesure des opérations sur la base et également selon les départs et arrivées d'entités, un départ ou une arrivée se traduisant par l'ajout ou la suppression des enregistrements provenant de l'entité correspondante.

2.6 Synthèse

Dans cette partie nous avons présenté différentes architectures, systèmes et applications purement diffuses. Nous avons tout d'abord détaillé deux environnements de programmation reposant sur l'utilisation d'un espace de tuples : LIME et SPREAD. PeerWare propose un modèle de partage de données basé sur une arborescence et PERSEND stocke les données dans des bases de données.

Les systèmes purement diffus sont adaptés à la conception d'applications contextuelles. D'une manière générale, le contexte de l'utilisateur est représenté par l'ensemble des données accessibles. Avec SPREAD et LIME cet ensemble est constitué de tuples. Pour PeerWare les données accessibles sont représentées par une série d'arbres (cf. figure 2.4). Pour PERSEND, le contexte est représenté par l'ensemble des bases de données embarquées sur les entités voisines.

Plusieurs technologies sont disponibles pour déployer des applications purement diffuses. Les capteurs intelligents sont adaptés à la dissémination de données sur les objets physiques. Les PDA et téléphones mobiles sont quant à eux utilisés comme terminaux utilisateur.

Chapitre 3

Marquage électronique

Dans la partie précédente, nous avons présenté des applications purement diffuses qui embarquent sur chaque entité un calculateur sans fil. Dans cette partie, nous présentons des applications qui exploitent le marquage électronique.

Nous commençons par présenter le principe de base d'une application de marquage électronique. Nous poursuivons en décrivant leur architecture et terminons par le détail de plusieurs applications exploitant les différentes technologies existantes.

3.1 Présentation

Le principe du marquage électronique couple les applications au monde physique en embarquant sur les entités physiques des étiquettes détectables à distance. Trois technologies sont disponibles : i) les étiquettes radios (RFID) [54, 33, 64] ; ii) les codes barres [12, 50, 51] ; iii) les étiquette Web [31, 30].

Une fois embarquées sur les objets, les étiquettes sont associées à des données, qui sont en relation avec l'objet hôte. Nous avons ainsi une auto description de l'environnement. Les données décrivant l'environnement physique sont directement accessibles depuis celui-ci.

Une application de marquage électronique perçoit son environnement grâce à un lecteur d'étiquettes qui lui permet de détecter et lire les étiquettes proches du lecteur. L'application propose ensuite un service qui est fonction des étiquettes détectées.

Pour réaliser un guide virtuel à l'aide d'étiquettes électroniques, il suffit d'ajouter une étiquette dans chaque œuvre et d'équiper le terminal utilisateur d'un lecteur d'étiquettes. Une table associant étiquettes et informations est stockée dans le terminal. Le contexte de l'utilisateur est représenté par les étiquettes lues par le lecteur de son terminal. L'application est donc un cycle de détections d'étiquettes et d'affichages des informations correspondantes.

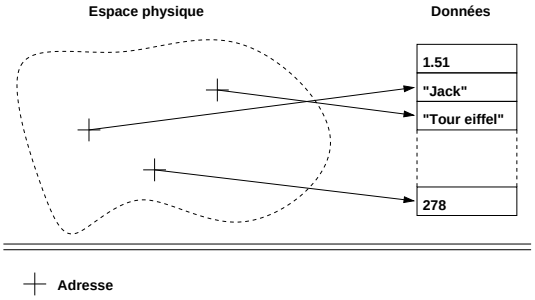

FIG. 3.1 – Adressage physique basé sur des étiquettes électroniques. Les adresses sont distribuées dans l'espace physique et permettent d'accéder indirectement aux données

3.2 Architecture

L'architecture d'une application de marquage électronique repose principalement sur un serveur associant données et étiquettes. Ce serveur est interrogé régulièrement en fonction des étiquettes détectées par le lecteur. Dans cette partie nous détaillons cette architecture du point de vue logiciel et matériel. Nous présentons également un modèle de programmation exploitant cette architecture.

3.2.1 Architecture logicielle

Une application basée sur le marquage électronique associe chaque étiquette à des données. Ces données sont le plus souvent stockées dans un serveur de données distant. Ce serveur est découplé du monde physique. Nous parlons alors de stockage virtuel des données : les données ne sont pas directement stockées sur les entités.

Une application de marquage électronique repose sur un mode d'adressage physique. La zone d'adressage correspond à la portée de lecture de l'entité. Par exemple, pour déterminer quelles sont les données proches de l'utilisateur, il suffit de détecter les étiquettes proches et de récupérer les données associées. Une étiquette représente donc l'adresse de la donnée correspondante. La figure 3.1 illustre ce mode d'adressage physique combiné au stockage virtuel.

Les applications de marquage électronique ne reposent pas sur des échanges de données entre entité. Le plus souvent le terminal utilisateur capture les étiquettes à proximité et les envoie au serveur qui lui renvoie les données correspondantes. Ces applications suivent le schéma de fonctionnement décrit sur la figure 3.2.

3.2.2 Modèle de programmation

Lorsqu'il développe une application de marquage électronique, la tâche du programmeur consiste principalement à définir le service en fonction des étiquettes détectées. Römer et Schoch [54] ont proposé un modèle de programmation événementiel pour des applications basées sur des RFID. Les deux évènements considérés par ce modèle de

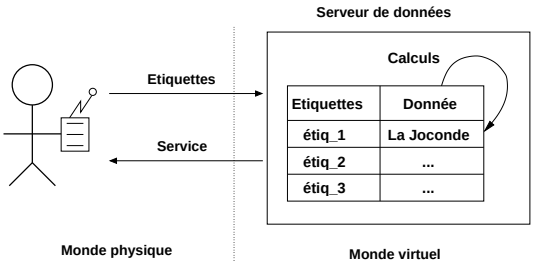

FIG. 3.2 – Approche basée sur le marquage électronique

programmation sont l'apparition et la disparition d'une étiquette.

Dans une application de marquage électronique chaque entité est associée à une étiquette. Ainsi, détecter un objet revient à détecter l'étiquette correspondante. Nous considérons que, lorsqu'il manipule des étiquettes dans son programme, le programmeur manipule les entités qui leurs sont associées. Römer et Schoch présentent cette notion d'équivalence numérique dans leur article [54], ils la baptisent *digital counterpart*. Ainsi l'apparition et la disparition d'une étiquette de la zone de lecture sont respectivement équivalentes à l'apparition et la disparition de l'entité correspondante.

3.2.3 Support technologique

Nous présentons maintenant les trois catégories d'étiquettes : i) les RFID ; ii) les codes barres ; iii) les étiquettes Web.

3.2.3.1 RFID

Les étiquettes RFID sont conçues pour être ajoutées aux objets de la vie courante, tout comme les classiques codes barres. Leur lecture est faite par ondes radio et donc, contrairement aux codes barres, une ligne de vue n'est pas nécessaire pour les détecter. Ainsi, ils peuvent être rendus invisibles en les intégrant directement dans les objets. La distance maximum de lecture varie de quelques centimètres à plusieurs mètres selon le type d'étiquette. Les RFID peuvent être soit actifs soit passifs. Les premiers contiennent juste un numéro unique et n'ont pas besoin d'alimentation, tandis que les seconds contiennent une petite zone de mémoire accessible par le lecteur et requièrent une alimentation.

3.2.3.2 Codes barres

Les codes barres sont une autre solution pour réaliser des étiquettes électroniques. Rekimoto et al ont développé deux systèmes. Le plus ancien [51] propose des codes barres réalisés sous la forme d'une succession de bandes verticales rouges et bleues. Dans la seconde approche [50] les codes barres sont représentés par une grille de carrés noirs et blancs et sont baptisés CyberCodes. López, quant à lui, propose des codes

barres circulaires [12] où l'information est encodée avec des secteurs d'anneaux noirs et blancs.

Toutes ces technologies reposent sur l'utilisation d'une caméra et de techniques de traitements d'images pour la reconnaissance des étiquettes. En plus de détecter la référence de l'étiquette, ces traitements permettent également d'obtenir la position relative de l'étiquette par rapport à la caméra, aussi bien avec les codes barres circulaires que les CyberCodes. Notons que toutes ces techniques sont consommatrices de puissance de calcul et impliquent l'utilisation d'une machine puissante pour réaliser les traitements.

Les codes barres présentent certains avantages par rapport aux RFID. Leur coût de fabrication est négligeable, une simple imprimante suffit. De plus, ils permettent d'obtenir la position relative de l'étiquette par rapport à la caméra. Leur inconvénient majeur est la nécessité d'une ligne de vue pour effectuer la lecture. Ceci implique que les codes barres doivent être visibles à la surface des objets et les rend inadaptés pour les situations où l'esthétique prime. Les RFID peuvent être rendus invisibles. En revanche, leur coût est supérieur et il n'est pas possible d'avoir la position de l'étiquette ; l'application sait seulement que l'étiquette est à portée de lecture.

3.2.3.3 Étiquettes Web

L'objectif du projet CoolTown [30, 31] est de fournir une "présence Web" aux entités physiques. Une page Web est donc associée à chaque entité et son adresse (URL) est accessible dans son voisinage physique. Les objets embarquent leur URL soit via un RFID soit via un calculateur embarqué équipé d'une interface de communication infrarouge. Lorsque l'étiquette repose sur des communications infrarouges, l'URL est directement accessible depuis l'espace physique. Dans le cas des RFID, l'application fait appel à un serveur qui renvoie l'URL correspondant au RFID détecté. Une fois l'URL d'un objet détectée, le terminal l'utilise pour récupérer et afficher la page Web correspondante. L'argument en faveur de l'utilisation de la technologie Web est sa simplicité et son fort niveau de déploiement actuel.

Avec CoolTown, le serveur de données correspond au serveur Web qui délivre le service. Il reçoit en entrée des requêtes qui sont basées sur le protocole Web (HTTP) et décrivent le service requis.

3.2.4 Infrastructure matérielle

Une application de marquage électronique stocke les données associées aux étiquettes dans un serveur de données découplé du monde physique. Un réseau global reliant le serveur aux terminaux est nécessaire. Pour certaines applications les terminaux utilisateurs font directement office de serveur de données. A ce moment là le réseau de communication global est inutile. Par contre, le problème de la mise à jour des terminaux se pose. En effet, si les données associées aux entités changent, il faut alors mettre à jour les données dans l'ensemble des terminaux.

Une application de marquage électronique repose sur un mode d'adressage physique. Les terminaux détectent les données dans leur environnement proche grâce à un lecteur

d'étiquettes. Dans le cas d'étiquettes de type RFID, un lecteur spécialisé doit être ajouté aux terminaux. Dans le cas de codes barres, nous pouvons exploiter l'appareil photo intégré aux téléphones mobiles récents.

3.3 Applications

Après avoir présenté dans la partie précédente l'architecture d'une application basée sur le marquage électronique, nous présentons maintenant plusieurs de ces applications. Nous commençons avec des applications d'annotation contextuelle. Nous poursuivons avec des applications exploitant la réalité enrichie et terminons avec une application d'assistant cuisinier.

3.3.1 Applications d'annotation contextuelle

Les étiquettes électroniques sont particulièrement adaptées à la création d'applications d'annotation contextuelle. Par exemple, Rekimoto propose d'utiliser son système de codes barres pour les guides virtuels pour musées. Les codes-barres, disposés à coté des œuvres, sont lus par le guide qui diffuse ensuite les informations correspondantes.

Dans le projet CoolTown, les étiquettes Web permettent aux utilisateurs d'accéder aussi bien à des informations décrivant les objets qu'aux services proposés par les objets. Par exemple, lorsqu'un visiteur découvre dans une boutique un livre qui l'intéresse, il sauvegarde la page Web décrivant ce livre. Lorsqu'il passe devant une imprimante, il accède via la page Web de l'imprimante au service d'impression et obtient un tirage papier de la description du livre.

Dans [53] les auteurs présentent un projet de supermarché électronique proposant de nouveaux services à ses clients. Les applications sont principalement des applications d'annotation contextuelle. Ainsi, les produits sont enrichis par des informations numériques telles que leur prix ou bien leurs ingrédients, grâce à des étiquettes de type RFID. Ceci permet à l'application de proposer au client des promotions adaptées au contenu de son chariot. Par exemple, lorsqu'il vient de mettre un produit du rayon informatique dans son chariot, son terminal lui diffuse les promotions faites dans ce rayon susceptibles de l'intéresser. De même, le prix du chariot est calculé et présenté au client en permanence, lui évitant ainsi de le vider lors du passage en caisse. L'application propose également des services en dehors du supermarché. Lorsque le client est chez lui, les appareils équipés d'un lecteur de RFID consultent les produits achetés pour établir une liste de commissions contenant les articles les plus courants. Cette liste peut être utilisée, la fois suivante, pour consulter les supermarchés environnants et trouver la meilleure offre. Du point de vu matériel, cette application est déployée en embarquant un RFID sur l'ensemble des articles, et en équipant les chariots d'un lecteur de RFID. De son côté, le client dispose d'un PDA pour la réception des informations.

3.3.2 Réalité enrichie

Les codes barres permettent de connaître la position et l'orientation relative de l'étiquette par rapport au lecteur. Lorsque le lecteur combine détection et visualisation, il est alors possible de réaliser des applications de réalité enrichie. Par exemple, Rekimoto propose d'associer les étiquettes à des objets tridimensionnels [50]. Par l'intermédiaire d'un dispositif d'affichage ad-hoc, la vue qu'a l'utilisateur de son environnement est constituée de la vue réelle additionnée à des objets tridimensionnels, qui sont affichés là où se trouvent les étiquettes. Dans [62] les auteurs présentent un jeu vidéo également basé sur la réalité enrichie. Le terrain du jeu est constitué d'un circuit de train en bois. Plusieurs étiquettes sont disposées sur le circuit afin de détecter la position du terminal relativement au circuit. Le terminal utilisateur est un PDA équipé d'une caméra. Le jeu rajoute des trains virtuels circulant sur le circuit, l'objectif étant de contrôler les aiguillages pour éviter les collisions.

Dans [52] les auteurs présentent un système qui repose sur l'utilisation d'un téléphone pour la lecture des étiquettes et l'affichage des informations. Le téléphone analyse en permanence le flot d'images capturées par l'appareil photo intégré et superpose à ce flot d'image des informations. Ce système propose également d'interagir avec les objets physiques par l'intermédiaire de gestuelles basées sur les déplacements du téléphone relativement à l'étiquette. Par exemple, lorsque l'utilisateur passe son téléphone devant le code barre d'une publicité, le téléphone lui propose plusieurs options accessibles via différentes gestuelles, telles qu'accéder à de plus amples informations en déplaçant son téléphone vers la droite, ou obtenir l'adresse de la boutique la plus proche en déplaçant son téléphone vers la gauche.

3.3.3 Assistant cuisinier

Dans [54, 33], Römer et Schoch présentent un assistant cuisinier virtuel qui propose des recettes en fonction des aliments posés sur le plan de travail. Cette application est réalisée en ajoutant sur les emballages des aliments une étiquette électronique et en équipant le plan de travail d'un lecteur. Cette application repose sur leur modèle de programmation événementiel, les événements surveillés sont l'apparition et la disparition d'un aliment du plan de travail. Chaque fois que l'un d'eux se produit, la liste des recettes possibles est mise à jour.

3.4 Comparaison avec les applications purement diffuses

Dans la partie précédente nous avons présenté les applications purement diffuses. La principale différence entre une application purement diffuse et une application de marquage électronique vient de leur mode de stockage des données. Une application purement diffuse repose sur un mode de stockage physique : les données sont directement stockées sur les entités. Une application de marquage électronique repose sur un stockage virtuel : les données sont stockées dans un serveur. Ce serveur pose le problème du passage à l'échelle. En effet, il fournit les données à l'ensemble des utilisateurs.

Lorsque le nombre d'utilisateurs augmente, le serveur de données devient donc un goulot d'étranglement potentiel. Un stockage virtuel nécessite également un réseau global afin de relier le serveur de données aux utilisateurs. Avec un stockage physique le problème du passage à l'échelle ne se pose pas, car les données sont embarquées sur les entités. Les requêtes de données ne sont pas toutes envoyées vers un même serveur mais sont physiquement distribuées sur les différentes entités.

3.5 Synthèse

Dans cette partie, nous avons présenté des applications de marquage électronique. Le marquage électronique ajoute aux entités une étiquette électronique. Dans un serveur de données ces étiquettes sont associées à des données. Une entité détecte les étiquettes grâce à un lecteur, dont la portée de détection est limitée.

Les applications de marquage électronique reposent sur un mode d'adressage physique. La portée de lecture de l'entité représente sa zone d'adressage. Une application de marquage électronique repose sur un mode de stockage virtuel, car les données associées aux étiquettes sont stockées dans un serveur découplé du monde physique.

Les applications sont variées. Nous avons notamment présenté des applications de réalité enrichie, d'annotation contextuelle ou encore d'assistant cuisinier. Une application de marquage électronique scrute en permanence l'environnement à la recherche d'étiquettes et délivre des services en fonction des étiquettes détectées.

Pour réaliser ces applications deux technologies principales sont disponibles : les codes barres et les RFID. Chacune d'elles présente des avantages et des inconvénients. Les codes barres nécessitent une ligne de vue pour la lecture, mais sont économiques. Une simple imprimante suffit pour les créer. De plus, le lecteur de codes barres peut obtenir la position et l'orientation du code barre par rapport à sa propre position. Les codes barres permettent ainsi de réaliser des applications de réalité enrichie à moindre coût, car basées sur des terminaux standards et des étiquettes en papier. Contrairement aux codes barres, les RFID peuvent être rendus invisibles en les intégrant à l'intérieur des objets, mais leur coût est supérieur et ils ne permettent pas de localiser ni d'orienter précisément l'étiquette par rapport au lecteur.

Chapitre 4

Contribution

Dans ce chapitre nous introduisons notre contribution. Celle-ci est structurée autour du rôle de la géométrie en informatique diffuse. Lorsque nous parlons de géométrie nous entendons arrangements géométriques d'entités. Par exemple, si nous considérons les deux entités physiques A et B nous pouvons avoir les arrangements géométriques suivants : A est à l'intérieur de B, A est à droite de B, A est près de B... Nous entendons géométrie également au sens de mouvements d'entités, autrement dit au sens de modifications des arrangements géométriques. Par exemple : A rencontre B, A arrive à l'abri bus, A entre dans la chambre...

4.1 Propriétés géométriques du monde physique

En observant notre environnement physique, nous remarquons que celui-ci est souvent structuré selon des propriétés géométriques. Dans une bibliothèque les livres sont rangés par ordre alphabétique, le plus souvent de *gauche à droite* et de *haut en bas*. Cette structure est relative : un livre est rangé immédiatement à droite du livre qui le précède alphabétiquement. Un parking structuré en *grille*. Pour retrouver son véhicule, un automobiliste doit retenir la *colonne* dans laquelle se trouve son véhicule ainsi que sa *position* dans cette colonne. Dans une rue les bâtiments sont numérotés dans l'ordre croissant en commençant à une des extrémités de la rue.

La géométrie intervient également dans le fonctionnement de notre environnement. Par exemple, à l'intersection de deux rues, c'est l'organisation géométrique des véhicules qui détermine quel véhicule doit passer en premier. De même, dans une file d'attente la personne située *devant* moi sera servie avant. Le fonctionnement d'un réseau d'autobus dépend également de propriétés géométriques : pour prendre le bus, un piéton doit se rendre à l'abri bus et c'est la *proximité* du piéton et de l'abri bus qui provoquera l'arrêt du bus. Ainsi, l'organisation géométrique de l'environnement physique a un sens et nous interagissons avec lui en fonction de cette organisation.

4.2 Géométrie et informatique diffuse

Une application d'informatique diffuse est liée à l'environnement physique et dépend des mouvements des entités physiques, ainsi que de leur organisation géométrique. Ainsi, les propriétés géométriques qui organisent et régissent le fonctionnement de l'environnement physique interviennent également dans le fonctionnement des applications. Par exemple, dans un musée un guide virtuel présente à l'utilisateur les informations relatives à l'œuvre qui se trouve *devant* lui. Dans un supermarché le prix du chariot est représenté par la somme des prix des produits qu'il *contient*.

Le rôle de la géométrie en informatique diffuse a été introduit dans [3], dans le cadre d'une application d'espace intelligent [4]. Dans cet article, les auteurs démontrent la nécessité de connaître la configuration géométrique de l'espace physique, au-delà la simple notion de proximité. Les auteurs s'appuient sur un exemple d'application qui notifie l'utilisateur de l'arrivée d'un courriel, sur l'écran le plus approprié. Pour cette application, l'écran le plus proche n'est pas toujours le plus approprié. Par exemple, si l'écran le plus proche se trouve dans le dos de l'utilisateur, l'utilisateur ne verra pas la notification.

Dans cette thèse, nous étudions tout d'abord le rôle des propriétés géométriques, qui structurent et contrôlent l'espace physique, dans la programmation des applications purement diffuses. Dans le chapitre 2, nous avons vu que l'approche purement diffuse permet au programmeur de synchroniser ses programmes sur des rencontres entre entités. Cependant de nombreuses applications ne reposent pas uniquement sur ce type de déplacement. Dans cette thèse, nous proposons un modèle de programmation qui permet non seulement de synchroniser les programmes sur les rencontres entre entités, mais aussi sur les départs et arrivées d'entités dans une zone de l'espace physique. Afin d'assurer la cohérence entre l'état des applications et l'état du monde physique, notre modèle de programmation permet également au programmeur de décrire simplement les configurations géométriques dans lesquelles ces synchronisations doivent avoir lieu.

Nous nous reposons également sur la géométrie pour fiabiliser les transferts atomiques de données entre deux entités physiques. Nous proposons un protocole qui repose sur une contrainte géométrique, afin de minimiser le nombre d'opérations non-atomiques. Cette contrainte définit une distance seuil au-delà de laquelle deux entités ne peuvent plus démarrer un transfert.

Nous proposons également un mécanisme qui permet de construire simplement un système d'information dont l'organisation est analogue à l'organisation géométrique de l'environnement physique. Au sein de ce système d'information, nous verrons qu'il est possible de déterminer le contexte de l'utilisateur en fonction de la notion de proximité. Par exemple, l'espace physique est un système d'information composé des informations portées par les entités physiques. Dans ce système d'information, le contexte de l'utilisateur est représenté par les données portées par les entités proches de lui. Nous étudions également l'utilisation du contexte de l'utilisateur pour la navigation dans le système d'information.

Deuxième partie

Rôle de la géométrie dans la programmation des applications

Chapitre 5

Calcul dans l'espace physique

Dans la partie précédente nous avons présenté l'informatique diffuse. Nous avons également présenté deux approches différentes pour réaliser des applications : l'approche purement diffuse et le marquage électronique. Dans cette seconde partie nous étudions le rôle de la géométrie au niveau de la programmation des applications purement diffuses.

Dans ce chapitre nous présentons la programmation spatiale qui est un modèle de programmation dédié à la création d'applications purement diffuses. La programmation spatiale repose sur les mouvements et l'organisation géométrique des entités physiques pour simplifier la programmation d'applications purement diffuses. La programmation spatiale a été introduite par Couderc et Banâtre dans [8, 9]. Nous présentons tout d'abord les principes de la programmation spatiale. Nous présentons ensuite l'application de la programmation spatiale à la création d'applications contextuelles. Finalement, nous présentons les limitations de la programmation spatiale.

5.1 Programmation spatiale

Une application spatiale est constituée d'une série d'interactions physiques. Nous appelons interaction physique tout déplacement d'entité qui modifie l'état de l'espace physique. Par exemple, considérons une application qui calcule le prix d'un chariot dans un supermarché. Le contenu de ce chariot représente le prix total à payer à la fin des commissions. Les insertions et retraits de produits changent l'état de l'éspace physique et modifient donc le prix total du chariot : l'insertion d'un produit correspond à une addition et le retrait à une soustraction. Ici, les interactions physiques qui composent l'application sont donc les insertions et retraits de produits.

Pour réaliser une application spatiale, nous devons tout d'abord identifier les interactions physiques qui composent cette application. Ensuite, chaque interaction physique est associée à un traitement informatique ; l'insertion d'un produit est associée à `prix_total += prix_article` et le retrait à `prix_total -= prix_article`. Finalement, l'application est un cycle de détections d'interactions physiques et d'exécutions des traitements informatiques correspondant : les traitements informatiques sont synchronisés sur les interactions physiques.

Pour synchroniser des traitements informatiques sur des interactions physiques, la programmation spatiale repose sur les mécanismes de mémoire spatiale et de synchronisation physique que nous détaillons dans les deux parties suivantes. Dans la troisième partie nous illustrons la programmation spatiale à l'aide d'une application. La quatrième partie décrit les bénéfices de la programmation spatiale. Finalement, la cinquième partie présente SPREAD, qui est un système permettant au programmeur de créer des applications spatiales.

5.1.1 Mémoire spatiale

Le mécanisme de mémoire spatiale transforme l'espace physique en une mémoire. Pour cela, chaque entité porte directement les données qui lui sont relatives et peut accéder aux données portées par les entités voisines. Dans cette mémoire, les données se déplacent avec les entités qui les portent. L'organisation géométrique des données est donc analogue à l'organisation géométrique des entités. De plus, tout changement dans l'organisation géométrique des entités implique un changement analogue dans l'organisation géométrique des données. Ainsi, les entités perçoivent les interactions physiques comme des déplacements de données. Par exemple, dans une bibliothèque, la bibliothèque détecte l'insertion d'un livre dans un rayon en détectant l'arrivée des données portées par ce livre.

Les entités portent leurs données via un calculateur embarqué contenant ces données. Ce calculateur est équipé d'une interface de communication à courte portée, qui permet à l'entité de communiquer avec les entités voisines.

Chaque entité dispose de deux opérations afin d'accéder à la mémoire :

- `out(t)` qui permet de publier la donnée t ;
- `read` qui permet de lire une donnée et la renvoie à l'application appelante.

5.1.1.1 Publication d'une donnée

Lorsqu'une entité publie une donnée avec une opération `out`, celle-ci est stockée dans le calculateur embarqué sur cette entité. Les données publiées par les entités peuvent aussi bien être des symboles élémentaires utilisés par le programme exécuté par l'entité que des informations de haut niveau, comme des données personnelles.

Les données portées par une entité ne doivent être accessibles que dans une zone de l'espace physique définie relativement à la position de cette entité. En effet, la détection d'une donnée correspond souvent à la détection d'une interaction physique. La donnée ne doit donc être détectée que lorsque l'interaction a bien eu lieu. Par exemple, supposons qu'une porte automatique ne s'ouvre que si elle détecte l'identifiant d'un utilisateur autorisé. Si l'identifiant de l'utilisateur occupe un trop grand volume, la porte peut détecter l'identifiant de l'utilisateur et s'ouvrir alors que celui-ci est trop éloigné pour que la porte puisse déterminer s'il souhaite la franchir ou non (cf. figure 5.1). La programmation spatiale associe donc chaque donnée à une zone de l'espace physique dans laquelle elle est accessible. Cette zone est définie par rapport à la position de l'entité qui porte la donnée et appelée forme de la donnée. La forme de la donnée doit être

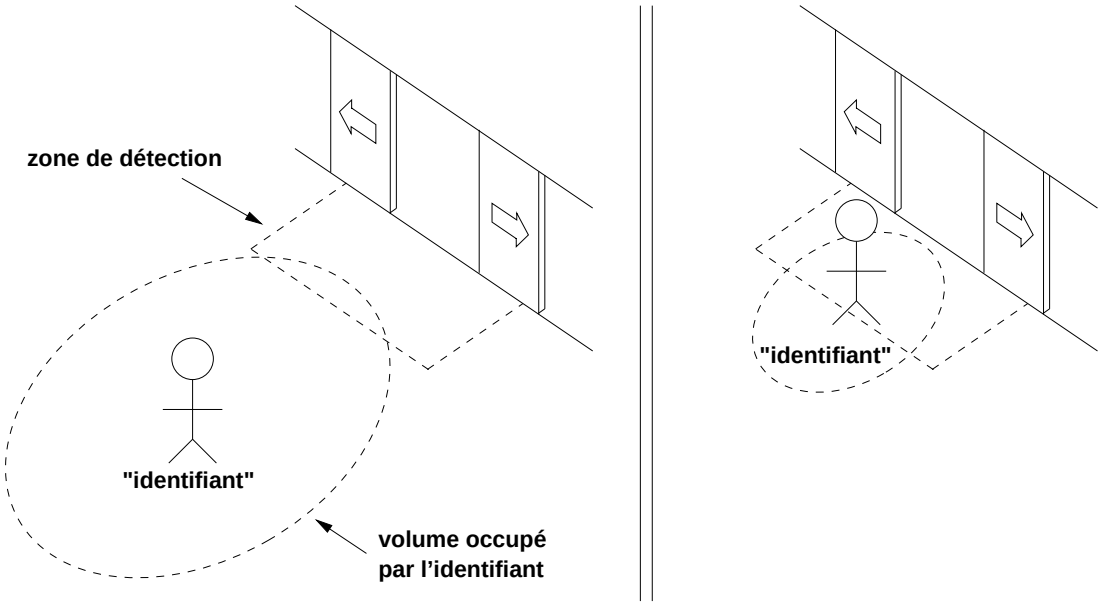

FIG. 5.1 – Illustration du problème de la taille du volume occupé par les données. Gauche : avec un grand volume. Droite : avec un petit volume. En utilisant un volume trop grand, la porte peut s'ouvrir alors que l'utilisateur ne veut pas la prendre.

adaptée à l'interaction physique à laquelle elle est associée.

Pour qu'une entité puisse accéder à une donnée, elle doit se trouver à l'intérieur de la forme de la donnée.

5.1.1.2 Lecture d'une donnée

Lorsqu'une entité exécute une opération `read`, elle a accès à un ensemble de données, ensemble qui peut être vide. Cet ensemble correspond aux données associées aux formes à l'intérieur desquelles l'entité se trouve (cf. figure5.2).

Pour sélectionner une donnée parmi les données accessibles, la programmation spatiale représente les données sous la forme de tuples. Une entité sélectionne un tuple parmi l'ensemble des tuples accessibles grâce à un mécanisme de correspondance (*matching*) entre tuples et motifs, similaire à celui employé par Linda [17]. Comme nous l'avons vu précédemment, un tuple est une suite ordonnée d'éléments typés : entiers, flottants ou chaînes de caractères. Un motif est un tuple dans lequel zéro ou plusieurs champs n'ont que leur type de défini. Il y a correspondance entre un tuple et un motif lorsqu'ils ont le même nombre d'éléments, les types des éléments se correspondent deux à deux et les valeurs des champs sont égales deux à deux. Un champ ne contenant qu'un nom de type comme `int` ou `string` est égal à toutes les valeurs que peut prendre ce type. Par exemple, le tuple `<10,“Voiture”,12.5>` correspond au motif `<int, string, 12.5>`. La définition formelle de la correspondance entre tuples et motifs est présentée

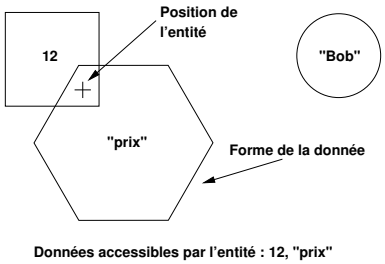

FIG. 5.2 – Une entité peur accéder à une donnée si elle se trouve à l'intérieur de la forme correspondante

dans [17].

La programmation spatiale choisit de représenter les données sous la forme de tuples pour plusieurs raisons. Tout d'abord, grâce au mécanisme de correspondance, une entité peut sélectionner un tuple parmi l'ensemble des tuples accessibles. Ensuite, la structure en suite ordonnée de champs offre un degré d'expressivité plus élevé que des données élémentaires, comme des mots machines, tout en restant relativement compacte. La compacité des données est un facteur important car la programmation spatiale est mise en œuvre sur des calculateurs embarqués ayant une mémoire de taille limitée.

Finalement, les données embarquées par les entités sont des tuples et l'opération `read` est dotée d'un paramètre, qui est un motif. Lorsque plusieurs tuples correspondent au motif, l'un d'entre eux est choisit au hasard et renvoyé à l'application. Si aucun tuple ne correspond au motif, l'opération `read` reste bloquée jusqu'à ce qu'un tuple correspondant au motif soit disponible.

5.1.2 Synchronisation physique

Dans la partie précédente nous avons présenté le mécanisme de mémoire spatiale qui nous permet de transformer l'espace physique en une mémoire. Ce mécanisme permet à une à une entité de porter les données qui lui sont relatives et de lire les données portées par les entités voisines. Dans cette mémoire, les données se déplacent avec les entités qui les portent et l'organisation géométrique des données est analogue à l'organisation des entités. Dans cette partie nous présentons le principe de synchronisation physique, qui repose sur l'opération lecture en mémoire.

Le mécanisme de synchronisation physique permet de synchroniser le programme exécuté par une entité sur la détection d'une interaction physique de type rencontre entre deux entité. Une rencontre entre deux entités correspond soit à la rencontre entre une entité fixe et une entité mobile, soit à la rencontre de deux entités mobiles. Lorsqu'une entité exécute une opération `read(m)`, celle-ci reste bloquée jusqu'à ce que l'entité soit à l'intérieur de la forme d'un tuple correspondant au motif m. D'un point de vue physique, l'opération reste bloquée jusqu'à ce que l'entité rencontre une autre entité portant un tuple correspondant au motif m. Chaque opération `read` détecte donc une rencontre et les traitements informatiques exécutés après cette opération sont synchronisés sur cette

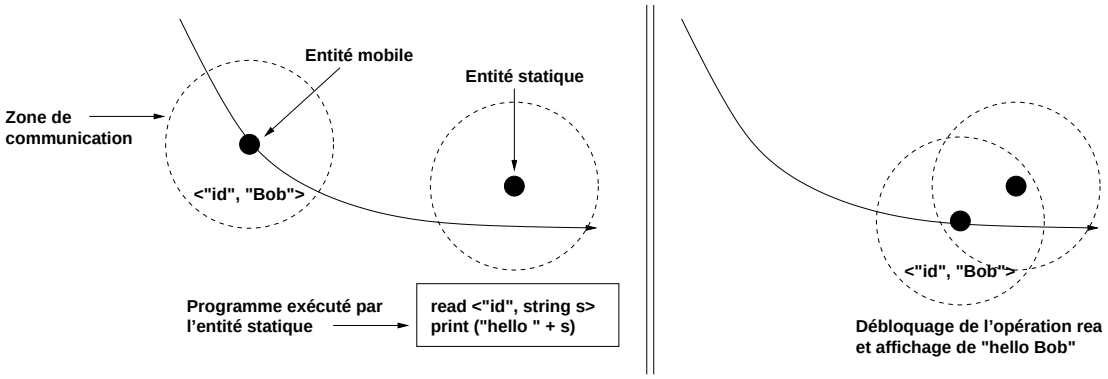

FIG. 5.3 – Exemple de synchronisation physique. Le programme de l'entité statique est synchronisé sur la découverte d'un tuple porté par l'entité mobile et donc sur la rencontre de cette entité.

détection. Le principe de synchronisation physique est illustré sur la figure 5.3.

Les systèmes parallèles traditionnels offrent différentes primitives de synchronisation, telles que les sémaphores ou barrières de synchronisation. Les sémaphores permettent de protéger l'accès à une ressource, telle qu'une zone de mémoire partagée ou un périphérique matériel. Lorsqu'un processus accède à une ressource, il la réserve via le sémaphore correspondant. Si la ressource est déjà réservée, le processus reste bloqué jusqu'à la libération du sémaphore correspondant. Les barrières de synchronisation permettent d'organiser des rendez-vous entre un nombre de processus donné. Les processus qui arrivent en premier à la barrière restent bloqués jusqu'à ce que tous les processus aient atteint la barrière. L'ensemble des processus reprennent alors leur exécution.

Le mécanisme de synchronisation physique est analogue à l'accès à une ressource protégée par un sémaphore. Ici, la ressource correspond au tuple et la libération de la ressource correspond à la découverte du tuple. La synchronisation par sémaphore est une synchronisation temporelle. Lors d'une synchronisation physique, la forme du tuple représente la zone physique dans laquelle la synchronisation peut avoir lieu. Selon la dimension temporelle, une synchronisation a lieu à un instant t, selon la dimension physique une synchronisation a lieu dans une zone donnée. La forme de la donnée est donc l'équivalent physique de l'instant t où la ressource est libérée.

Pour l'instant le mécanisme de synchronisation physique nous permet de synchroniser des applications uniquement sur des rencontres entre deux entités. Nous présentons dans les chapitres suivants de nouvelles opérations pour la programmation spatiale, qui nous permettent de synchroniser des applications sur d'autres interactions physiques, comme la séparation de deux entités.

5.1.3 Exemple d'application

Pour illustrer la programmation spatiale, nous détaillons dans cette partie l'application Ubibus [2]. L'objectif d'Ubibus est d'aider un malvoyant à prendre son bus, en

notifiant le malvoyant lorsque son bus est arrivé et en notifiant le conducteur lorsqu'une personne souhaite monter dans son bus.

Ubibus est composé de trois entités : le piéton, l'abri bus et le bus. Chacune d'elles exécute un programme. Le programme du piéton commence par publier la requête d'arrêt. Cette requête est représentée par un tuple composé d'un identifiant de requête, du numéro de ligne et de la direction choisie. Le programme est ensuite bloqué sur l'attente du tuple indiquant que le bus correspondant est arrivé. Le programme du piéton est le suivant :

```
out <''req_bus, 16, ``beaulieu''>
read<''bus_arrive'', 16, ``beaulieu''>
notifier_pieton();
```

Dans Ubibus, l'abri bus est un point de synchronisation fixe entre le piéton et le bus. Il détecte les requêtes de bus et publie les requêtes d'arrêt. Le programme de l'abri bus est donc bloqué sur la découverte d'une requête de bus. Une fois ce tuple détecté, l'abri bus publie la requête d'arrêt correspondante.

```
read <''req_bus, 16, ``beaulieu''>
out <''req_arret'', 16, ``beaulieu''>
```

Le piéton ne peut pas publier la requête d'arrêt lui-même car il pourrait alors arrêter le bus ailleurs qu'aux arrêts prévus. Nous aurions alors la sémantique d'une application pour arrêter un taxi : le piéton publie un tuple et un taxi s'arrête dès qu'il détecte ce tuple. Ici l'abri bus est utilisé comme point de synchronisation spatial entre le bus et le piéton. Ainsi, en utilisant une entité statique intermédiaire, nous pouvons donc fixer le lieu de synchronisation entre deux entités mobiles.

Le programme du bus quant à lui est initialement bloqué sur la découverte d'une requête d'arrêt. Une fois cette requête découverte, il publie le tuple indiquant au piéton que son bus est arrivé et prévient le chauffeur qu'un piéton souhaite prendre ce bus.

```
read<''req_arret'', 16, ``beaulieu''>
out <''bus_arrive'', 16, ``beaulieu''>
notifier_chauffeur();
```

Ubibus illustre le principe de synchronisation physique. Nous avons une première synchronisation entre le piéton et l'abri bus et une seconde synchronisation entre le bus et l'abri bus.

5.1.4 Bénéfices de la programmation spatiale

Les mécanismes de synchronisation physique et de mémoire spatiale permettent au programmeur d'exprimer simplement la rencontre entre deux entités en terme de lectures en mémoire. Ainsi, au niveau du code du programme exécuté par une entité, les opérations de lecture en mémoire sont une analogie directe des rencontres auxquelles

l'entité participera. Cette approche de la programmation impose au programmeur de penser en terme de rencontres entre entités et de définir les traitements associés à ces rencontres.

Les programmes réalisés grâce à la programmation spatiale sont simples car ils reposent fortement sur les interactions physiques existantes. Le programmeur se contente d'associer des traitements informatiques aux interactions physiques. Les interactions physiques contrôlent ensuite le déroulement du programme. Autrement dit, le flot de contrôle des programmes est représenté par la séquence d'interactions physiques.

La programmation spatiale s'oppose à l'approche virtuelle qui repose sur une modélisation logique du monde physique, comme une cartographie, stockée dans un serveur. Dans ce cas, le programmeur ne manipule pas des données mais des abstractions physiques telles que "l'utilisateur est entré dans le salon". Comme nous l'avons vu précédemment (cf. 1.3.2) l'approche virtuelle nécessite le déploiement d'une infrastructure de localisation et d'un infrastructure de communication pour relier les entités au serveur contenant le modèle. La programmation spatiale ne nécessite ni infrastructure de localisation, ni infrastructure de communication.

5.1.5 Le système SPREAD

Nous avons introduit précédemment le système SPREAD [8], qui permet de développer des applications spatiales (cf. 2.3.3).

5.1.5.1 Accès à la mémoire

SPREAD embarque sur chacune des entités un espace de tuples local et les entités peuvent accéder à cet espace de tuples à l'aide de plusieurs opérations :

- `out(t)` qui publie le tuple t dans l'espace de tuple local ;
- `read(m)` qui renvoie un tuple t correspondant au motif m ;
- `check(m)` qui vérifie la présence d'un tuple correspondant au motif m dans l'espace de tuple local ;
- `drop` qui supprime un tuple de l'espace de tuples local.
- `capture(m)` qui renvoie l'ensemble des tuples correspondant au motif m.

L'opération `read` reste bloquée jusqu'à ce qu'un tuple correspondant au motif soit accessible. Lorsqu'un programme exécute une opération `read(m)`, il est endormi et SPREAD commence alors par chercher un tuple correspondant au motif dans l'espace de tuple local de l'entité. Si aucun tuple n'est trouvé dans l'espace de tuples local, une requête est diffusée aux entités voisines afin de trouver un tuple correspondant dans leur espace de tuples respectif (cf. figure 5.4). Si aucun tuple n'est trouvé sur les entités voisines, SPREAD continue de les interroger régulièrement, jusqu'à ce qu'un tuple correspondant au motif ait été trouvé. Lorsqu'un tuple est finalement trouvé, le tuple est transféré à l'application et celle-ci est réveillée.

De leur coté, les opérations `capture` et `check` sont des opérations non bloquantes. L'opération `capture` interroge l'espace de tuples local et les entités voisines, tandis que l'opération `check` n'interroge que l'espace de tuples local.

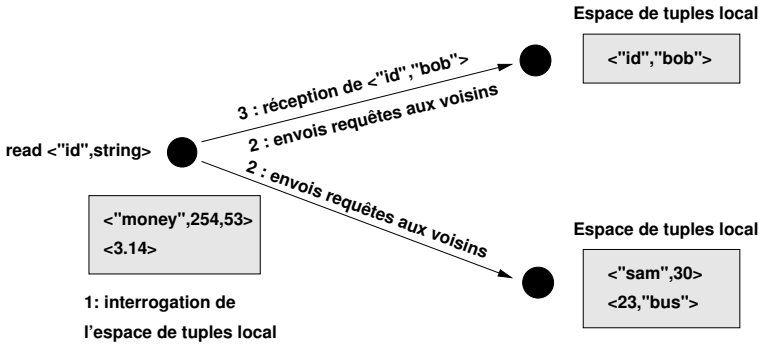

FIG. 5.4 – Protocole de l'opération `read`. L'opération interroge d'abord l'espace de tuples local et, si elle ne trouve pas de tuple correspondant, interroge ensuite les entités voisines.

SPREAD a été développé avec le langage de programmation Java. Il est disponible à la fois sur les PDA de type Pocket PC et les téléphones mobiles, disposant d'une machine d'exécution Java.

5.1.5.2 Forme des tuples

Actuellement la seule forme disponible pour les tuples est la sphère centrée sur l'entité portant ce tuple. Lorsqu'une entité publie un tuple, elle précise le rayon de la sphère associée à ce tuple. Une entité est à l'intérieur de la sphère d'un tuple lorsque la distance qui la sépare de l'entité correspondante est inférieure au rayon de la sphère. Une entité détecte la distance qui la sépare d'une seconde entité en mesurant la force du signal radio reçu depuis cette entité et en utilisant la fonction d'atténuation de l'interface radio. Cette fonction renvoie la distance entre l'entre l'entité réceptrice et l'entité émettrice en fonction de la force du signal radio reçu.

SPREAD repose sur des interfaces de communication radio à courte portée. Une entité ne peut accéder à un tuple que si elle se trouve à l'intérieur de la zone de communication de l'entité qui porte ce tuple. Le rayon maximum des sphères est donc limité à la portée de communication des entités.

5.2 Programmation spatiale et applications contextuelles

Dans cette partie, nous présentons l'application de la programmation spatiale à la réalisation d'applications contextuelles. Dans un premier temps nous présentons une définition du contexte basée sur une notion de proximité. Dans un second temps nous présentons comment utiliser la programmation spatiale pour réaliser des applications contextuelles basées sur cette définition du contexte.

5.2.1 Contexte et proximité

Les définitions habituelles représentent le contexte de l'utilisateur comme l'ensemble des paramètres caractérisant sa situation, tels que sa position, son rythme cardiaque, la luminosité extérieure... Couderc et Banâtre [9] proposent une approche différente en considérant que l'utilisateur est localisé dans un espace et que son contexte est représenté par l'ensemble des éléments de cet espace qui sont proches de lui. Par exemple, dans l'espace physique le contexte de l'utilisateur est représenté par les bâtiments ou les personnes proches. Cette notion de proximité n'est pas limitée à la dimension physique et peut être appliquée à d'autres dimensions, comme la dimension textuelle. Par exemple, le contexte d'un utilisateur qui surfe sur le Web est représenté par l'ensemble des pages qui sont proches de lui textuellement ; cet ensemble pouvant être déterminé à l'aide d'un moteur de recherche.

Pour déterminer si un élément est proche de l'utilisateur, Couderc et Banâtre supposent disposer d'une métrique permettant de calculer la distance entre un élément et l'utilisateur. Au sein d'un espace E composé des éléments e, le contexte C_U de l'utilisateur U est donc définit ainsi : $C_U = \{e_0, \ldots, e_n\}/\forall i, d(U, e_i) \leq T$, la fonction $d(U, e_i)$ permettant d'obtenir la distance entre l'utilisateur et l'élément e_i. T représente le seuil au-dessous duquel un élément est considéré comme proche.

5.2.2 Mise en œuvre grâce à la programmation spatiale

Lorsque l'utilisateur se déplace dans l'espace physique, il se déplace également dans la mémoire spatiale. Si le contexte de l'utilisateur est représenté par ce qui est proche de lui, alors, au sein de la mémoire spatiale ce même contexte est représenté par les tuples qui sont proches de l'utilisateur. Nous considérons qu'un tuple est proche de l'utilisateur si l'utilisateur se trouve à l'intérieur de la forme de ce tuple. Selon la dimension physique, le contexte de l'utilisateur est donc représenté par l'ensemble des tuples accessibles.

Une application contextuelle est le plus souvent constituée d'un cycle de capture du contexte et de délivrance d'un service, qui est fonction de ce contexte. Grâce à l'opération `capture`, SPREAD permet de capturer aisément le contexte physique de l'utilisateur. En effet, `capture(m)` renvoie l'ensemble des tuples accessibles qui correspondent au motif m. Le contexte de l'utilisateur change en fonction de ses déplacements. Il doit donc être maintenu à jour en exécutant régulièrement des `capture`.

5.2.3 Application

Dans [9], Couderc et Banâtre présentent WebWalker, une application de guide pour musées qui repose sur la programmation spatiale et un contexte de proximité (cf. 2.3.3). WebWalker associe à l'ensemble des œuvres de l'exposition des tuples contenant des URL et des mots clés. Les URL associées à une œuvre pointent vers des pages décrivant cette oeuvre. L'ensemble des mots clés proches sont utilisés pour envoyer une requête vers le moteur de recherche Google[1], afin de trouver des pages se rapportant à l'en-

[1] www.google.fr

vironnement physique de l'utilisateur. Ces pages sont déterminées en fonction de leur proximité textuelle aux mots clés présents dans le contexte de l'utilisateur. Le contexte de l'utilisateur est donc représenté par l'ensemble des pages Web proches de lui physiquement ou textuellement. Le contexte de l'utilisateur est régulièrement mis à jour en fonction de ses déplacements dans le musée.

5.3 Limitations

Dans cette partie nous présentons les deux principales limitations de la programmation spatiale. La première limitation est due à la forme des données et à leur mode d'adressage. Ceux-ci ne nous permettent pas de décrire suffisamment précisément les configurations géométriques des interactions physiques. La seconde limitation est due au manque d'expressivité du modèle de programmation. En effet, la sémantique de l'opération `read` ne nous permet de synchroniser les applications que sur des interactions physiques de type rencontres entre entités.

5.3.1 Adressage et forme des données

Actuellement, la forme d'une donnée est limitée à une sphère centrée sur l'entité qui l'a publiée. Cette forme n'est pas adaptée à toutes les applications.

Une donnée est le plus souvent associée à la détection d'une interaction physique, comme l'ouverture d'une porte automatique. La forme de la donnée conditionne la détection de l'interaction. Cette forme doit donc être adaptée à la forme des entités qui participent à l'interaction et à la configuration géométrique de l'interaction. En effet, si le volume associé à une donnée est trop différent de la forme réelle de l'entité correspondante, il est alors possible de détecter des interactions physiques qui n'ont pas réellement eu lieu. Considérons par exemple un pot à crayon intelligent qui détecte les insertions et retraits de crayons. Si nous associons une sphère aux données portées par un stylo, il est alors possible de détecter l'insertion d'un stylo dans le pot à crayon alors que le stylo est toujours sur le bureau (cf. figure 5.5). La forme des données doit donc être suffisamment précise afin d'assurer la cohérence entre l'état des programmes exécutés par les entités et l'état du monde physique.

Les interactions physiques peuvent également nécessiter des formes qui ne sont pas centrées sur l'entité. Pour l'exemple de la porte automatique, il serait intéressant que la forme de l'identifiant de l'utilisateur soit un cône situé en avant de l'utilisateur et non pas une sphère centrée sur lui.

Considérons maintenant une application de carrefour routier. L'objectif de cette application est de remplacer les feux tricolores par un dispositif ajouté dans les véhicules. Pour déterminer la couleurs du feux à afficher à l'intérieur des véhicules, ceux-ci doivent être capables de détecter la position relative des autres véhicules. Les véhicules signalent leur présence en publiant un tuple du type `<"voiture">` et chaque véhicule essaye en permanence de détecter les autres véhicules à l'aide d'opérations `read`. Actuellement, lorsque le `read` est débloqué, le véhicule qui l'exécute sait seulement qu'il se trouve à l'intérieur de la forme d'un autre véhicule. Par contre, il ne lui est pas possible de

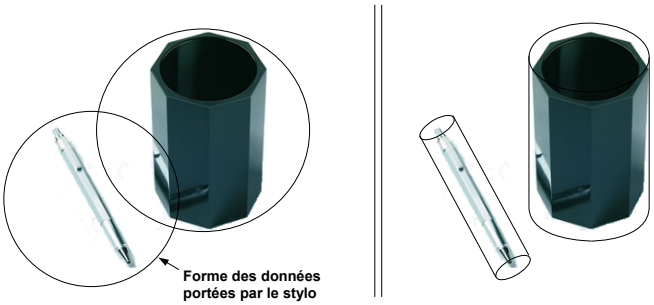

FIG. 5.5 – Illustration du problème de la forme des données. Gauche : les données sont sphériques, le crayon est considéré comme étant à l'intérieur du pot. Droite : les données ont une forme adaptée à la forme de l'objet correspondant, le crayon est considéré comme étant à l'extérieur du pot.

savoir si ce second véhicule se trouve sur sa gauche ou sur sa droite. De même, dans une bibliothèque il n'est pas possible pour un livre de distinguer les données portées par le livre situé à sa gauche des données portées par le livre de droite.

Dans le chapitre 6 nous présentons une extension géométrique pour la programmation spatiale, afin de permettre au programmeur de définir précisément la configuration géométrique des interactions physiques.

5.3.2 Détection d'interactions physiques

La programmation spatiale nous permet actuellement de synchroniser les applications sur les rencontres entre entités. Cependant, les applications ne sont pas uniquement composées de rencontres. Par exemple dans Ubibus, si le piéton change d'avis et quitte l'abri bus, alors l'abri bus doit effacer la requête d'arrêt. Avec les opérations actuellement disponibles, le départ d'une entité est détecté en analysant les états physiques successifs de l'environnement de l'entité. Lorsqu'un tuple est présent dans un état et absent dans le suivant, l'application détecte le départ de l'entité correspondante. Le départ d'une entité est donc détecté en scrutant explicitement l'environnement. Nous présentons dans le chapitre 7 de nouvelles opérations pour la programmation spatiale, qui nous permettent de détecter l'arrivée et le départ d'une entité, sans scrutation explicite de l'environnement.

Considérons maintenant une application aidant l'utilisateur à trouver un taxi. Pour trouver un taxi, l'utilisateur publie une tuple du type `<"taxi_req">`. Un taxi recherche un passager à l'aide d'une opération `read<"taxi_req">`. Avec cette approche, une fois le piéton dans le taxi, sa requête de taxi est toujours disponible pour d'autres taxis. Ici, nous avons besoin d'une opération qui détecte et supprime simultanément le tuple du piéton. Dans le chapitre 8, nous présentons une opération qui permet à une entité de supprimer un tuple porté par une autre entité.

5.4 Bilan

Dans ce chapitre nous avons présenté la programmation spatiale qui est un modèle de programmation pour réaliser des applications purement diffuse. La programmation spatiale utilise l'espace physique comme une mémoire : les données sont portées par les entités physiques et se déplacent avec elles. Grâce au mécanisme de synchronisation physique, nous pouvons créer des applications synchronisées sur les rencontres entre entités. Nous avons également présenté l'utilisation de la programmation spatiale pour créer des applications contextuelles. Finalement, nous avons présenté les limitations de la programmation spatiale, qui sont liées d'une part à la forme et l'adressage des données, et d'autre part à la sémantique de l'opération `read` qui ne nous permet de synchroniser les programmes que sur des rencontres entre entités. Dans les chapitres suivant nous modifions et étendons le modèle de programmation afin de prendre en compte ces limitations.

Chapitre 6

Programmation géométrique

Dans le chapitre précédent nous avons présenté la programmation spatiale, son application à la création d'applications contextuelles et ses limitations. Nous avons vu que le modèle de programmation ne nous permet pas de définir suffisamment précisément la configuration géométrique des interactions physiques. Dans cette partie, nous présentons une extension géométrique [38] qui pallie à cette limitation. Nous commençons par présenter le principe de cette extension géométrique. Nous poursuivons par sa réalisation et terminons par son évaluation.

6.1 Principe

L'extension géométrique propose à la fois de nouvelles forme pour les tuples et un nouveau mode d'adressage des tuples, appelé adressage géométrique. Nous présentons chacun de ces éléments dans cette partie, ainsi qu'un exemple de mise en œuvre de cette extension.

6.1.1 Forme des tuples

Actuellement les tuples sont tous sphériques. Comme nous l'avons vu précédemment (cf. 5.3.1), cette forme n'est pas adaptée à toutes les interactions physiques. Pour pallier à cette limitation, nous proposons désormais de choisir la forme des données parmi un ensemble de formes élémentaires : la sphère, la boîte, le cylindre, le cône, le secteur et le point (cf. figure 6.1). Cette forme est définie relativement à l'orientation et la position de l'entité.

Nous avons choisi de limiter la forme des tuples à des formes élémentaires pour conserver la réactivité de l'opération de lecture. En effet, lorsqu'une entité exécute une opération `read(m)`, elle reçoit des entités voisines les tuples correspondant au motif m. Pour chacun de ces tuples, l'entité doit ensuite vérifier si elle à l'intérieur de la forme correspondante. Elle renvoie à l'application le tuple associé au premier test d'inclusion réussi. Ces tests d'inclusion doivent être suffisamment rapides pour conserver la réactivité de l'opération `read`, or la complexité des formes augmente donc d'autant les temps

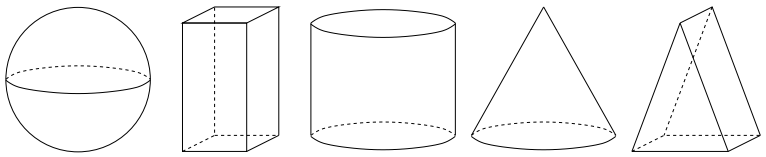

FIG. 6.1 – Formes disponibles pour les tuple

de calcul nécessaires aux tests. C'est pourquoi nous avons choisi de limiter la forme des tuples à des formes élémentaires.

Désormais, lorsqu'une entité publie un tuple, elle définit la structure du tuple, la valeur de ses champs et sa forme géométrique. La forme géométrique est définie dans le repère propre à l'entité, qui se déplace avec elle. Par exemple :

```
Point center(0,0,0);
Box user_box(center,1,1,2);
out(user_box, <‘‘user_id’’, ‘‘John’’>);
```

Ici l'application publie un tuple dont la forme est une boite centrée sur l'entité, d'un mètre de largeur et de profondeur, et de deux mètres de hauteur. Le tuple `<‘‘user_id’’, ‘‘John’’>` est accessible à l'intérieur de cette boite.

6.1.2 Adressage géométrique

Dans une mémoire traditionnelle nous pouvons définir des structures de données basées sur une notion d'ordre. Par exemple, dans une liste chaînée, à partir d'un élément nous pouvons accéder à l'élément précédent ou l'élément suivant. Dans une pile, nous ne pouvons accéder qu'à l'élément du dessus. Dans l'espace physique, nous pouvons également observer des structures de données reposant sur une notion d'ordre, ou plus précisément sur l'organisation relative des entités qui la composent. Par exemple, un carrefour routier est une structure de données géométrique relative, représentée par les différentes voies qui le composent. Pour déterminer si une voiture peut traverser ou non le carrefour, nous devons accéder aux différentes voies relativement à celle occupée par la voiture. Une pile de documents est également une structure de données géométrique relative : le document situé au-dessus du document courant a la plus grande priorité. Une bibliothèque est également une structure de données relative, les livres y sont rangés selon l'ordre alphabétique du nom de leur auteur.

Actuellement, lorsque qu'un `read` est débloqué, la seule information dont nous disposons est que l'entité qui exécute le `read` se trouve dans la forme du tuple qui a débloqué l'opération. Cette information ne lui permet pas de connaître la position du tuple relativement à son orientation. Nous ne pouvons donc pas de manipuler des structure de données physiques basées sur l'organisation relative des entités, telles qu'un carrefour routier.

Nous proposons donc d'associer à chaque opération `read` une zone d'adressage, qui est définie par rapport à la position et l'orientation de l'entité. Avec ce mode d'adressage

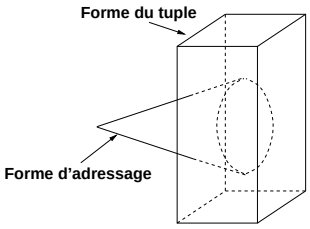

FIG. 6.2 – Adressage géométrique : un tuple est accessible lorsque la forme du tuple est en intersection avec la zone d'adressage

un tuple est accessible si la zone d'adressage est en intersection avec la forme du tuple (cf. figure 6.2). Les formes disponibles pour la zone d'adressage sont les mêmes que pour la forme des tuples. Revenons à notre exemple de carrefour routier. Une voiture exécute une opération du type `read(secteurDroit,<“voiture”>)` pour détecter si une autre voiture sur sa droite, avec `secteurDroit` qui représente la zone d'adressage de l'opération. Ainsi lorsque l'opération `read` se débloque, la voiture détecte qu'une autre voiture se trouve sur sa droite. Nous appelons ce mode d'adressage : adressage géométrique. L'adressage géométrique nous permet de manipuler des structures de données physiques relatives.

Notons que pour revenir au mode d'adressage initial de la programmation spatiale, il suffit de prendre un point comme zone d'adressage. Le point est en intersection avec une forme simplement s'il est situé à l'intérieur de cette forme.

6.1.3 Mise en oeuvre

Pour réaliser une application spatiale, le programmeur doit définir le programme exécuté par chaque entité. Avec l'adressage géométrique, le programmeur définit en plus les formes associées aux tuples et aux opérations de lecture. Nous mettons maintenant en œuvre ce nouveau modèle de programmation pour réaliser l'application du chariot présentée précédemment. Cette application permet de calculer le prix total du chariot à la fin des commissions, sans avoir à le vider. Pour cela, chaque produit publie son prix sous la forme d'un point à l'aide du programme suivant :

```
Point p(0,0,0);
out (p,<''prix'',prix>);
```

Le point associé au prix représente l'origine du repère associé au produit correspondant. Ce repère se déplace avec le produit.

A la fin des commissions, le client passe dans une zone de paiement avec son chariot. Cette zone de paiement exécute le programme suivant :

```
Point origine(0,0,0);
Boite zone_lecture(origine,2,2,2);
List list_prix = capture(zone_lecture,<''prix'', ? >);
total = add_list(list_prix);
```

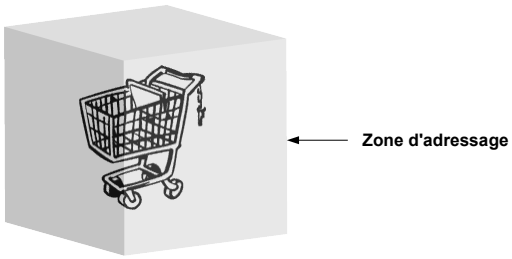

FIG. 6.3 – Définition d'une zone de paiment pour calculer le prix d'un chariot

Ce programme commence par définir une zone d'adressage, représentée par une boîte de deux mètres de côté, dans laquelle le client passe avec son chariot. Ensuite, la zone de paiement capture l'ensemble des prix contenus dans le chariot et les additionne. Ceci est illustré sur la figure 6.3.

6.2 Réalisation

Nous avons intégré cette extension géométrique dans SPREAD. Pour cela chaque entité est associée à un repère qui est aligné sur l'orientation de l'entité et se déplace avec elle. Une entité définit la forme de ses données dans son propre repère.

Les tests d'intersection entre formes des données et zones d'adressage sont réalisés soit avec une méthode exacte basée sur les propriétés géométriques des volumes, soit avec une méthode basée sur des volumes approchés, composés d'un ensemble de tétraèdres élémentaires. Par exemple, dans le cas de la méthode exacte, deux sphères sont en intersection si la distance entre les centres est inférieure à la somme des rayons. Pour la seconde méthode, les volumes sont approchés par un assemblage de tétraèdres. Les tétraèdres de chaque volume sont ensuite testés deux à deux. L'algorithme du calcul d'intersection entre deux tétraèdres est celui présenté dans [16]. Lorsque deux tétraèdres sont en collision, les volumes correspondant sont également en collision. La figure 6.4 présente la décomposition en tétraèdres d'une boîte, d'un cône, et d'un secteur. Cette figure présente également l'approximation d'un cylindre par deux boîtes.

Les tests d'intersection doivent donc être les plus rapides possible afin d'assurer une bonne réactivité aux applications. Pour les tests d'intersection entre un point et un volume quelconque ou entre une sphère et un volume quelconque nous nous reposons sur la méthode exacte. Pour les autres cas, tels que l'intersection entre un cylindre et un secteur, les algorithmes employés par la méthode exacte deviennent très complexes. Dans un souci de réactivité et de simplicité, nous nous rabattons alors sur la méthode par décomposition en tétraèdres élémentaires.

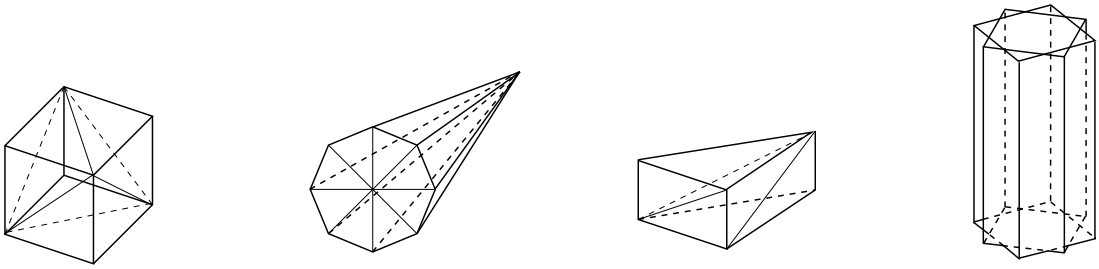

FIG. 6.4 – Décomposition en tétraèdres d'une boite, d'un cône et d'un secteur. Le cylindre est approché par deux boites, elles-mêmes décomposées en tétraèdres.

	Boîte	Cône	Cylindre	Point	Secteur	Sphère	Tétraèdre
Boite	23.0t	55.7t	50.8t	3.9t	19.3t	4.8t	18.7t
Cône		67.4t	65.1t	5.3t	24.0t	5.6t	24.8t
Cylindre			49.6t	5.6t	24.8t	5.2t	40.0t
Point				0.7t	4.0t	1.1t	3.4t
Secteur					21.4t	24.2t	25.6t
Sphère						1.0t	11.1t
Tétraèdre							8.8t

TAB. 6.1 – Durée des tests en fonction de la durée du test entre deux sphère

6.3 Évaluation

Nous évaluons maintenant le temps moyen des tests d'intersection. Pour cette première évaluation, nous cherchons à reproduire les différentes configurations géométriques pouvant se produire au cours d'une application. Pour cela, nous définissons un cube de 200 mètres de côté. Dans ce cube, nous générons des volumes aléatoirement, dont les dimensions caractéristiques sont comprises entre 1 et 100 mètres. Par exemple, les dimensions caractéristiques d'un cylindre ou d'un cône sont le rayon et la hauteur. Pour chaque couple de volumes possible nous générons 2000 configurations géométriques et mesurons la durée du test de collision.

Quelle que soit l'approche retenue pour effectuer les tests d'intersection, les algorithmes utilisés sont uniquement basés sur du calcul matriciel, qui ne comporte que des branchements et des opérations mathématiques portant sur des données locales. Ainsi, plutôt que de présenter des temps absolus et liés à la plate-forme de test, nous préférons exprimer la durée des différents tests en fonction d'une durée élémentaire t représentée par la durée du test d'intersection entre deux sphères. Les résultats sont présentés dans la table 6.1.

Les valeurs obtenues nous permettent de comparer le temps moyen de chaque combinaison de volumes en fonction du temps obtenu pour l'intersection de deux sphères. Le temps le plus long est celui obtenu pour une collision entre deux cônes qui est de

$67.4t$. avec $t = 0.09$ms. Grâce à cette table, nous pouvons choisir lors de la conception d'une application les différentes combinaisons de volumes en fonction des besoins de l'application et de la puissance des calculateurs embarqués sur les entités.

Il est difficile d'évaluer l'impact des tests d'intersection sur la réactivité de l'application avec ces chiffres, car le temps total consommé par les tests d'intersection dépend du nombre de tuples reçus par l'entité. Par exemple, si une opération `read` utilise une zone d'adressage conique et reçoit 20 tuples cylindriques, le temps pris par les tests d'intersection est en moyenne de : $`20 * 65.1 * 0.09 = 117.2$ms. Si l'opération ne reçoit qu'un tuple ponctuel, le test d'intersection dure 0.5ms. Cependant, pour garantir la réactivité, nous pouvons allouer un temps fixe pour les tests d'intersection. Si nous connaissons le nombre maximum de tuples que peut recevoir une opération, nous pouvons déterminer les combinaisons de volumes qui garantissent ce temps de réaction. Par exemple, si nous allouons aux tests d'intersection 100 millisecondes et si l'opération `read` reçoit au maximum 20 tuples, nous pouvons utiliser toute les combinaisons de volumes dont le test d'intersection a une durée inférieure à 5 millisecondes.

6.4 Discussion

6.4.1 Travaux apparentés

Les bases de données spatiales permettent également d'associer une forme géométrique à des données. Ces formes géométriques sont utilisées pour modéliser le monde physique. Une fois la base de données constituée, il est possible de l'interroger à la manière d'une base de données classique. Pour une base de données gérant des données démographiques il est par exemple possible de faire une requête pour connaître la densité de population d'un quartier. Les bases de données spatiales ont été développées conjointement aux Systèmes d'Information Géographiques [44, 35] pour leur offrir une solution de stockage des données adaptée à leurs besoins. Une introduction détaillée aux bases de données spatiales est faite dans [24].

Notre approche diffère des bases de données spatiales de par la distribution dans l'espace physique des données et de leur forme géométrique. Une base de données spatiale est un système typiquement virtuel (cf. 1.3.2). Lorsqu'une base de données spatiale inclut des données liées à des entités mobiles, nous devons déployer un système de localisation pour suivre la position des entités et un réseau de communication global pour relier les entités à la base de données spatiale. L'ensemble des entités mobiles envoient leur position à la base de données. Lorsque le nombre d'entités est important, la base de données peut donc devenir un goulot d'étranglement. Nous avons alors un problème d'extensibilité. Dans notre cas, les données et les formes géométriques sont directement portées par les entités et se déplacent avec elles. Le problème des mises à jour de positions et par la même le problème d'extensibilité ne se posent donc pas. Notre approche nécessite également un système de positionnement, cependant celui-ci n'est utilisé que pour déterminer la position relative des entités. Le système de positionnement n'est pas utilisé pour mettre à jour une base de données globale.

6.4.2 Bilan

Dans ce chapitre nous avons présenté une extension géométrique pour la programmation spatiale. Cette extension permet aux entités de définir précisément les configurations des interactions physiques à détecter. Pour cela, l'extension géométrique permet aux entités d'associer une forme géométrique aux tuples et une zone d'adressage aux opérations de lecture. Un tuple ne peut être lu que si la zone d'adressage de l'opération est en intersection avec la forme de ce tuple. Ce mode d'adressage géométrique est adapté aux structures de données physiques relatives et permet à une entité d'adresser les tuples relativement à son orientation.

Grâce au mécanisme de synchronisation physique, le programmeur décrit explicitement dans son code les rencontres entre entité. Avec l'adressage géométrique, le programmeur décrit non seulement ces rencontres, mais également les configurations géométriques dans lesquelles elles doivent se produire. Le code de l'application reflète donc directement les interactions qui ont lieu dans le monde physique et leur configuration géométrique. Cependant, cette plus grande expressivité au niveau du modèle de programmation implique des calculs supplémentaires et donc une réactivité des applications plus faible.

Nous avons évalué la durée moyenne des tests d'intersection entre formes des données et zones d'adressage, en fonction d'une durée élémentaire correspondant à la durée d'un test d'intersection entre deux sphères. Les résultats de l'évaluation nous permettent de choisir les combinaisons de volumes en fonction des besoins de l'application visée et de la puissance des calculateurs mis en jeu.

Chapitre 7

Synchronisation sur les arrivées et départs d'entités

Dans le chapitre précédent, nous avons présenté une extension géométrique pour la programmation spatiale. Désormais, l'opération d'écriture en mémoire est dotée d'un paramètre supplémentaire qui correspond à la forme géométrique du tuple. De leur côté, les opérations de lecture sont également dotées d'un paramètre supplémentaire qui correspond à la zone d'adressage de l'opération. Un tuple ne peut être lu que si sa forme est en intersection avec la zone d'adressage de l'opération. Ces formes géométriques nous permettent de définir plus précisément les configurations géométriques des interactions physiques à détecter.

Dans ce chapitre, nous présentons deux nouvelles opérations de lecture en mémoire. En effet, la sémantique des opérations actuellement disponibles ne permet pas de programmer simplement certaines applications. Actuellement, il n'est pas possible de synchroniser une application sur le départ d'une entité d'une zone de l'espace physique ou sur la séparation de deux entités. Seule la synchronisation sur la rencontre de deux entités est possible, via l'opération `read`.

7.1 Limitation du modèle de programmation

Dans cette partie, nous présentons une limitation du modèle de programmation par l'intermédiaire d'une application qui calcule automatiquement le prix d'un chariot. Dans la première version de cette application, le prix du chariot était calculé à la fin des commissions par une zone de lecture. Désormais, nous voulons que le chariot calcule lui-même son prix, au fur et à mesure des insertions et des retraits de produits (cf. figure 7.1). Le programme exécuté par le chariot est le suivant :

```
Point origine(0,0,0);
Boite boite_chariot(origine,0.5,0.5,1); //boite englobant le chariot
while(1) {
  List list_prix = capture(zone_lecture,<''prix'', ? >);
```

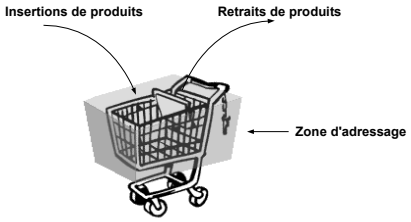

FIG. 7.1 – Le chariot calcule lui-même son prix, en fonction des insertions et retraits de produits

```
  total = add_list(list_prix);
  afficher(total);
}
```

Pour maintenir le prix à jour, nous observons que le chariot est obligé d'inclure une opération `capture` dans une boucle. Cette stratégie correspond à une scrutation de l'environnement physique. Cette approche n'est pas satisfaisante. D'une part, l'opération `capture` ne garantit pas que la liste de tuples renvoyée soit exhaustive. En effet, une opération `capture` n'attend que pendant un temps limité les réponses des entités voisines. Il est donc possible que certains tuples correspondant aux motifs de l'opération ne soient pas reçus par le `capture`. D'autre part, ce style de programmation ne reflète pas les actions qui ont lieu dans le monde physique et correspond mal au "paradigme" de la programmation spatiale. Nous ne pouvons pas utiliser l'opération `read` car celle-ci choisit un tuple au hasard lorsque plusieurs tuples correspondent au motif. Ainsi, une séquence de `read` ne garantit pas une lecture exhaustive du contenu du chariot.

Dans la partie suivante nous présentons deux nouvelles opérations qui permettent d'exprimer simplement, au niveau du code de l'application, l'ajout et le retrait d'un produit dans le chariot. La première opération permet de synchroniser les applications sur la détection d'un *nouveau* tuple. La seconde opération synchronise une application sur la disparition d'un tuple.

7.2 Les opérations readOnce et lostOne

Dans la partie précédente nous avons vu que le modèle de programmation ne permet pas d'exprimer simplement l'ajout et le retrait d'un produit dans un chariot. Ce problème n'est pas limité à l'application du chariot. Nous ne pouvons pas non plus détecter simplement l'arrivée et le départ d'un piéton au niveau d'un abri bus, ou bien l'ajout ou le retrait d'un livre d'une étagère. Ce problème revient donc à détecter l'arrivée et le départ d'une entité, dans une zone où d'autres entités du même type peuvent être déjà présentes. Actuellement, une scrutation explicite de l'environnement physique avec l'opération `capture` est nécessaire pour résoudre ce problème.

Dans cette partie, nous présentons donc deux nouvelles opérations : `readOnce` et `lostOne` qui solutionnent ce problème simplement.

7.2.1 Principe

L'opération `readOnce` renvoie un tuple qui n'a pas été lu jusqu'à présent et reste bloquée jusqu'à ce qu'un tel tuple soit disponible. Ainsi, en appelant plusieurs fois de suite `readOnce(m)`, une application peut énumérer l'ensemble des tuples correspondant à m. Lorsqu'il ne reste plus de tuple à lire, `readOnce` reste bloquée. Grâce à `readOnce`, nous pouvons donc synchroniser une application sur l'arrivée d'un *nouveau* tuple et donc sur l'arrivée de l'entité qui le porte.

L'opération `readOnce` se différencie de l'opération `read` dans sa stratégie de sélection du tuple renvoyé. L'opération `read` ne distingue les tuples les uns des autres que par leur structure et la valeur des différents champs. Si plusieurs tuples identiques correspondent au motif m, du point de vue de l'application l'opération `read(m)` renvoie toujours le même tuple. Il n'est donc pas possible d'énumérer le contenu de l'espace de tuple avec une séquence de `read(m)`. De son côté, l'opération `readOnce` distingue les tuples les uns des autres et renvoie systématiquement un nouveau tuple.

L'opération `lostOne` est l'équivalent du `readOnce` pour la disparition d'une entité ; elle reste bloquée jusqu'à ce qu'un tuple ait disparu. Ce tuple doit avoir été lu auparavant avec un `readOnce`. L'opération `lostOne` synchronise l'opération sur la disparition d'un tuple, qui correspond au départ de l'entité qui le porte. Lorsqu'un tuple a disparu, il peut être de nouveau lu par une opération `readOnce`.

Il est très simple de programmer une application du type chariot (cf. 7.1) avec ces deux opérations. Nous avons deux processus qui s'exécutent en parallèle sur le chariot. Le premier processus détecte l'ajout d'un produit et le second le retrait. Les programmes exécutés par chacun des processus sont les suivants :

```
//Premier processus
while(1) {
  prix = readOnce(boite_chariot,<''prix'', ?>);
  total += prix;
}

//Second processus
while(1) {
  prix = lostOne(boite_chariot,<''prix'', ?>);
  total -= prix;
}
```

Dans ces programmes `box_chariot` correspond au volume englobant le chariot. Les tuples correspondant aux prix des articles sont ponctuels.

Avec cette approche, les nouveaux tuples sont lus un par un, de même que les tuples disparus sont détectés un par un. Ainsi, lorsque plusieurs produits sont insérés dans le chariot, le prix affiché ne sera cohérent avec l'espace physique qu'après la lecture de

l'ensemble des nouveaux tuples. Le prix du chariot converge au bout d'un temps t qui est fonction du temps de réaction de l'opération `readOnce` et du nombre de produits insérés simultanément. Nous évaluons dans la partie 7.3.4 les temps de réaction des opérations `readOnce` et `lostOne`.

7.2.2 Étiquetage des interactions physiques

Nous avons vu dans le chapitre 5 que pour réaliser une application spatiale le programmeur doit tout d'abord identifier les interactions physiques qui composent cette application. Il définit ensuite les traitement informatiques à exécuter lorsque ces interactions se produisent. Jusqu'à présent nous ne pouvions synchroniser des traitements informatiques que sur la rencontre de deux entités. Désormais, nous pouvons synchroniser des traitements informatiques sur trois types d'interactions physiques : i) la rencontre de deux entités, grâce à l'opération `read` ; ii) l'arrivée d'une entité dans un zone de l'espace physique, grâce à l'opération `readOnce` ; iii) le départ d'une entité précédemment rencontrée d'une zone de l'espace physique, grâce à l'opération `lostOne`.

Nous considérons que les opérations `read`, `readOnce` et `lostOne` nous permettent d'étiqueter les interactions physiques avec des traitements informatiques [40]. Une interaction est associée à l'une de ces opérations et étiquetée avec les traitements qui sont exécutés à la suite de l'opération.

7.3 Réalisation

Nous avons intégré les opérations `readOnce` et `lostOne` dans SPREAD. Ces opérations ont donc été développées avec le langage Java. Nous présentons maintenant leur fonctionnement.

7.3.1 Opération readOnce

L'opération `readOnce` consiste principalement en une scrutation de l'environnement afin de découvrir un nouveau tuple. Auparavant cette scrutation était faite explicitement dans le code des programmes, à l'aide d'une séquence de `capture`. Elle est désormais encapsulée dans l'opération `readOnce`.

Pour découvrir un nouveau tuple, l'opération `readOnce` diffuse régulièrement aux entités voisines une requête de tuples contenant le motif et sa zone d'adressage. Les entités voisines recherchent les tuples correspondant dans leur espace de tuple local et renvoient l'ensemble des tuples dont la forme est en intersection avec la zone d'adressage. L'opération `readOnce` recherche parmi les tuples reçus, un tuple qu'elle n'a pas encore lu et le renvoie à l'application. Si l'opération `readOnce` ne trouve pas de nouveau tuple, elle rediffuse alors régulièrement une requête de tuples aux entités voisines.

L'opération `readOnce` renvoie systématiquement un nouveau tuple. Elle doit donc pouvoir distinguer les tuples les uns des autres et conserver un historique des tuples déjà lus. Pour distinguer les tuples les uns des autres, nous avons ajouté à chaque tuple un identifiant unique. L'entité conserve dans un tableau les identifiants des tuples déjà

lus. Dans notre cas, les identifiants sont composés d'un compteur de tuples propre à l'entité et de l'adresse IP de l'entité.

7.3.2 Opération lostOne

Pour détecter la disparition d'un tuple, l'opération `lostOne` vérifie régulièrement que les tuples déjà lus sont toujours accessibles. Si l'un de ces tuples n'est plus accessible pendant une durée supérieure à t_{max}, il est alors déclaré disparu. Le temps de réaction sera nécessairement supérieur à t_{max}, la valeur de t_{max} conditionne donc directement la réactivité de l'opération `lostOne`.

Pour détecter la disparition d'un tuple, l'opération `lostOne` commence par associer une estampille à chaque tuple déjà lu. Initialement, cette estampille est égale à la date de découverte du tuple. L'opération `lostOne` diffuse ensuite régulièrement des requêtes de tuples aux entités voisines, pour vérifier la présence des tuples et mettre à jour leur estampille. Au début de chaque requête les estampilles des tuples sont vérifiées et les tuples dont l'estampille est âgée de plus de t_{max} sont déclarés disparus.

Nous fixons t_{max} à deux fois la période des requêtes de tuples. Si la période des requêtes est de 600 millisecondes, le temps de détection de la perte d'un tuple sera au minimum de 1200 millisecondes. Nous considérons un tuple comme disparu après deux requêtes afin de distinguer les pertes de paquets des tuples réellement disparus. En effet, les entités communiquent via des communications de proximité qui sont non fiables. Les tuples envoyés par une entité à une autre entité peuvent donc être perdus par le réseau alors qu'ils sont toujours physiquement présents.

Notons que les requêtes de tuples sont les mêmes pour les opérations `lostOne` et `readOnce`. Les entités qui reçoivent ces requêtes ne distinguent donc pas les opérations les unes des autres. Elles se contentent de renvoyer à l'entité qui fait la requête, l'ensemble des tuples qui correspondent au motif et dont la forme est en intersection avec la zone d'adressage.

7.3.3 Support technologique

L'implémentation actuelle suppose que les entités qui participent à l'application embarquent chacune un Pocket PC. Cette solution n'est pas applicable dans la pratique, aussi bien pour des raisons de coûts financiers que d'encombrement. Deux approches sont possibles : les RFID et les capteurs intelligents. Les RFID sont des étiquettes électroniques qui nous permettent d'embarquer de l'information sur les entités physiques à moindre coût. Les RFID sont lisibles à distance par l'intermédiaire d'ondes radio et sont distinguables les uns des autres. Ainsi, les tuples seraient directement stockés dans des RFID embarqués sur les objets. Le point faible de cette approche est que seule l'entité qui est équipée du lecteur d'étiquettes peut détecter les apparitions de nouvelles étiquettes et les pertes d'étiquettes. De plus, un lecteur d'étiquettes est coûteux et relativement encombrant. Cette approche convient donc bien pour les applications où les entités devant exécuter des opérations `readOnce` et `lostOne` sont rares par rapport aux entités devant porter de l'information. Cette approche est par exemple adaptée à

Période des requêtes	`readOnce`	`lostOne`
150ms	200ms	470ms
300ms	260ms	780ms
600ms	470ms	1590ms

TAB. 7.1 – Temps de réaction des opérations `readOnce` et `lostOne` en fonction la période des requêtes de tuples

l'application du chariot : chaque produit porte un RFID contenant son prix et le chariot est équipé d'un lecteur d'étiquettes.

Si les entités doivent toutes exécuter des opérations `readOnce` et `lostOne`, nous devons nous rabattre sur des technologies de type capteurs intelligents qui ont un coût et un encombrement moindre que les Pocket PC, mais supérieur aux RFID. Ce type de plate-forme a également des capacités énergétiques et de calcul très réduites, dont il faudrait tenir compte dans l'implémentation des opérations. Par exemple, ce type d'architecture est le plus souvent endormie, il nous faudrait donc adapter le protocole des opérations `readOnce` et `lostOne` à ce mode de fonctionnement, car actuellement ces opérations reposent sur une attente active.

7.3.4 Évaluation

Afin de valider la réalisation des opérations `readOnce` et `lostOnce` nous avons effectué une série d'évaluations. Cependant, la réalisation étant dépendante à la fois de SPREAD et des technologies de la plate-forme d'exécution, nous ne présentons ici que les résultats principaux des évaluations. Les résultats complets sont présentés dans l'annexe A.

Nous avons évalué le temps de réaction des opérations `readOnce` et `lostOne` en fonction de la période des requêtes de tuples. Ces évaluations montrent que nous pouvons ajuster le temps de réaction des opérations en modifiant la période des requêtes de tuples. Par exemple, le temps de réaction moyen de l'opération `readOnce` est de 410 millisecondes avec une période de 600 millisecondes et de 200 millisecondes avec une période de 150 millisecondes. Les résultats sont présentés dans le tableau 7.1.

Aux vue de ces résultats, nous serions tenté de choisir la période la plus petite possible afin de maximiser la réactivité des opérations `readOnce` et `lostOne`. Cependant, en diminuant la période des requêtes de tuples nous limitons également le nombre total de tuples que peut gérer une opération. En effet, lors de chaque requête de tuples, les opérations reçoivent un certain nombre de tuples des entités voisines. Lorsque le temps de traitement de ces tuples par l'opération est supérieur à la période des requêtes de tuples, une congestion apparaît au niveau de l'entité qui reçoit les tuples. Cette congestion entraîne la perte de nombreux tuples et les opérations `readOnce` et `lostOne` deviennent inutilisables. Les courbes 7.2 et 7.3 donnent le nombre de tuples maximal que peuvent traiter les opérations `readOnce` et `lostOne` en fonction de la période des requêtes de tuples. Ainsi, si nous connaissons à l'avance le nombre de tuples maximal

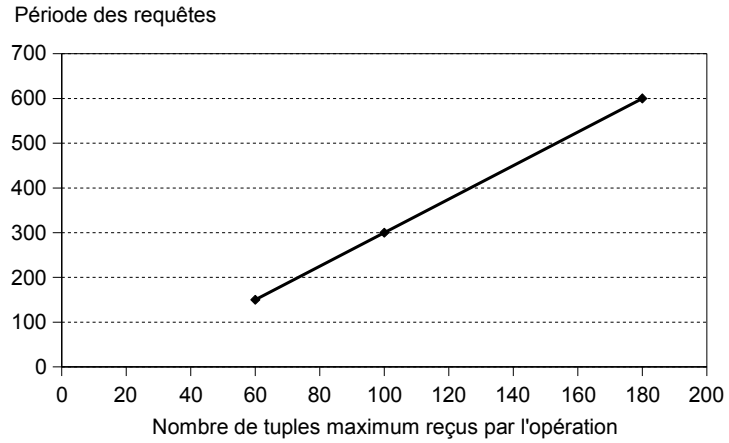

FIG. 7.2 – Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `readOnce`

qui peut recevoir une opération, nous pouvons choisir grâce aux courbes 7.2 et 7.3 la période donnant la meilleure réactivité, tout en évitant la congestion.

La réactivité obtenue limite le champ d'application des opérations `readOnce` et `lostOne` aux applications destinées aux êtres humains. Les temps de réactivité obtenus ne nous permettent pas d'envisager l'utilisation de ces opérations lorsque la réactivité doit être très élevée, comme dans le contexte de chaînes de productions par exemple. Cependant, du point de vue d'un être humain, la réactivité obtenue est acceptable. Par exemple avec notre application de chariot, lorsque le client insère un produit dans le chariot, le temps de réaction sera de 200 millisecondes pour une période de 150 millisecondes. Si le client retire un produit, le temps de réaction sera de 470 millisecondes. Ainsi, à moins que l'utilisateur ne surveille constamment l'afficheur du chariot, le prix affiché sera donc toujours cohérent avec le contenu du chariot.

7.4 Bilan

Dans cette partie nous avons présenté deux nouvelles opérations : `readOnce` et `lostOne`. Les opérations `readOnce` et `lostOne` permettent de synchroniser les programmes exécutés par les entités sur la découverte d'un nouveau tuple et la disparition d'un tuple. L'opération `readOnce` permet donc de lire un tuple qui n'a pas encore été lu, tandis que l'opération `lostOne` renvoie un tuple qui a disparu. Ces deux opérations sont particulièrement utiles pour détecter des interactions physiques telles que les entrées et sorties d'entités du même type d'une zone donnée de l'espace physique, comme pour l'application du chariot.

Nous avons intégré ces opérations dans SPREAD et évalué leur réactivité. La réactivité des opérations est acceptable du point de vue de l'utilisateur, mais ne nous permet pas d'utiliser ces opérations pour des applications nécessitant une réactivité élevée. Si

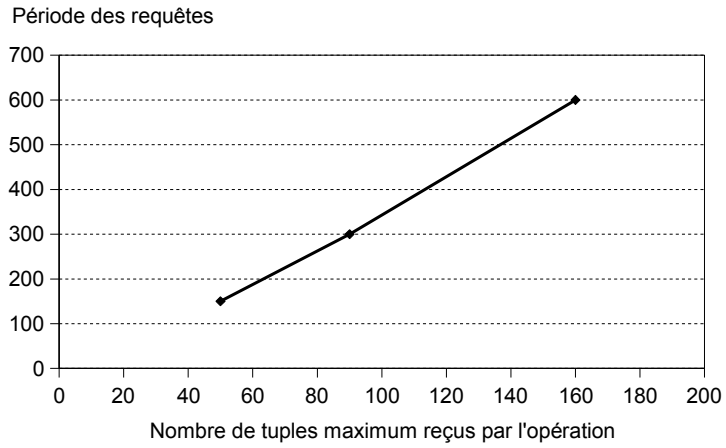

FIG. 7.3 – Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `lostOne`

nous connaissons à l'avance le profil de l'application, nous pouvons choisir la réactivité optimale en fonction du nombre maximum de tuples gérés par une entité.

Ces évaluations nous ont permis de valider le fonctionnement de ces opérations. Cependant, dans un cas réel nous ne pourrions utiliser un Pocket PC par entité physique et nous devrions donc utiliser des technologies plus adaptées, telles que les RFID ou les capteurs intelligents.

Chapitre 8

Transfert atomique de tuple

Dans le chapitre précédent nous avons présenté deux nouvelles opérations pour la programmation spatiale. Ces opérations permettent au programmeur de synchroniser ses programmes sur l'arrivée d'une nouvelle entité et sur le départ d'une entité précédemment rencontrée.

Dans ce chapitre, nous considérons l'application qui aide l'utilisateur à trouver un taxi. Pour trouver un taxi, l'utilisateur publie un tuple du type `<“ taxi_req ”>`. Nous avons vu dans la partie 5.3.2 que le taxi ne peut pas utiliser l'opération `read` pour trouver un passager, car le tuple reste présent dans l'espace de tuple du passager une fois l'opération terminée. Dans ce chapitre nous présentons l'opération `take` [39] qui pallie à cette limitation et permet à une entité de retirer un tuple de l'espace de tuples d'une autre entité.

Dans les chapitres précédents, la géométrie intervenait dans la description des configurations géométriques des interaction physiques. Dans ce chapitre, la géométrie intervient toujours dans la détection des interactions physiques, mais aussi au niveau du protocole de l'opération `take`. En effet, nous verrons que le protocole de l'opération `take` recours à une contrainte géométrique afin de minimiser le nombre d'opérations non-atomiques.

Dans la partie suivante, nous présentons le principe de fonctionnement de l'opération `take`, qui est principalement basée sur un protocole d'*atomic commitment*. Dans la deuxième partie nous détaillons le protocole de l'opération. Nous verrons notamment que ce protocole recourt à une contrainte géométrique pour garantir l'atomicité de l'opération `take`. Dans la troisième partie, nous évaluons l'impact de la contrainte géométrique sur le taux de défaillances de l'opération. Dans la quatrième partie, nous présentons comment restaurer la cohérence au niveau de l'application. Finalement, la cinquième partie présente la réalisation de l'opération `take`.

8.1 Principe de fonctionnement

L'opération `take(m)` renvoie un tuple correspondant au motif m et dont la forme est en intersection avec la zone d'adressage utilisée par l'opération. Le tuple renvoyé

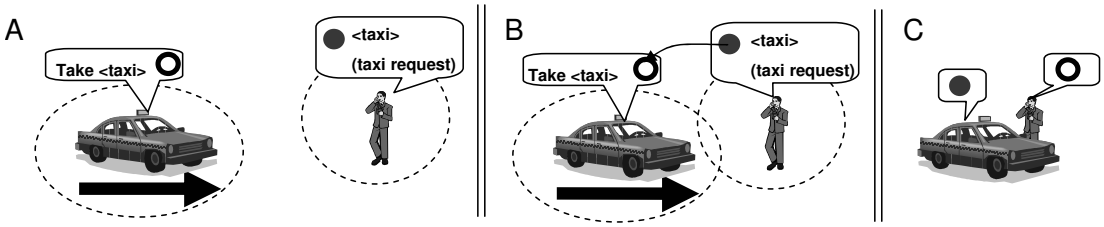

FIG. 8.1 – Utilisation de l'opération `take` pour réaliser l'application du taxi

est également retiré de l'espace de tuples de l'entité qui le porte. Si aucun tuple n'est disponible, l'opération reste bloquée. Tout comme l'opération `read`, l'opération `take` permet de synchroniser les applications sur des rencontres entre entités. Cependant, avec l'opération `take`, un tuple ne peut être impliqué que dans une seule rencontre. Dans le cas de l'application du taxi, ceci signifie que le piéton ne peut pas arrêter plusieurs taxis avec son seul tuple. Nous présentons sur la figure 8.1 l'utilisation de l'opération `take` pour réaliser l'application du taxi.

La sémantique de l'opération `take` est celle du passage d'un jeton. Le programme exécuté par une entité R est bloqué sur la découverte d'un jeton et une entité J possède un jeton. Lorsque R rencontre J, J passe à R son jeton et le programme de R est débloqué. Une fois le passage de jeton terminé, J ne possède plus le jeton. Le jeton ne peut donc être impliqué que dans une seule synchronisation physique. R peut publier à son tour le jeton qu'elle a reçu de J. Cependant, nous considérons alors qu'elle publie un nouveau jeton, car celui-ci est associé à une nouvelle entité.

Afin d'éviter la duplication et la perte du tuple lors du transfert d'une entité à l'autre, l'opération `take` doit être atomique. Dans la suite de cette partie nous étudions plus en détail l'atomicité de l'opération. Dans la première partie nous montrons la nécessité de l'atomicité. Dans la seconde partie, nous présentons pourquoi l'opération ne peut pas être systématiquement atomique dans le contexte de la programmation spatiale.

8.1.1 Atomicité

Une opération atomique s'exécute intégralement ou pas du tout. Si une opération atomique repose sur des états intermédiaires pendant son exécution, alors, dans le cas d'une défaillance l'opération doit restaurer l'état initial précédent l'opération. Dans l'exemple du taxi, l'atomicité du transfert du tuple implique que :

- soit le tuple à bien été transféré et l'opération est débloquée sur le taxi ;
- soit le tuple n'a pas été transféré et l'opération est toujours bloquée sur le taxi.

Lorsque l'opération s'arrête dans un état intermédiaire, l'opération n'est pas atomique : le tuple peut être soit dupliqué soit perdu. Dans l'exemple du taxi :

- si le tuple est dupliqué, alors le taxi s'est bien arrêté pour prendre le piéton. Cependant, le piéton possède toujours le tuple dans son terminal et peut arrêter d'autres taxis.

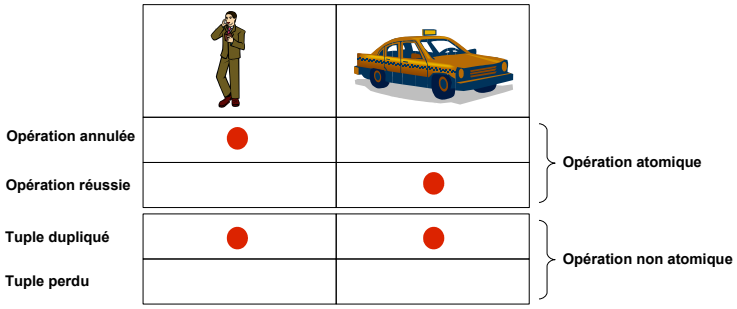

FIG. 8.2 – L'opération `take` peut se terminer dans quatre états différents

– si le tuple est perdu, le taxi n'a pas reçu le tuple et ne s'est pas arrêté. De son côté le piéton n'a plus de tuple dans son terminal et ne pourra donc plus arrêter de taxi.

Ces quatre cas de figure sont illustrés sur la figure 8.2.

L'opération `take` doit également être atomique de manière spatiale. Une opération `take` commence lorsque deux entités se rencontrent et n'a de sens que tant que ces deux entités restent co-localisées. L'atomicité spatiale garantit que l'opération commence et termine alors que les entités sont co-localisées dans une zone donnée. L'opération ne peut pas être divisée entre deux rencontres. Si l'atomicité spatiale n'est pas garantie, nous pouvons avoir le scénario suivant : lorsque le taxi rencontre le piéton une opération `take` est démarrée, mais le temps de communication disponible n'est pas suffisant pour terminer l'opération. L'opération est suspendue dans un état intermédiaire et se terminera lorsque le taxi rencontrera de nouveau le piéton, ce qui peut ne jamais arriver. L'atomicité spatiale évite de telles situations.

8.1.2 L'atomicité ne peut pas être systématiquement garantie

L'opération `take` est implémentée avec un protocole d'*atomic commitment*. L'opération `take` est exécutée sur des interfaces réseau sans fil qui peuvent perdre des messages. Gray à montré dans [22] que, en présence de pertes de messages, il n'existe aucun protocole de longueur fixe qui solutionne l'*atomic commitment*. Ainsi, un protocole garantissant que l'opération `take` est atomique ne peut pas être de longueur fixe, il peut nécessiter un nombre de messages non borné. L'opération `take` est exécutée par des entités mobiles qui ont une portée de communication limitée. Le temps de communication disponible pour terminer un `take` est donc borné. Dans ce contexte, seuls des protocoles de longueur fixe peuvent être utilisés. Par conséquent, dans le contexte de la programmation spatiale, aucun protocole ne peut garantir que l'opération `take` soit atomique.

Une opération `take` qui n'est pas atomique est une défaillance. Les erreurs qui provoquent ces défaillances sont les pertes de messages, qui sont provoquées par les phénomènes d'ombrage, de surcharges réseau, de collisions, de perturbations électro-

magnétiques...De plus, dans le contexte de la programmation spatiale des messages peuvent également être perdus simplement parce que les entités sont trop éloignées pour pouvoir communiquer.

Nous ne pouvons pas garantir qu'il n'y aura aucune défaillance, mais nous pouvons essayer de les réduire. Dans la partie suivante nous présentons un protocole pour l'opération `take` qui réduit le nombre d'opérations non-atomiques. Nous présentons également comment prendre en compte les défaillances résiduelles au niveau de l'application.

8.2 Protocole

Dans cette partie nous présentons le protocole de l'opération `take`. Ce protocole est basé sur le *two-phases commit protocol* (2PC). Si le protocole se termine, alors il garantit que l'opération est atomique. Si des messages sont perdus durant le protocole, alors le protocole ne garantit pas que l'opération est atomique.

Dans cette partie nous commençons par présenter le protocole 2PC. Nous poursuivons par une adaptation du protocole 2PC à la programmation spatiale. Nous terminons en présentant comment réduire le nombre d'opérations non-atomiques avec une contrainte géométrique et un mécanisme de réémission.

8.2.1 Le two-phases commit protocol

Le protocole 2PC est un protocole bien connu qui a été proposé pour résoudre le problème de l'atomicité d'une transaction dans une base de données. Le protocole 2PC repose sur un coordinateur et des participants. Pendant la première phase du protocole, le coordinateur demande à chaque participant de la transaction de voter. Si un participant vote "oui", alors il est prêt à valider la transaction. Autrement dit, il est dans un état où il peut encore restaurer son état initial. Si tous les participants votent "oui", alors la première phase est terminée et le coordinateur sauve un enregistrement de confirmation. Une fois l'enregistrement sauvé, la seconde phase commence et le coordinateur diffuse aux participants un message de confirmation de transaction. Si l'un des participants vote "non" pendant la première phase, alors le coordinateur diffuse un message d'annulation de la transaction aux participants et ceux-ci restorent leur état initial.

Le protocol 2PC tolère des pertes de messages et des défaillances des participants. Si une erreur arrive pendant la première phase, alors le coordinateur peut effectuer des réémissions et finalement annuler la transaction. Si une erreur se produit dans la seconde phase, le coordinateur doit réémettre le message de confirmation de transaction jusqu'à ce que chaque participant ait acquitté la confirmation de transaction. Ainsi, le nombre de messages nécessaires à la terminaison du protocole 2PC peut être non borné. Par conséquent, le protocole 2PC ne peut pas être utilisé pour l'opération `take`, il doit être adapté.

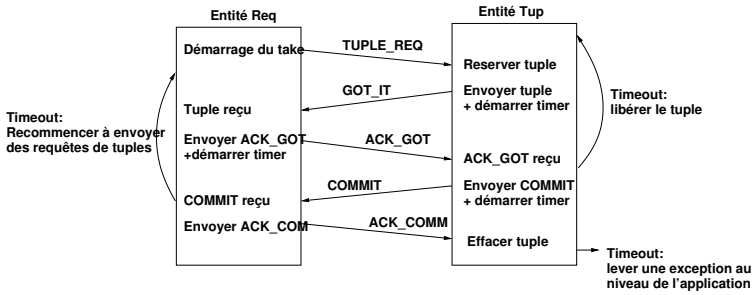

FIG. 8.3 – Protocole de l'opération `take`

8.2.2 Adaptation du protocole 2PC pour l'opération take

Nous présentons maintenant une adaptation du protocole 2PC pour réaliser l'opération `take`. Deux entités participent à l'opération `take` : l'entité *Req* qui exécute l'opération et l'entité *Tup* qui possède le tuple. Dans un premier temps, nous présentons le principe de fonctionnement du protocole. Dans un second temps, nous présentons les défaillances du protocole qui sont dues aux pertes de messages.

8.2.2.1 Principe

Le protocole compte deux participants qui sont les deux entités qui participent à l'opération `take`. L'entité *Tup* est également le coordinateur. Pendant la première phase du protocole, *Tup* retire le tuple de son espace de tuples et le réserve pour qu'il ne soit plus disponible pour d'autres opérations. Ensuite, *Tup* envoie le tuple à *Req*. Pendant la seconde phase, *Tup* envoie un message de confirmation à *Req* et efface définitivement le tuple. Lorsque *Req* reçoit le message de confirmation, il débloque l'opération `take` et renvoie le tuple à l'application.

Le protocole de l'opération `take` compte quatre messages (cf. figure 8.3) : `GOT_IT`, `ACK_GOT`, `COMMIT` et `ACK_COMM`. Au démarrage de l'opération, l'entité *Req* diffuse régulièrement une demande de tuples aux entités voisines (`TUPLE_REQ`). Ce message contient à la fois le motif et la zone d'adressage. Lorsque l'entité *Tup* reçoit la demande de tuple le protocole de l'opération commence : *Tup* réserve un tuple pour cette opération et en envoie une copie à *Req* dans le message `GOT_IT`. Ensuite, l'entité *Req* acquitte la réception du tuple en envoyant le message `ACK_GOT`. Lorsque l'entité *Tup* reçoit `ACK_GOT` la première phase est finie. Pendant la seconde phase, *Tup* envoie le message `COMMIT` à *Req* qui l'acquitte avec un message `ACK_COMM`. Lorsque *Tup* reçoit `ACK_COMM`, elle efface le tuple.

8.2.2.2 Défaillances du protocole

Chacun des quatre messages du protocole peut être perdu. Pendant la première phase du protocole, si le message `GOT_IT` ou le message `ACK_GOT` sont perdus alors l'opération est annulée et le système reste dans un état cohérent. Il n'y a pas de défaillance. Si le

message `GOT_IT` est perdu, *Req* continue d'envoyer des demandes de tuple. De son côté, *Tup* ne peut pas recevoir le message `ACK_GOT`, donc, *Tup* annule l'opération et libère le tuple après un timeout donné. De la même manière, si le message `ACK_GOT` est perdu, alors *Tup* annule l'opération et libère le tuple.

Si des messages sont perdus pendant la seconde phase du protocole, l'opération échoue. Le système est alors dans l'un de ces deux états incohérents :

- Si le message `COMMIT` est perdu, alors l'opération `take` est toujours bloquée sur l'entité *Req* et celle-ci recommence à diffuser des demandes de tuples après un timeout donné. Sur l'entité *Tup*, le tuple est toujours réservé.
- Si le message `ACK_COMM` est perdu, l'entité *Req* à reçu le message `COMMIT` et l'opération `take` a été débloquée. Sur l'entité *Tup*, le tuple est toujours réservé. Après un timeout donné l'entité *Tup* lève une exception pour avertir l'application que l'opération a échoué. Nous étudions dans la partie 8.4 la prise en compte de cette défaillance au niveau de l'application.

8.2.3 Réduction du nombre de défaillances

Les défaillances sont dues aux pertes de messages. Nous distinguons deux types de fautes qui provoquent ces pertes de messages : i) les erreurs réseau ; ii) les entités sont trop loin pour communiquer. La première catégorie de faute peut être prise en compte par un mécanisme de réémission standard. La seconde catégorie de faute est propre à la programmation spatiale. Dans ce cas, soit les entités sont en train de s'approcher l'une de l'autre et le protocole a démarré dans une zone où les communications sont trop peu fiables, soit les entités s'éloignent l'une de l'autre et le protocole n'a pas eu assez de temps pour se terminer. Pour prévenir ce type de situation le protocole utilise une contrainte géométrique.

8.2.3.1 Ajout de réémissions

Ajouter des réémissions dans un protocole est une technique habituelle pour tolérer des fautes transitoires, telles que des perturbation électromagnétiques. Dans notre protocole les messages critiques sont les messages `COMMIT` et `ACK_COMM`. Si l'un d'entre eux est perdu l'opération échoue. Nous ajoutons donc des réémissions pour le message `COMMIT`. Si l'entité *Tup* ne reçoit pas le message `ACK_COMM`, le message `COMMIT` ou le message `ACK_COMM` a été perdu, un message `COMMIT` est alors renvoyé.

Un mécanisme de réémission doit être utilisé avec précautions, parce qu'il peut violer la propriété d'atomicité spatiale de l'opération. En ajoutant des réémissions dans le protocole nous augmentons le temps de communication du protocole, ce qui augmente la zone dans laquelle les entités peuvent se déplacer et terminer le protocole. En effet, si nous permettons un nombre de réémissions non borné, alors le temps de communication du protocole peut être également non borné. Dans ce cas, la situation où une opération `take` commence lors d'une première rencontre entre deux entités et se termine lors d'une seconde rencontre peut se produire. Ce type de situation viole la propriété d'atomicité spatiale.

Les paramètres critiques d'un mécanisme de réémission sont le nombre de réémissions et le délai entre deux réémissions. Ces deux paramètres doivent être ajustés en fonction du temps de communication disponible entre deux entités, qui dépend lui-même de la vitesse des entités, de leur trajectoire et de leur portée de communication. Ils sont donc difficiles à déterminer. Une solution a été proposée dans [61] pour un problème similaire, qui consiste en la découverte automatique d'entités mobiles par l'intermédiaire de messages de présence.

8.2.3.2 Utilisation d'une contrainte géométrique

Si les entités *Tup* et *Req* ne disposent pas d'un temps de communication assez grand alors l'opération `take` peut ne pas avoir assez de temps pour se terminer, ce qui implique que les messages `COMMIT` ou `ACK_COMM` peuvent être perdus. Ces messages peuvent aussi être perdus si l'opération commence à la frontière de la zone de communication, où les communications sont les moins fiables.

Pour prévenir les situations précédentes, nous proposons d'utiliser une contrainte géométrique qui définit une zone sûre dans laquelle une opération `take` peut commencer. Cette contrainte géométrique définit une distance seuil. Une opération `take` ne peut commencer que si la distance qui sépare les entités *Tup* et *Req* est inférieure à ce seuil (cf. figure 8.4). Ainsi, l'opération ne commence pas à la frontière de la zone de communication et un temps de communication minimum est disponible lorsque l'opération commence. Par exemple, nous considérons deux entités avec une portée de communication de 300 mètres et une vitesse de 13 mètres par seconde. Le temps de communication le plus court se produit lorsque les entités se déplacent dans des directions opposées. Dans ce cas là, si le seuil est de 50 mètres, alors la longueur du chemin le plus court pour quitter la zone de communication est de 250 mètres, lorsque l'opération démarre. Ainsi, le temps de communication minimum est de $(300 - 50)/(13 * 2) = 9.6s$.

Avec une contrainte géométrique, un `take` ne peut commencer qu'à l'intérieur de la zone définie par la contrainte. Cette zone de démarrage représente donc la zone à l'intérieur de laquelle l'opération `take` peut adresser une opération. Ainsi, si le programmeur définit une zone d'adressage dépassant la zone de démarrage, la zone d'adressage sera circonscrite par la zone de démarrage.

8.3 Évaluation de la contrainte géométrique

Dans la partie précédente nous avons présenté le protocole de l'opération `take`. Ce protocole utilise une contrainte géométrique et un mécanisme de réémission pour réduire le nombre de défaillances. Dans cette partie nous étudions l'impact de ces mécanismes sur le taux de défaillance du protocole.

8.3.1 Présentation de la simulation

Pour des raisons pratiques nous évaluons le protocole avec le simulateur NS [47] et l'extension pour les réseaux sans fil. Au cours de la simulation les entités communiquent

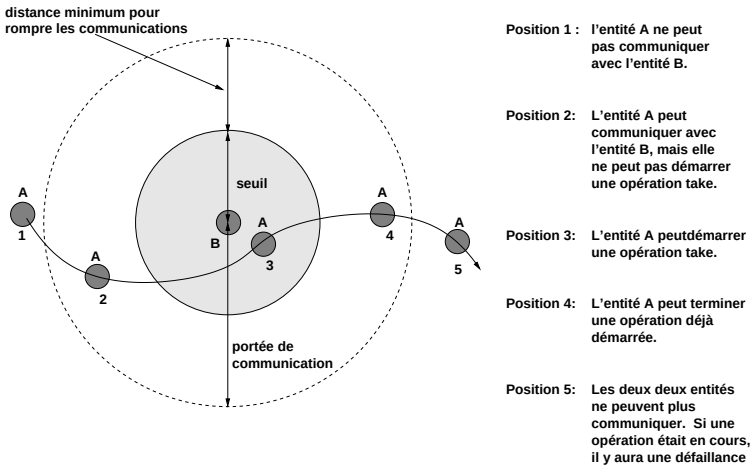

FIG. 8.4 – Une contrainte géométrique garantit un temps de communication minimum entre les entité A et B

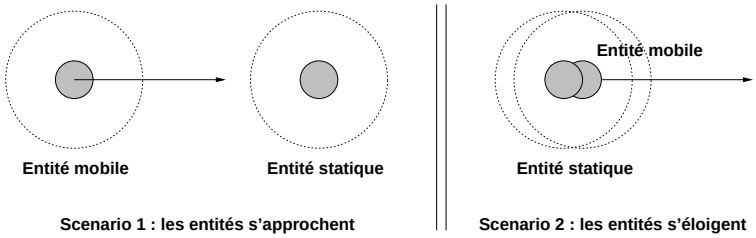

FIG. 8.5 – Scénario utilisés dans la simulation

avec des interfaces de communication WiFi. Pour simuler les pertes de paquets, nous utilisons le modèle de propagation *shadowing model*. Le paramétrage est choisi pour simuler un environnement urbain.

Pour évaluer l'impact de la contrainte géométrique nous devons identifier les situations où la contrainte agit réellement sur le taux de défaillance du protocole. Nous avons identifié deux situations :

- l'opération est bloquée et les entités s'approche l'une de l'autre. L'opération démarre donc à la frontière de la zone de communication, où les communications sont le moins fiables.
- les entités s'éloigne l'une de l'autre, les entités peuvent communiquer mais le temps de communication restant peut être trop court pour terminer le protocole.

Nous proposons deux scénarios de simulation qui sont conçus pour reproduire ces situations. Nous n'utilisons pas un scénario de simulation basé sur des mouvements aléatoires, car les deux situations précédentes se produisent rarement dans de tels scénarios. La figure 8.5 présente les deux scénarios utilisés dans la simulation.

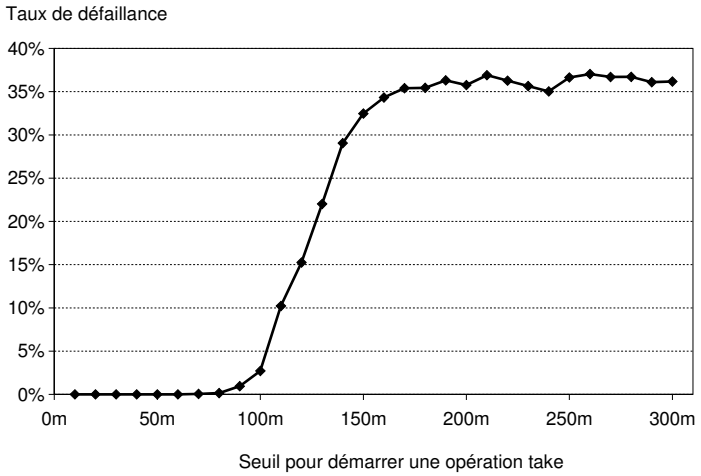

FIG. 8.6 – Taux de défaillance sans réémission. Scénario 1 : les entités se rapprochent

8.3.2 Premier scénario : les entités s'approchent

L'objectif de cette première simulation est d'évaluer l'impact de la contrainte géométrique sur le taux de défaillance du protocole lorsque les entités qui participent à l'opération s'approchent.

Pour cette simulation nous considérons un scénario similaire au scénario de l'application du taxi. Nous avons une entité mobile qui cherche un tuple (le taxi) et une entité statique (le piéton) qui possède un tuple. La vitesse de l'entité mobile est de 50 kilomètres heure. Une simulation comprend 2000 passages. Au début de chaque passage, le taxi est localisé à 500 mètres du passager. Le taxi exécute une opération `take` qui reste bloquée parce qu'il est trop loin du passager. Ensuite, le taxi commence à se déplacer vers le piéton (cf. figure 8.5 gauche). Une fois l'opération `take` terminée le taxi retourne à son point de départ. Pour cette première simulations il n'y a pas de réémission dans le protocole.

Pendant la première simulation il n'y a pas de contrainte géométrique. L'opération peut donc commencer dès que les entités peuvent communiquer. Ensuite, nous réexécutons la simulation avec un seuil de 290 mètres. Cette fois-ci, l'opération `take` ne peut commencer que lorsque la distance qui sépare les deux entités est inférieure à 290 mètres. La simulation est réexécutée plusieurs fois avec un seuil variant de 290 à 10 mètres.

Pour chaque simulation, nous avons n_r opérations réussies et n_d défaillances. Nous voulons obtenir le taux de défaillance t_d qui représente le rapport du nombre de défaillances par rapport au nombre total d'opérations essayées : $t_d = n_d/(n_r + n_d)$. Les résultats sont présentés sur la figure 8.6. Sur la courbe, le point correspondant à un seuil de 300 mètres représente le taux de défaillances sans contrainte géométrique, puisque la portée théorique de WiFi est de 300 mètres.

8.3.2.1 Analyse des résultats

Pour cette simulation nous observons que le taux de défaillance est nul pour un seuil inférieur ou égal à 60 mètres. Ceci est un résultat important car nous n'avons pas utilisé de réémission. Ainsi, en réglant le seuil à 60 mètres, pour cette simulation, les opérations sont toutes atomiques sans aucune réémission. De plus, l'absence de réémission économise de l'énergie et de la bande passante réseau.

En réglant le seuil entre 300 mètres et 170 mètres le taux de défaillance est compris entre 34 et 35 pour cent. Puisque un seuil de 300 mètres est équivalent à ne pas utiliser de seuil du tout, nous en concluons que la contrainte géométrique n'a aucun effet sur le taux de défaillance pour les seuils supérieurs à 170 mètres. En pratique nous remarquons que les entités communiquent difficilement au-delà de 200 mètres. C'est pourquoi la contrainte n'agit sur le taux de défaillances qu'avec un seuil inférieur à 160 mètres.

Cette simulation montre que la contrainte géométrique peut grandement améliorer la fiabilité du protocole. Sans aucune contrainte, le taux de défaillance est compris entre 34 et 35 pour cent. En réglant le seuil à 60 mètres, nous obtenons un taux de défaillance nul. En pratique nous ne pouvons pas garantir un taux de défaillance nul. Pendant une simulation, 2000 opérations sont exécutées. Nous avons donc un taux de défaillance inférieur à $1/2000 = 0,05\%$. Avec une simulation plus longue, nous avons évalué que le le taux défaillance est inférieur à 0.005 %. Nous avons arrêté la simulation après 20000 opérations réussies.

8.3.2.2 Impact du mécanisme de réémission

Pour étudier l'impact du mécanisme de réémission, nous ajoutons des réémissions pour le message `COMMIT`. Le nombre de réémissions est réglé à deux, ce qui implique qu'au maximum trois messages `COMMIT` sont envoyés par opération. Nous conservons le premier scénario et exécutons la même série de simulation. Les résultats sont présentés sur la figure 8.7.

Pour déterminer le nombre de réémissions nous exécutons la simulation avec différents nombres de réémissions. Au-delà de deux réémissions, le taux de défaillance ne diminue plus. De plus, le nombre de réémissions doit être minimisé afin de limiter la zone dans laquelle l'opération peut se terminer et de conserver la propriété d'atomicité spatiale. Dans le cas présent, nous avons deux réémissions espacées de 0.5 seconde, ce qui augmente de temps de communication d'au maximum une seconde. Pendant une seconde, une entité dont la vitesse est de 50 kilomètres heure peut se déplacer n'importe où dans un cercle de 14 mètres de rayon.

Cette simulation montre que le taux de défaillance est nul avec un seuil inférieur à 100 mètres. Le mécanisme de réémissions augmente donc la limite supérieure du seuil donnant un taux de défaillance nul. Précédemment nous avions une limite de 60 mètres. De plus, lorsque le seuil est compris entre 170 et 300 mètres, le taux de défaillance est aussi plus faible que précédemment et compris entre 12 et 14 pour cent.

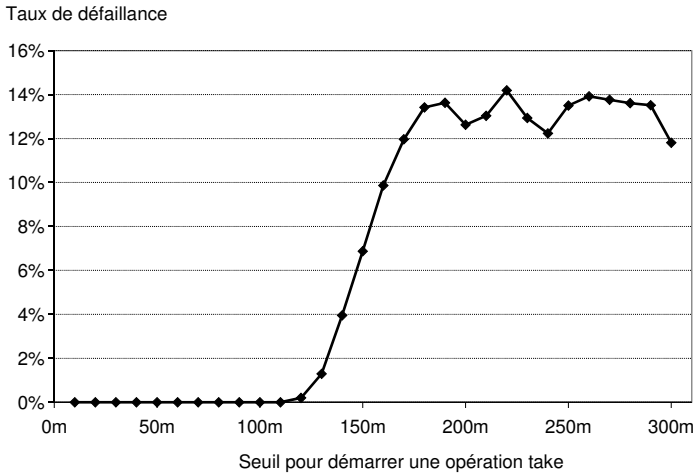

FIG. 8.7 – Taux de défaillance avec deux réémissions. Scénario 1 : les entités se rapprochent.

8.3.3 Second scénario : les entités s'éloignent

Nous considérons le premier scénario comme étant favorable. En effet, le temps de communication est maximisé car les entités se rapprochent. Nous voulons maintenant déterminer quel seuil donne le taux de défaillance le plus faible lorsque les entités s'éloignent. Ce scénario correspond au pire cas, car le temps de communication est cette fois-ci minimisé.

Dans ce second scénario nous avons toujours une entité statique et une entité mobile qui recherche un tuple. Au début d'un passage, les entités sont situées à la même position et l'entité mobile commence à s'éloigner de l'entité statique (cf. figure 8.5 droite). Pendant la première simulation, les opérations `take` commencent lorsque la distance entre les deux entités est inférieure à 10 mètres. Pendant la seconde simulation, les opérations ne commencent que lorsque la distance entre les deux entités est comprise entre 10 et 20 mètres, et ainsi de suite. Pour ces simulations nous gardons le nombre de réémissions à deux pour le message `COMMIT`. Les résultats sont présentés sur la figure 8.8.

Ces simulations montrent que si les opérations sont démarrées entre 0 et 90 mètres toutes les opérations sont atomiques. Entre 160 et 210 mètres, nous avons un taux d'erreur de 100 pour cent. Cependant, dans cette zone nous avons très peu d'opérations qui commencent. Pour 2000 passages, seules 10 opérations commencent. Dans cette zone, les communications sont si peu fiables qu'il est très difficile de commencer une opération. Par conséquent, il est encore plus difficile de terminer une opération. Au-delà de 210 mètres aucune opération ne commence.

Pour ce scénario, le seuil maximum garantissant un taux de défaillance nul est inférieur au seuil obtenu pour le premier scénario. Ce scénario correspondant au pire cas,

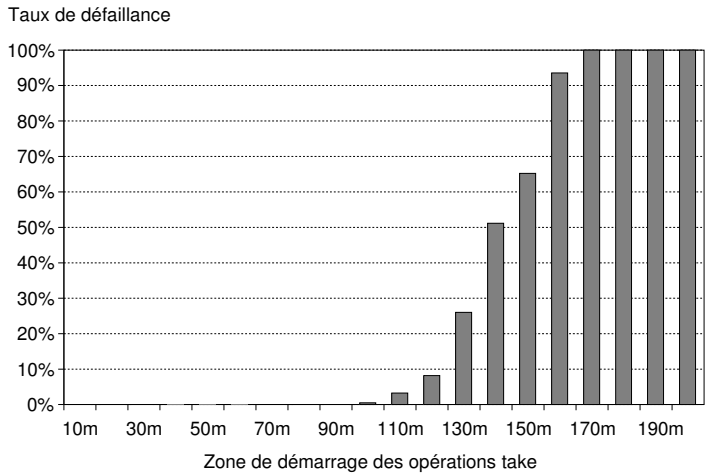

FIG. 8.8 – Taux de défaillance en fonction de la zone où les opérations sont démarrées, avec 2 réémissions. Scénario 2 : les entités s'éloignent.

ainsi, pour un scénario général dans lequel la trajectoire des entités n'est pas connue par avance, le seuil doit être réglé à 90 mètres.

8.3.4 Bilan

Le protocole de l'opération `take` est une variation du protocole 2PC. Dans le contexte de la programmation spatiale nous ne pouvons pas garantir le toutes les opérations seront atomiques. Cependant, en utilisant une contrainte géométrique le taux de défaillance peut être grandement réduit. Pour les simulations précédentes, nous pouvons obtenir un taux de défaillance nul. Cependant, dans la pratique le protocole ne peut garantir un taux de défaillance nul. Nous avons exécuté des simulations comprenant jusqu'à 20000 opérations réussies et aucune opération non-atomique. Nous considérons donc que le taux de défaillance est inférieur à 0,005 %.

A lui seul le mécanisme de réémission ne peut réduire le taux de défaillance à zéro. Cependant, combiné à la contrainte géométrique, le mécanisme de réémission augmente le seuil maximum garantissant un taux de défaillance nul. Par exemple, dans le premier scénario le seuil maximum est de 60 mètres sans réémission et 100 mètres avec deux réémissions.

Lorsque la trajectoire des entités n'est pas connue par avance, le seuil doit être choisit en fonction du second scénario, qui représente le scénario pire cas.

8.4 Prise en compte des défaillances au niveau de l'application

Dans la partie précédente, nous avons évalué comment une contrainte géométrique peut réduire le nombre d'opérations non-atomiques. Les défaillances résiduelles plongent le système composé des deux entités impliquées dans l'opération dans un état incohérent. Dans cette partie, nous étudions la restauration de la cohérence au niveau de l'application. Nous considérons de nouveau l'entité *Tup* qui possède un tuple et l'entité *Req* qui exécute une opération `take`.

8.4.1 Détection des défaillances

Nous avons vu qu'en cas de défaillance, le système peut entrer dans deux états incohérents différents (cf. 8.2.2.2).

Seule l'entité *Tup* peut détecter que le système est dans un état incohérent. Dans chacun des états incohérents, l'entité *Tup* est bloquée dans la seconde phase du protocole avec le tuple toujours réservé. Après un nombre de réémissions donné, *Tup* déclare son état local comme étant incohérent et détecte donc que le système est dans un état incohérent. *Req* ne peut pas détecter l'incohérence, car dans chacun des états globaux incohérents son état local est cohérent : soit elle attend un tuple, soit elle a reçu le tuple et débloqué l'opération.

8.4.2 Restauration de la cohérence

Puisque l'incohérence du système est détectée par l'entité *Tup*, c'est cette même entité qui doit restaurer la cohérence. Lorsque l'entité *Tup* détecte une défaillance, elle sait qu'une autre entité voulait l'un de ses tuples et que l'opération `take` a échoué. Ainsi, elle doit décider si le tuple doit être effacé ou libéré pour d'autres opérations. Pour prendre cette décision, *Tup* a besoin d'informations décrivant la situation physique courante.

8.4.2.1 Informations nécessaires pour restaurer la cohérence

Au niveau du protocole, l'entité *Tup* sait seulement que le message `COMMIT` ou le message `ACK_COMM` ont été perdus. Si le message `COMMIT` a été perdu, alors l'entité *Req* n'a pas validé l'opération et a recommencé à diffuser des requêtes de tuples. Si le message `ACK_COMM` a été perdu, alors l'entité *Req* a validé l'opération et *Tup* doit effacer le tuple. *Tup* ne peut savoir lequel de ces deux messages a été perdu, elle ne peut donc pas prendre de décision pour restaurer la cohérence.

Seul le développeur de l'application dispose des informations suffisantes pour restaurer la cohérence lorsqu'une opération échoue. Une application est composée d'une série d'opérations qui inclue des opérations `take`. Généralement un tuple est destiné à une opération bien précise. Ainsi, grâce à sa connaissance de l'application, le développeur d'applications peut prévoir pour chaque tuple comment restaurer la cohérence. S'il

n'est pas possible de prévoir par avance comment restaurer la cohérence, l'aide ou la notification de l'utilisateur peuvent être requis.

Nous proposons un mécanisme d'exception pour restaurer la cohérence au niveau de l'application. Ce mécanisme associe à chaque tuple une fonction d'exception qui est exécutée par l'entité qui porte ce tuple lorsqu'il est impliqué dans une défaillance.

8.4.2.2 Exemple

Dans le cas de l'application de taxi, le taxi correspond à l'entité *Req* et le piéton à *Tup*. Lorsque l'opération `take` exécutée par le taxi échoue, le taxi était probablement trop loin du piéton lorsqu'il a commencé l'opération.

Le taxi ne peut pas détecter les défaillances, car dans chacun des états globaux incohérents son programme est dans un état cohérent. Si le message `COMMIT` venant du piéton a été perdu, le taxi a annulé l'opération, a recommencé à diffuser des requêtes de tuples et ne s'est pas arrêté. Si le message `ACK_COMM` a été perdu, le taxi a validé l'opération et s'est arrêté. Si le chauffeur ne voit personne attendant un taxi, il décide de partir et relance une nouvelle requête de tuple. Ici, bien que le programme du taxi soit dans un état cohérent, l'état global incohérent se manifeste par l'abscence du piéton. Si le chauffeur voit le piéton, celui-ci monte dans le taxi et le taxi s'en va.

Du point de vue du piéton, lorsqu'une défaillance survient, le tuple doit être libéré si aucun taxi ne s'est arrêté, afin de trouver un autre taxi. Si un taxi s'est arrêté, le taxi à reçu le message `COMMIT` et le message `ACK_COMM` a été perdu, le tuple doit être effacé du terminal du piéton. Ainsi, pour restaurer la cohérence au niveau du piéton, nous devons savoir si un taxi s'est arrêté ou non. La collaboration du piéton est donc nécessaire. Par conséquent, l'application doit notifier le piéton qu'une défaillance a eu lieu et lui demander s'il voit un taxi arrêté pour lui. Ici, la fonction d'exception pose juste une question au piéton.

Pour simplifier les développements et pour les applications tolérant les pertes de tuples, le développeur peut simplement décider d'effacer systématiquement les tuples impliqués dans des opérations échouées. Pour les applications ne tolérant aucune perte de tuple, le développeur peut décider de libérer tous les tuples, avec le risque d'avoir des tuples dupliqués.

8.5 Réalisation

Nous avons intégré l'opération `take` au sein de SPREAD. Dans cette partie nous présentons tout d'abord l'implémentation de l'opération. Nous évaluons ensuite sa réactivité.

8.5.1 Implémentation

L'implémentation de l'opération `take` suit le protocole présenté précédemment. Actuellement, l'implémentation repose sur des Pocket PC équipés de récepteur GPS. L'entité *Req* qui exécute l'opération `take` inclut donc dans ses requêtes de tuples sa position

géographique. Ainsi, chaque entité qui reçoit ces requêtes peut déterminer si la distance qui la sépare de l'entité *Req* est inférieure au seuil de démarrage. Le mécanisme d'exception utilisé par l'opération `take` exploite le mécanisme d'exception fourni par Java.

8.5.2 Évaluation de la réactivité

Nous avons évalué la réactivité de l'opération `take`. Le temps de d'exécution moyen d'une opération est de 50 millisecondes lorsqu'aucune réémission n'est nécessaire. Ce temps de réaction est adapté au profil de mobilité visé par la programmation spatiale. En effet, les entités les plus rapides sont les voitures, or dans un scénario urbain leur vitesse maximum est de 50 kilomètres heure. Ainsi, une voiture parcourt en moyenne 70 centimètre pendant l'exécution d'une opération. Du point de vue de l'utilisateur l'opération est donc presque instantanée.

Le protocole emploie deux réémissions au maximum, espacées de 0,5 s. Ainsi, si des réémissions sont nécessaires à la terminaison de l'opération, le temps maximum est d'environ 1,1 s. Dans ce cas là, une voiture parcourt environs 14 mètres entre le début et la fin de l'opération. Cette distance est bien plus importante que la précédente et doit être prise en compte lors de la programmation des applications. Par exemple, dans le cas de l'application du taxi, la zone d'adressage du taxi est située en avant du taxi. Cette zone doit être suffisamment grande afin de garantir que le taxi n'a pas dépassé le piéton lorsque l'opération se termine.

8.6 Bilan

8.6.1 Travaux apparentés

Les protocoles d'*atomic commitment* ont été fortement étudiés dans le cadre des systèmes distribués [5, 49]. Cependant, ces travaux supposent que le réseau de communication ne perd ni ne duplique aucun message. Dans notre cas, des messages peuvent être perdus, en raison de la trop grande distance séparant les entités par exemple.

Dunham et al [15] ont proposé le modèle de transaction Kangaroo pour l'informatique mobile. Dans ce modèle, chaque entité mobile est connectée au réseau global par l'intermédiaire d'un point d'accès. La programmation spatiale ne repose pas sur un réseau global et doit donc prendre en compte le temps de communication limité. De plus, avec le modèle Kangaroo une transaction n'est pas spatialement atomique, elle peut commencer à une position donnée et se terminer à une autre position.

LIME [45] propose un mécanisme de transaction entre entités mobiles équipées d'interfaces de communication à courte portée. LIME résout le problème de l'atomicité en contraignant la mobilité des entités. Ainsi, la transaction dispose systématiquement du temps nécessaire à la terminaison du protocole. Dans le contexte de la programmation spatiale, les entités sont considérées comme autonomes et nous ne pouvons pas contraindre leur mobilité.

8.6.2 Discussion

Dans ce chapitre nous avons présenté l'opération `take` qui permet à une entité de retirer un tuple porté par une autre entité. Du point de vue des interactions physiques, l'opération `take` nous permet de synchroniser les applications sur des rencontres entre entités. Cependant, contrairement à l'opération `read`, avec l'opération `take` un tuple ne peut être impliqué que dans une seule rencontre.

L'opération `take` doit être atomique. Cependant, nous avons montré que dans le contexte de la programmation spatiale, il est impossible de garantir l'atomicité de cette opération. L'opération repose sur l'utilisation d'une contrainte géométrique pour minimiser le nombre d'opérations non-atomiques. Cette contrainte définit une distance seuil au-delà de laquelle une opération ne peut démarrer. L'évaluation de cette contrainte nous a montré qu'elle permet de réduire le taux de défaillance de l'opération `take` à zéro au cours d'une simulation. Sans contrainte, lorsque le protocole comprend deux réémissions, le taux de défaillance de l'opération est compris entre 12 et 14 pour cent.

Finalement, nous avons présenté comment restaurer la cohérence au niveau de l'application. Pour cela, le développeur doit prévoir comment restaurer la cohérence pour chaque tuple, lorsque ce tuple participe à une opération échouée. Autrement dit, le développeur doit décider s'il faut libérer ou effacer le tuple.

Troisième partie

Géométrie et systèmes d'information

Chapitre 9

Géométrie et contexte

Dans les chapitres précédents nous avons étudié le rôle de la géométrie dans la programmation d'applications purement diffuses. Nous avons présenté un modèle de programmation qui permet au programmeur de décrire directement dans ses programmes les interactions physiques qui composent l'application ainsi que leur configuration géométrique. Les applications crées avec ce modèle reposent donc directement sur l'organisation géométrique de l'environnement physique et réagissent aux changements qui y sont apportés. Nous avons également présenté un mécanisme de contrainte géométrique qui permet d'améliorer la fiabilité des transferts atomiques de données entre deux entités.

Dans cette partie nous étudions la notion de contexte dans un système d'information et son utilisation pour la navigation dans le système [41]. L'espace physique est considéré comme un système d'information particulier. Nous verrons que la géométrie intervient aussi bien au niveau de la structure des systèmes d'information qu'au niveau de la construction du contexte.

Dans ce chapitre nous présentons donc en détail la notion de contexte, et particulièrement comment il peut être utilisé pour naviguer implicitement dans un système d'information. Nous commençons par présenter dans la première partie la notion de contexte au sein de l'espace physique. Nous y détaillons notamment comment utiliser la programmation spatiale pour réaliser une application contextuelle. Nous poursuivons, en décrivant comment utiliser le contexte pour naviguer dans un système d'information. Finalement, dans la troisième partie nous définissons formellement le contexte.

9.1 Contexte et espace physique

Dans cette partie nous présentons la notion de contexte. Nous commençons par introduire la notion de contexte en nous basant sur la définition de contexte présente dans les dictionnaires. Nous poursuivons par la présentation du contexte d'un utilisateur qui évolue dans l'espace physique. Nous terminons en présentant l'utilisation de la programmation spatiale pour programmer des applications contextuelles.

9.1.1 Notion de contexte

Initialement, la notion de contexte est avant tout une notion linguistique ou temporelle. Le *petit Larousse illustré* définit le contexte ainsi :

- ensemble du texte qui précède ou suit une phrase, un groupe de mots, un mot ;
- ensemble de circonstances qui accompagnent un événement : replacer un fait dans un contexte historique.

Nous pouvons extraire plusieurs propriétés de ces définitions. Tout d'abord le contexte est une notion *relative*, le contexte situe une entité référence par rapport à un ensemble plus global, comme un mot dans un texte ou un événement historique. Le contexte est un *ensemble* d'éléments. D'après le dictionnaire, ces éléments sont déterminés par une relation de proximité : les circonstances qui accompagnent un événement ; l'ensemble du texte qui suit et précède une phrase. Le contexte contient les éléments qui sont proches de l'entité référence. Par exemple, dans la phrase "le chat guette l'oiseau perché sur la branche", le contexte du mot "oiseau" est représenté par {"le chat guette", "perché sur la branche"}. Notons que les éléments de ce contexte sont ordonnés, le premier élément est localisé avant le mot référence et le second après.

En observant la notion de contexte historique ou de contexte textuel nous remarquons que la notion de contexte est avant tout une notion locale, définie par rapport à une entité de référence. Ainsi, de notre point de vue, la notion qui décrit le mieux ce qu'est le contexte est la notion de proximité : le contexte d'un passage de texte est composé des phrases proches ; le contexte d'une page Web est composé des pages qui ont un lien vers elle ; le contexte d'une personne est représenté par les bâtiments et personnes proches.

En plus de la notion de proximité, le contexte inclut également une notion de structure. Par exemple, dans l'espace physique le contexte est représenté par l'ensemble des entités physiques proches de l'utilisateur. Ici, la structure des éléments du contexte correspond à la structure géométrique des entités physiques. Le contexte d'une phrase est représenté par les phrases précédentes et suivantes. Ce contexte distingue donc les éléments en fonction de leur position relativement à la position de l'entité référence. La structure du contexte est représentée par les informations qui caractérisent la position des éléments du contexte relativement à l'entité de référence. Ces informations structurelles peuvent être représentées par la position des éléments relativement à la position de l'entité, par leur distance relative, par leur position angulaire... Lorsque qu'aucune information sur la structure des éléments n'est disponible, le contexte est représenté par les seuls éléments proches.

La notion de contexte n'est pas limitée à un texte ou un évènement historique, elle est également présente dans l'espace physique ou dans les systèmes d'information tels que le Web. Nous choisissons donc de généraliser la notion de contexte ainsi :

> *Le contexte est défini dans un espace composé d'éléments, comme un texte ou un système d'information. Dans cet espace le contexte d'une entité référence est composé des éléments qui sont proches de cette entité et de la structure de ces éléments.*

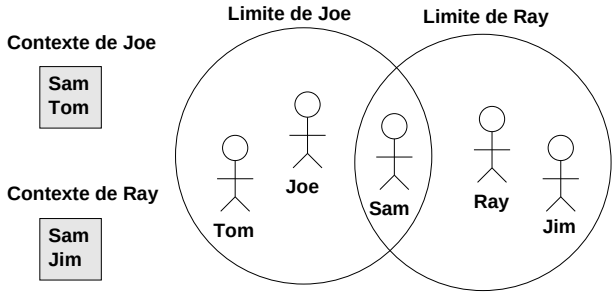

FIG. 9.1 – Contexte physique d'une personne

Par exemple dans l'espace physique, nous considérons que le contexte d'une personne est représenté par les personnes proches d'elle. Nous considérons également une personne comme proche d'une autre lorsque la distance qui les sépare est inférieure à une certaine limite. Si nous fixons la distance maximum à dix mètres, le contexte d'une personne est représenté par l'ensemble des personnes situées à moins de dix mètres d'elle (cf. figure 9.1).

9.1.2 Contexte physique

Dans l'espace physique le contexte de l'utilisateur est représenté par les entités physiques qui sont proches de lui. La programmation spatiale embarque sur les entités physiques les informations qui leur sont liées. Ainsi, du point de vue d'une application spatiale, le contexte physique de l'utilisateur est représenté par les informations qui sont associées aux entités physiques proches de lui et par leur structure géométrique.

Dans l'espace physique, le contexte est structuré selon l'organisation géométrique des informations. Nous pouvons par exemple utiliser la structure géométrique du contexte pour présenter le contexte sur le terminal de l'utilisateur d'une manière analogue à son organisation physique ; les informations situées devant l'utilisateur sont affichées en haut de l'écran, les informations à gauche de l'utilisateur sont affichées dans la moitié gauche de l'écran.

9.1.2.1 Éléments du contexte

Pour construire le contexte de l'utilisateur, nous définissons une zone d'adressage relativement à la position et à l'orientation de l'utilisateur. Le contexte de l'utilisateur contient l'ensemble des informations dont la forme est en intersection avec la zone d'adressage de l'utilisateur. Notons que le contexte de l'utilisateur ne doit contenir que des informations pertinentes. La forme des informations définit donc la zone de l'espace physique dans laquelle elles sont pertinentes pour l'utilisateur

Grâce à l'adressage géométrique, nous pouvons filtrer les informations présentes dans le contexte de l'utilisateur. Par exemple, nous pouvons définir un secteur dirigé vers l'avant de l'utilisateur, afin de restreindre le contenu du contexte de l'utilisateur et

ne lui présenter que des informations liées aux entités physiques situées devant lui.

9.1.2.2 Mise à jour du contexte

Le contexte physique de l'utilisateur évolue en fonction de ses déplacements. Nous devons donc mettre à jour les informations qu'il contient. Ces informations sont stockées sous forme de tuples. Grâce aux opérations `readOnce` et `lostOne` (cf. chapitre 7), nous pouvons surveiller facilement l'entrée ou le départ d'une information du contexte de l'utilisateur. Le programme qui surveille le contexte de l'utilisateur est donc similaire à celui exécuté par le chariot (cf. 7.2.1) et composé de deux processus : le premier surveille les arrivées de nouvelles informations ; le second surveille les départs d'informations.

Au niveau de l'application, les mises à jours de contexte peuvent être reflétées de différentes manières. Si l'application se contente de proposer à l'utilisateur le contenu de son contexte, alors le contexte peut être simplement mis à jour dynamiquement. Cependant, nous pouvons envisager que ces mises à jour du contexte représentent en elles-mêmes une information. Par exemple, nous pourrions mettre en évidence les derniers éléments ajoutés dans le contexte. Dans le cas d'un guide virtuel, une nouvelle information présente dans le contexte de l'utilisateur est équivalente à la présence d'une nouvelle entité à proximité de l'utilisateur. Nous pourrions également mettre en évidence les dernières informations sorties du contexte, avant de les effacer définitivement.

9.2 Navigation contextuelle

Dans la partie précédente nous avons introduit la notion de contexte en nous basant sur la définition traditionnelle du contexte. Nous avons vu plusieurs exemples de contexte, tels que le contexte d'un utilisateur naviguant sur le Web, le contexte d'un utilisateur dans l'espace physique ou encore le contexte d'un mot dans une phrase. Nous avons également présenté l'utilisation de la programmation spatiale pour construire le contexte physique de l'utilisateur.

Dans cette partie nous étudions l'utilisation du contexte pour naviguer de proche en proche dans un système d'information, l'espace physique étant considéré comme un système d'information particulier.

9.2.1 Contexte dans un système d'information

Un système d'information est composé d'un ensemble de documents. Dans cet ensemble de documents, si nous nous référons à la définition précédente, le contexte de l'utilisateur est représenté par l'ensemble des documents qui sont proches de lui. Pour déterminer cet ensemble, l'utilisateur doit être localisé dans le système d'information et le système d'information doit être doté d'une métrique qui mesure la distance entre l'utilisateur et un document. Par exemple, lorsque l'utilisateur effectue une recherche sur le Web grâce à un moteur de recherche, l'ensemble de mots composant la requête envoyée au moteur représente la position de l'utilisateur. L'ensemble de pages retourné par le moteur représente le contexte de l'utilisateur. La distance entre l'ensemble de

mot et un document est déterminée par une méthode interne au moteur de recherche, comme la méthode TF-IDF [55]. Ici le contexte est structuré selon la distance entre les documents et l'ensemble de mots.

Dans l'espace physique nous pouvons également considérer que nous déterminons le contexte de l'utilisateur dans un système d'information. En effet, la mémoire spatiale peut être considérée comme un système d'information, pour lequel les documents sont directement stockés dans l'espace physique. De plus, l'organisation de ce système d'information est analogue à l'organisation des entités sur lesquelles les informations sont stockées. Nous parlons alors de systèmes d'information spatiaux [10].

9.2.2 Contexte et notion de dimension

Un système d'information est organisé selon une ou plusieurs dimensions. Par exemple, le Web est organisé selon la dimension topologique et la dimension textuelle. Un système d'information spatial est organisé selon la dimension physique.

Dans un système d'information, le contexte de l'utilisateur est construit selon une des dimensions qui l'organisent. Cette dimension correspond à la dimension utilisée pour déterminer la proximité d'une information relativement à la position de l'utilisateur. Par exemple, lorsque nous déterminons le contexte d'un utilisateur naviguant sur le Web à l'aide d'un moteur de recherche, nous déterminons son contexte selon la dimension textuelle.

Afin de réduire la quantité d'informations présentées à l'utilisateur, nous pouvons déterminer le contexte de l'utilisateur selon plusieurs dimensions. Par exemple, considérons un utilisateur recherchant un restaurant. Dans l'espace des restaurants, nous déterminons le contexte de l'utilisateur selon la dimension physique et la dimension "gastronomique". La dimension physique permet de déterminer les restaurants proches de l'utilisateur. La dimension gastronomique permet de déterminer les restaurants correspondant aux préférences de l'utilisateur. La combinaison de ces deux dimensions nous permet de proposer à l'utilisateur des restaurants correspondant à ses goûts et physiquement proches de lui. En construisant le contexte selon une seule dimension, nous risquons de présenter trop d'informations à l'utilisateur. Dans le chapitre 10, nous présentons une application reposant sur un contexte multidimensionnel.

9.2.3 Navigation contextuelle dans un système d'information

Dans un système d'information le contexte de l'utilisateur est déterminé selon une ou plusieurs des dimensions qui composent ce système. Dans cette partie, nous présentons comment ce contexte peut être utilisé pour simplifier la navigation dans le système d'information. Nous présentons deux modes de navigation : la navigation physique et la navigation virtuelle.

Nous considérons un système d'information composé de photos numériques et organisé selon les dimensions physique, temporelle et textuelle. Dans ce système d'informationchaque photo est étiquetée avec la position géographique à laquelle elle a été prise, la date à laquelle elle a été prise et les commentaires textuels qui lui ont été ajoutés.

Dans ce système d'information, la position géographique de l'utilisateur est déterminée avec un système de localisation, comme le GPS. Sa position temporelle correspond à la date courante. Sa position textuelle peut être représentée par un ensemble de mots clé stockés dans son profil.

9.2.3.1 Navigation physique

Nous considérons que l'utilisateur accède au système d'information depuis son téléphone mobile. Dans un tel contexte l'utilisateur est déjà engagé dans une autre activité et la navigation dans le système d'information doit être la plus implicite possible.

Le mécanisme de navigation physique permet à l'utilisateur de naviguer dans le système d'information simplement par ses déplacements physiques. Pour cela, nous surveillons la position géographique de l'utilisateur et déterminons automatiquement le contexte de l'utilisateur en fonction de cette position. Ainsi, lorsque l'utilisateur se déplace, son terminal lui présente automatiquement les photos qui ont été prises près de lui. Lorsque la position de l'utilisateur change, son contexte est mis à jour et son terminal lui présente les nouvelles photos. L'utilisateur navigue donc dans le système d'information sans interaction explicite avec un ordinateur.

9.2.3.2 Navigation virtuelle

Lorsqu'il se déplace dans l'espace physique, les photos présentes dans le contexte de l'utilisateur peuvent également être utilisées pour commencer une navigation virtuelle. La navigation virtuelle consiste à sauter d'un document à l'autre en se basant sur les documents présents dans le contexte de l'utilisateur. Lorsque l'utilisateur sélectionne un document depuis son contexte physique, nous mettons à jour son contexte en fonction de la position de ce document. Depuis ce nouveau contexte, l'utilisateur peut sauter vers un autre document. Par exemple, en sélectionnant une photo présente dans son contexte, l'utilisateur se déplace virtuellement à la position de cette photo. Nous pouvons maintenant déterminer son contexte en fonction de cette nouvelle position. A partir de sa nouvelle position virtuelle l'utilisateur peut sélectionner une photo présente dans son nouveau contexte et se déplacer vers une nouvelle position virtuelle. Le mécanisme de navigation virtuelle permet par exemple à l'utilisateur d'explorer virtuellement et rapidement les alentours afin de déterminer la prochaine destination de sa visite. La navigation virtuelle permet à l'utilisateur d'accélérer sa navigation physique.

Le contexte de l'utilisateur est structuré selon la distance et l'azimut des photos relativement à l'utilisateur et le terminal de l'utilisateur organise les photos à l'écran en fonction de cette structure. Par exemple, les photos en bas à droite de l'écran sont situées derrière sur la droite de l'utilisateur. Ainsi, lorsqu'il démarre une navigation virtuelle l'utilisateur peut donc choisir facilement la direction qu'il veut explorer. S'il sélectionne une photo située en haut du terminal, il explorera l'espace situé devant lui.

La figure 9.2 présente un exemple de navigation virtuelle. Nous avons un système d'information composé de 15 photos numériques distribuées dans l'espace physique. L'utilisateur navigue physiquement à travers ce système d'information et son contexte

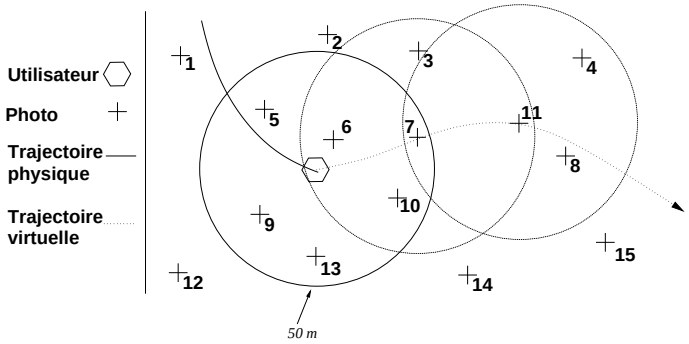

FIG. 9.2 – Exemple d'une navigation virtuelle à travers un ensemble de photos

est l'ensemble des photos situées à moins de 50 mètres de lui. Sur la figure, le contexte de l'utilisateur contient les photos 5, 6, 7, 9, 10 et 13. Depuis cette position, l'utilisateur peut se déplacer virtuellement vers l'une des photos de son contexte. Dans cet exemple, l'utilisateur choisit de se déplacer virtuellement à la photo 7. A cette nouvelle position son contexte contient les photos 3, 6, 10 et 11. Ensuite, il se déplace à la photo 11 et ainsi de suite.

9.2.4 Navigation et contexte multidimensionnel

Nous venons de voir que le contexte de l'utilisateur peut être déterminé selon plusieurs dimensions, notamment pour filtrer les informations présentées à l'utilisateur. Un contexte multidimensionnel peut également faciliter à l'utilisateur le passage d'une dimension à l'autre lorsqu'il navigue dans le système d'information.

Si nous structurons le contexte de l'utilisateur selon plusieurs dimensions, l'utilisateur est donc toujours proche des informations présentes dans son contexte selon chacune des dimensions. Lorsque l'utilisateur navigue selon une dimension, il peut à tout moment changer de dimension et naviguer selon une autre dimension, à partir de sa position courante dans le système d'information. Par exemple, considérons une application d'assistance aux réparateurs d'appareils électroménagers. Lorsque le réparateur se rend chez le client, il navigue selon la dimension physique. Une fois devant l'appareil en panne, le contexte physique du réparateur contient la référence de l'appareil. La référence de l'appareil est également localisée selon la dimension thématique. Ainsi, à partir de la référence de l'appareil, l'utilisateur continue sa navigation selon la dimension thématique afin d'accéder aux plans et notices de réparation de l'appareil. Ici les informations présentes dans le contexte de l'utilisateur sont utilisées comme point d'entrée dans le système d'information pour commencer des navigations virtuelles.

Le passage d'une dimension à l'autre nous permet de réduire les interactions explicites entre l'homme et la machine. En effet, au lieu d'effectuer une recherche explicite, l'utilisateur explore le système d'information selon la dimension voulue, à partir de sa position actuelle. Nous parlons alors de recherche par navigation. La recherche par navigation n'est utile que si l'information recherchée n'est pas trop éloignée de la posi-

tion courante. Nous présentons dans le chapitre 10 et l'annexe B deux applications qui exploitent ce mécanisme de recherche par navigation.

9.3 Définition formelle

Dans les deux parties précédentes nous avons présenté la notion de contexte en partant de la définition présente dans le dictionnaire. Cette définition nous a permis d'introduire le contexte de l'utilisateur au sein de l'espace physique. Nous avons ensuite étendu cette notion de contexte aux systèmes d'information. Dans cette partie nous présentons une définition formelle de cette notion de contexte.

9.3.1 Introduction

Nous voulons définir le contexte d'une entité, cette entité pouvant être un passage d'un texte, une personne, une page Web, un évènement historique... Le contexte de l'entité est défini dans un espace. Cet espace est composé d'éléments et le contexte de l'entité est représenté par l'ensemble des éléments qui sont proches d'elle. La nature de ces éléments dépend de l'espace considéré. Dans un texte les éléments peuvent être des mots ou des phrases. Dans l'espace physique les éléments peuvent être des personnes, des bâtiments, des objets ; dans un système d'information des documents...

La proximité est une notion qui dépend de l'espace considéré. Nous choisissons donc de définir un élément comme étant proche de l'entité considérée lorsque la distance qui les sépare est inférieure à un maximum donné. La manière dont est calculée cette distance dépend de la nature de l'espace considéré. Par exemple, dans l'espace physique nous utilisons la distance euclidienne. Selon la dimension temporelle nous calculons simplement la différence des dates.

9.3.2 Définition

Considérons un espace métrique A associé à une fonction d donnant la distance entre deux points ou éléments de A. L'espace A peut être un texte, l'espace physique, le temps ou tout autre espace dans lequel il est possible de calculer la distance entre deux éléments. Considérons E un élément particulier de A, appelé entité E, dont on cherche à déterminer le contexte $C(E)$. Dans le cas d'une application d'informatique diffuse, E correspond à l'utilisateur et A correspond à l'espace physique. Si A est le Web alors les éléments e_i sont des pages Web. E peut être une page Web particulière, une personne naviguant sur le Web, un robot Web... Pour construire le contexte de l'entité E nous devons être capable de calculer la distance entre E et les éléments e_i. Si E n'est pas directement un élément de l'espace A nous considérons qu'il existe un élément e_E, associé à l'entité E. La distance entre l'entité E et un élément e_i est alors équivalente à la distance entre e_E et e_i. Dans le cas d'une personne naviguant sur le Web, l'élément associé à l'entité correspond à la page Web qu'elle est en train de lire.

Pour calculer la distance entre deux éléments e_i et e_j de A nous avons la fonction $d(e_i, e_j)$. La définition de d n'est pas donnée ici. Elle dépend de l'espace considéré et

des dimensions selon lesquelles il est organisé.

Le contexte de E est représenté par l'ensemble des éléments de A situés à une distance de E inférieure à un maximum M, ainsi que par la structure de ces éléments :

$$C(E) = \{(e_1, s_1), (e_2, s_2), \ldots, (e_n, s_n)\} | \forall i, d(e_E, e_i) \leq M \tag{9.1}$$

Dans cette définition chaque élément e_i de $C(E)$ est associé à une information structurelle s_i. La nature de cette information dépend du type d'espace considéré, elle peut être représentée par la distance qui sépare l'élément e_i de E, par la position de e_i relativement à celle de E...

Par exemple, considérons A comme étant l'espace des restaurants. Nous structurons cet espace selon la dimension physique, donc d est la fonction de distance euclidienne. Si M est égal à 50 mètres, alors le contexte de l'utilisateur contient les restaurants situés à moins de 50 mètres de lui. Ce contexte est structuré selon la distance et l'azimut des restaurants relativement à l'utilisateur, un azimut de 0 ° correspondant à l'orientation courante de l'utilisateur. Par exemple : C(E) = {("Chez Maurice", 10m, 45 °),("L'orient", 5m, 170 °)}. Maintenant, nous considérons que A est le Web et les éléments e_i sont des pages Web. A est organisé selon la dimension topologique et la distance d mesure le nombre de liens hypertextes qui sépare deux page. Si M vaut 1, le contexte $C(E)$ d'une page Web E contient l'ensemble des pages qui ont un lien direct vers E. Ici, "être proche de" est équivalent à "avoir un lien direct vers". La seule information structurelle que contient ce contexte est la distance qui sépare les éléments de E, distance toujours égale à 1.

9.3.3 Contexte dans l'espace physique

Jusqu'à présent nous n'avons fait aucune supposition quant à la nature de l'espace A. Dans cette partie nous utilisons la définition précédente pour construire le contexte de l'utilisateur dans l'espace physique. Ceci nous permet de proposer des versions spécialisées de cette définition adaptées à l'espace physique.

9.3.3.1 Définition du contexte dans l'espace physique

Dans un espace euclidien comme l'espace physique, l'ensemble des point p_i vérifiant la condition $d(p_j, p) \leq R$ définissent une boule de centre p et de rayon R. Dans l'espace physique chaque élément e_i est associé à une position (un point) et ce point est utilisé pour calculer la distance entre cet élément et un autre. Ainsi la condition $d(e_E, e_i) \leq M$ peut être interprétée par e_i est à l'intérieur de la boule $B(e_E, M)$ de centre e_E et de rayon M. Ainsi, l'élément e_i appartient à $C(E)$ si e_i est à l'intérieur de la boule $B(e_E, M)$.

Dans l'espace physique $C(E)$ est structuré selon la distance et l'azimut des éléments e_i relativement à E. Ainsi, la définition (9.1) devient :

$$C(E) = \{(e_1, d_1, a_1), (e_2, d_2, a_2), \ldots, (e_n, d_n, a_n)\} | \forall i, e_i \in B(e_E, M) \tag{9.2}$$

Avec cette définition nous avons une boule centrée sur l'entité E. Le contexte de E est représenté par l'ensemble des éléments e_i inclus dans cette boule, avec d_i la distance séparant e_i de E et a_i l'azimut de e_i relativement à E.

9.3.3.2 Mise en œuvre grâce à la programmation spatiale

La programmation spatiale nous permet de créer facilement des applications contextuelles. La définition (9.2) peut être mise en oeuvre en définissant la forme des données comme étant des points et en utilisant une zone d'adressage sphérique.

La programmation spatiale ne limite pas les zones d'adressage aux seules sphères. Nous pouvons choisir la zone d'adressage parmi un ensemble de forme élémentaires. La définitions (9.2) devient :

$$C(E) = \{(e_1, d_1, a_1), (e_2, d_2, a_2), \ldots, (e_n, d_n, a_n)\} | \forall i, e_i \in Z(E) \tag{9.3}$$

avec $Z(E)$ correspondant à la zone d'adressage de E. Ici les données sont ponctuelles.

La programmation spatiale ne contraint pas la forme des données. Tout comme les zones d'adressage, la forme des données est choisie parmi un ensemble de formes élémentaires. Ainsi, la définition (9.3) devient :

$$C(E) = \{(e_1, d_1, a_1), (e_2, d_2, a_2), \ldots, (e_n, d_n, a_n)\} | \forall i, Z(E) \cap F(e_i) \neq \varnothing \tag{9.4}$$

$F(e_i)$ correspondant à la forme de l'élément e_i. Avec cette définition le contexte $C(E)$ est l'ensemble des éléments e_i dont la forme $F(e_i)$ est en intersection avec la zone d'adressage $Z(E)$.

Dans cette partie nous avons présenté deux définitions de $C(E)$. Dans la pratique nous nous reposerons sur la seconde définition qui est la plus générale. La première définition peut être utilisée lorsque les données sont toutes ponctuelles. Pour implémenter une application reposant sur cette définition du contexte, nous devons définir la forme $F(e_i)$ de chaque élément e_i et la zone d'adressage $Z(E)$.

9.3.4 Mise en œuvre

Dans cette partie, nous présentons comment mettre en œuvre notre définition du contexte. Dans la première partie nous présentons comment calculer la distance entre un élément et l'entité E , en fonction de la nature de l'espace considéré. Dans la seconde partie, nous présentons comment construire un contexte selon plusieurs dimensions.

9.3.4.1 Calcul de la distance

Pour mettre en œuvre cette définition de contexte, nous devons définir la fonction d permettant de calculer la distance entre l'entité E et les éléments e_i constituant l'espace dans lequel nous voulons déterminer le contexte de E. Cette fonction dépend de la ou des dimensions considérées dans la construction du contexte. Selon la nature des dimensions

considérées deux approches sont disponibles pour calculer la distance entre un élément et l'entité E.

Dans la première approche nous étiquetons les éléments avec leur position sur chacune des dimensions. E doit être également localisée selon la dimension considérée, pour calculer la distance qui la sépare d'un élément. La distance entre deux étiquettes est calculée de manière ad-hoc. Selon la dimension temporelle, cette distance correspond simplement à la différence des dates. Selon la dimension physique, cette distance est calculée avec la distance cartésienne. Ces étiquettes doivent être ajoutées à tous les éléments susceptibles de faire partie du contexte de l'utilisateur.

La seconde approche pour calculer la distance entre l'entité E et un élément se repose directement sur la structure des éléments ou la structure de l'espace contenant les éléments. Par exemple, il est possible de calculer la distance entre un groupe de mots et un document, grâce à la méthode TF-IDF. Dans ce cas là, la position de l'utilisateur est représentée par le groupe de mots et les éléments sont des documents textuels. Aussi, avec le Web nous pouvons nous reposer sur la structure de l'espace pour calculer la distance entre E et un élément. Avec le Web, cette distance peut simplement être le nombre de liens hypertextes séparant l'entité d'un élément, les éléments étant des pages et la position de l'entité étant représentée par la page courante. Dans l'espace physique, en utilisant la programmation spatiale, nous pouvons également nous reposer sur la structure de l'espace pour déterminer le contexte de E sans à avoir à étiqueter les éléments.

La seconde approche présente l'intérêt de ne pas nécessiter d'étiquettes pour calculer la distance entre un élément et l'entité E. En effet, avec la première approche les étiquettes doivent être ajoutées aux éléments soit lors de leur création, soit à posteriori si ceux-ci existent déjà lors du développement de l'application contextuelle. Dans ce dernier cas, les éléments doivent tous être étiquetés avant de pouvoir calculer le contexte de l'entité, ce qui peut être très coûteux en temps. Cependant, il existe des méthodes pour étiqueter les éléments automatiquement. Par exemple dans [23] les auteurs présentent un système qui étiquette automatiquement des photos avec les conditions météo de la photo, en se basant sur la position géographique et la date de la prise de vue.

9.3.4.2 Construction d'un contexte multidimensionnel

Jusqu'à présent nous avons présenté comment construire le contexte de l'utilisateur selon une dimension. Cependant, nous avons vu que le contexte de l'utilisateur peut être déterminé selon plusieurs dimensions. Nous présentons maintenant comment construire un contexte multidimensionnel.

Par exemple, considérons que l'utilisateur navigue dans un espace de photos organisé selon les dimensions temporelle et physique. D'après la définition (9.1), le contexte de l'utilisateur est représenté par l'ensemble des éléments situés à une distance inférieure au maximum M. Pour déterminer le contexte de l'utilisateur selon plusieurs dimensions, nous pouvons définir un maximum selon chaque dimension et déterminer le contexte de l'utilisateur selon chacune d'elles. Dans le cas présent, nous obtenons un ensemble de photos pour la dimension temporelle et un ensemble de photos pour la dimension

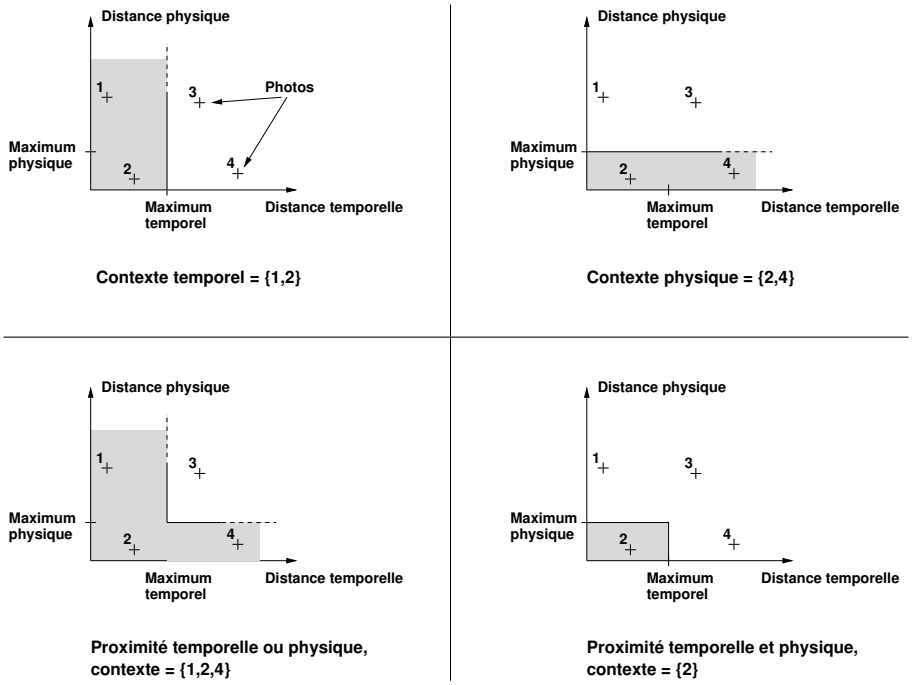

FIG. 9.3 – Construction d'un contexte multidimensionnel

physique. Si nous voulons déterminer les photos proches de l'utilisateur physiquement *et* temporellement, son contexte final correspond à l'intersection des deux ensembles. Si nous voulons déterminer les photos proches de l'utilisateur temporellement *ou* physiquement, son contexte final correspond à l'union des deux ensembles. Ces deux cas sont illustrés sur la figure 9.3.

Dans l'exemple précédent la distance maximum selon la première dimension est indépendante de la distance maximum selon la seconde dimension. Ainsi l'intersection des deux ensembles définit un rectangle. Nous pouvons imposer une dépendance entre les deux dimensions, comme illustré sur la figure 9.4 et définir une zone triangulaire. Une zone triangulaire impose qu'une image éloignée temporellement doit être proche physiquement de l'utilisateur.

9.4 Discussion

9.4.1 Travaux apparentés

Dans la partie 1.2.1 nous avons présenté les définitions du contexte de l'utilisateur présentes dans la littérature. La plupart des définitions de la littérature représentent le contexte comme un état, cet état pouvant être composé d'un ou plusieurs paramètres. Par exemple : l'utilisateur est en train de courir ; l'utilisateur se trouve devant un magasin de vêtement ; l'utilisateur et sa famille sont en train de dîner... La "valeur" du

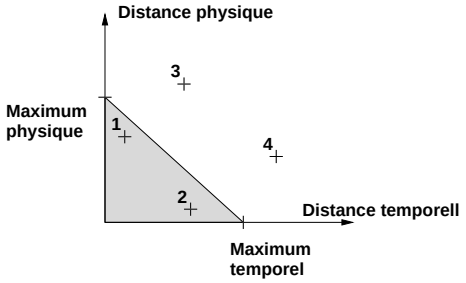

FIG. 9.4 – Construction d'un contexte multidimensionnel avec une dépendance entre les deux dimensions

contexte peut par exemple être déterminée à l'aide d'opérateurs d'agrégation comme avec le *context toolkit* [14] ou bien avec une série de relations [63].

Notre définition considère que l'utilisateur évolue dans un système d'information et représente son contexte par l'ensemble des informations proches de lui. Elle est conçue pour supporter un utilisateur qui navigue dans un système d'information, notamment en lui permettant de combiner aisément navigation virtuelle et physique. Notre définition est une généralisation à l'espace physique de la notion de contexte initialement présente dans les systèmes d'information.

9.4.2 Bilan

Dans cette partie, nous avons présenté une définition du contexte basée sur la notion de proximité. Nous nous sommes particulièrement focalisé sur le contexte d'un utilisateur qui navigue dans un système d'information. Un système d'information est structuré selon une plusieurs dimensions. Dans ce système d'information, le contexte de l'utilisateur est représenté par l'ensemble des informations proches de lui. Le contexte inclus également la structure des informations.

Dans l'espace physique, le contexte de l'utilisateur est également déterminé dans un système d'information. En effet, grâce à la programmation spatiale et au mécanisme de mémoire spatiale l'espace physique devient un système d'information spatial. Dans cet espace, le contexte de l'utilisateur est représenté par les informations qui sont proches de lui physiquement. La structure de ces informations est représentée par leur organisation géométrique dans l'espace physique. Autrement dit, les informations sont structurées selon leur position relativement à l'utilisateur.

Les informations présentes dans le contexte physique de l'utilisateur peuvent lui servir de point d'entrée pour démarrer des navigations virtuelles. Le mécanisme de navigation virtuelle permet à l'utilisateur de naviguer dans le système d'information en sautant d'information en information. Pour les systèmes d'information organisés selon plusieurs dimensions, l'utilisateur peut donc découvrir une information selon une dimension et poursuivre sa navigation virtuellement selon une autre dimension. Par exemple, un réparateur électroménager accède selon la dimension physique à la fiche

d'entretien de l'appareil qu'il est en train de réparer. Depuis cette fiche, il peut naviguer virtuellement dans le système d'information selon la dimension thématique, afin de découvrir le plan de l'appareil qu'il répare.

Chapitre 10

Navigateur de photos

Dans le chapitre précédent nous avons présenté la notion de contexte et donné une nouvelle définition pour le contexte de l'utilisateur basée sur la notion de proximité. Dans ce chapitre, nous présentons une nouvelle application contextuelle qui repose sur cette définition du contexte : un navigateur de photos permettant à l'utilisateur de naviguer à travers une collection de photos à la fois physiquement et virtuellement [42]. Dans cette application, la géométrie est utilisée à la fois pour structurer la collection de photos, construire le contexte de l'utilisateur et l'aider à naviguer à travers la collection de photos.

Dans la partie suivante nous présentons le navigateur de photos et notamment un scénario d'utilisation et les fonctionnalités proposées. La seconde partie présente l'implémentation du navigateur. Nous présentons les résultats de nos premières expérimentations sur le terrain dans la troisième partie.

10.1 Présentation

Dans cette partie, nous commençons par présenter la collection de photos dans laquelle évolue l'utilisateur. Nous présentons ensuite dans la seconde partie un scénario type d'utilisation de l'application. La troisième partie présente les fonctionnalités offertes par le navigateur de photos. Dans la quatrième partie nous présentons les modes d'interaction entre le navigateur et l'utilisateur.

10.1.1 Collection de photos

Avec le développement récent des appareils photos numériques et des téléphones mobiles équipés d'appareils photo, les utilisateurs ont pris l'habitude de prendre des photos quels que soient le lieu et le moment. Ainsi, une grande quantité de photos est créée dans l'environnement physique. La plupart du temps ces photos restent isolées sur le disque dur de l'utilisateur. Certaines personnes partagent leurs photos via des galeries en ligne, mais ce n'est qu'un partage pour la consultation. Cependant, nous remarquons que les photos prises par une personne peuvent intéresser une autre personne. Par exemple, considérons un spectateur d'un match de football. Si ce spectateur est mal situé

dans le stade, il peut être intéressé par les photos des buts faites depuis la tribune située en face de la sienne. De même, lors d'un attentat, les photos faites par les personnes proches des lieux de l'attentat peuvent être particulièrement utiles aux policiers pour reconstituer les faits.

Dans ce chapitre, nous considérons un système d'information permettant aux utilisateurs de partager leurs photos de manière globale. De plus, ces photos étant numériques, elles sont partageables sur le champs et directement consultables par d'autres utilisateurs. Nous considérons donc que les utilisateurs créent, partagent et accèdent aux photos depuis l'environnement physique. Dans ce chapitre nous considérons que ce système d'information est disponible et nous intéressons à la navigation à travers ce système d'information.

Le système d'information est organisé selon trois dimensions : la dimension physique, la dimension temporelle et la dimension textuelle. Dans ce système d'information, nous pouvons donc déterminer le contexte d'un utilisateur selon une ou plusieurs de ces dimensions. Par exemple, selon la dimension temporelle le contexte de l'utilisateur contient les photos qui ont été prises récemment. Les photos sont organisées selon la dimension temporelle en les étiquetant avec leur date de création. Selon la dimension physique, elles sont étiquetées avec la position géographique à laquelle elles ont été créées. Selon la dimension textuelle les photos sont enrichies avec des commentaires textuels.

10.1.2 Scénario

John commence par naviguer physique dans le système d'information, son contexte contient les photos qui sont physiquement proches de lui. Ces photos sont affichées sur l'écran du navigateur. A mesure que John se déplace, son contexte est automatiquement mis à jour et les changements sont reflétés sur l'écran. John arrive ensuite devant un château. Pour décider s'il le visitera ou non, il démarre une navigation virtuelle en sélectionnant une photo depuis son contexte. La position de cette photo devient sa position virtuelle ; il a virtuellement sauté à une autre position. Le contexte de John est alors représenté par les photos proches de sa position virtuelle. En effectuant plusieurs sauts, il peut ainsi virtuellement entrer dans le château et découvrir son intérieur. Cette possibilité est particulièrement intéressante pour entrer dans les monuments qui ne sont pas ouverts aux visites, ou pour décider d'une visite.

Finalement, notre utilisateur décide de ne pas visiter le château. Cependant il aimerait le voir en hiver. Il change donc sa position temporelle et la règle sur hiver pour obtenir des photos du château sous la neige. John règle ensuite sa position temporelle sur 1945 et poursuit la visite de la ville. Maintenant, son contexte contient des photos qui ont été prises près de sa position physique à la fin de la seconde guerre mondiale, lui permettant ainsi de visiter la ville dans le passé.

Arrivé à un carrefour, John ne sait pas quelle direction prendre. Il poursuit donc sa navigation physique par une navigation virtuelle pour explorer les alentours. Il sélectionne une photo de son contexte et fait quelques sauts dans une première direction. Ne trouvant rien dans cette direction, il essaye une autre direction et découvre une église qu'il décide de visiter. Il choisit donc de prendre cette direction et revient à la

navigation physique. Dans ce cas, la navigation virtuelle est utilisée pour "accélérer" la navigation physique. En faisant quelques sauts dans une direction donnée, l'utilisateur peut déterminer si cette direction l'intéresse suffisamment pour la prendre.

10.1.3 Fonctionnalités offertes par le navigateur

Dans la partie précédente nous avons présenté un scénario d'usage typique du navigateur de photo. Dans cette partie, nous détaillons les différentes fonctionnalités proposées par le navigateur.

10.1.3.1 Sélection physique de photos

Pendant qu'il navigue physiquement à travers la collection de photos, l'utilisateur a la possibilité de sauver les photos qui lui plaisent le plus. En utilisant la navigation multidimensionnelle, il peut sauver des photos insolites ou rares. Par exemple, lorsqu'il est au bord de la mer en été, l'utilisateur peut obtenir des photos de la plage en hiver en utilisant une navigation temporelle. Lorsque l'utilisateur marche dans un quartier en construction, il peut obtenir un aperçu du quartier une fois les travaux terminés grâce aux schémas d'architectes ; il peut également obtenir des photos du quartier avant les constructions.

Nous appelons ce schéma d'interaction sélection physique, parce que les mouvements physiques de l'utilisateur lui permettent de découvrir des photos. Tout comme une personne naviguant sur le Web doit sauter de page en page pour découvrir de nouvelles photos ou documents, avec la navigation physique l'utilisateur doit se déplacer pour découvrir de nouvelles photos.

10.1.3.2 Planification de trajet

Pendant que l'utilisateur navigue à travers la collection de photo, il peut utiliser la navigation virtuelle pour accélérer sa navigation physique et planifier son trajet. Lorsque l'utilisateur sélectionne une photo dans son contexte, il saute virtuellement à la position de cette photo et son contexte est mis à jour. En sautant de photo en photo l'utilisateur explore virtuellement son voisinage et peut découvrir un endroit intéressant. Il peut alors verrouiller cet endroit en tant que destination suivante. Dans ce cas, pendant que l'utilisateur navigue physiquement, le navigateur lui présente les photos physiquement proches de lui et affiche un compas qui indique la direction à prendre pour atteindre la destination.

10.1.3.3 Contournement d'obstacles physiques

Le mécanisme de navigation virtuelle peut également être utilisé pour aller là où il est physiquement impossible d'aller. Par exemple, lorsque l'utilisateur est en face d'un bâtiment fermé ou d'un bâtiment qui n'est pas ouvert aux visites, il peut effectuer quelques sauts virtuels pour entrer virtuellement dans le bâtiment et obtenir une vue

de l'intérieur. Ceci est également intéressant pour préparer les visites et décider si l'utilisateur et sa famille visiteront ou non le monument. Tout comme la planification de trajet, ce mécanisme permet à l'utilisateur d'accélérer sa navigation physique et l'aide à décider de ses prochaines actions.

10.1.3.4 Web cam virtuelle

La navigation temporelle propose une autre possibilité intéressante. Si nous considérons que nous avons accès à une vaste quantité de photos, alors grâce à la navigation temporelle il est possible d'accéder à différentes vues du lieu courant. Autrement dit, l'utilisateur accède à différentes "versions" du lieu selon la dimension temporelle.

Si la densité physique et temporelle des photos est suffisamment haute, alors l'utilisateur peut même surveiller l'activité du lieu. Il peut également consulter l'activité du lieu à posteriori. Nous appelons un tel mécanisme Web cam virtuelle. Il n'y a pas de Web cam, ce sont les photos successives prises par les différents utilisateurs qui permettent à l'utilisateur d'avoir une vue du lieu similaire à la vue fournie par une vraie Web cam. En fait, en utilisant la navigation virtuelle il est même possible à l'utilisateur de surveiller l'activité d'un lieu distant. Un tel scénario est réaliste lorsque l'on considère la quantité de photos numériques qui sont crées dans l'espace physique soit avec les téléphones équipés d'appareil photo soit avec les appareils photo numériques.

10.1.4 Interactions

Du point de vue des interactions, l'objectif du navigateur est de réduire les interactions explicites entre l'utilisateur et la machine.

La navigation physique réduit les interactions explicites avec la machine car elle est directement contrôlée par les mouvements physiques de l'utilisateur. Autrement dit, l'utilisateur interagit avec l'application simplement en se déplaçant dans l'espace physique. Ce type d'interactions, appelé interactions implicites [57], est particulièrement adapté aux applications exécutées dans l'espace physique. Elles permettent à l'utilisateur de se concentrer complètement sur son activité courante.

La combinaison des navigations physique et virtuelle nous permet également de réduire les interactions explicites entre l'homme et la machine. Pendant que l'utilisateur navigue physiquement à travers le système d'information son contexte contient en permanence des photos physiquement proches. A tout moment, ces photos peuvent servir de point de départ pour une navigation virtuelle. Ainsi, lorsque l'utilisateur veut explorer son voisinage physique, il n'a pas à spécifier à l'application la zone qu'il souhaite explorer, son contexte lui fournit automatiquement les points de départs.

10.2 Implémentation

Dans la partie précédente nous avons introduit notre navigateur de photos en présentant un scénario typique, les différents services proposés et le mode d'interaction avec l'application. Dans cette partie nous détaillons l'implémentation du navigateur, nous

décrivons brièvement son architecture logicielle, puis son interface. Finalement, nous présentons comment le contexte de l'utilisateur est construit à partir de la collection de photo.

10.2.1 Architecture Logicielle

Pour cette mise en œuvre, notre objectif à été de minimiser la quantité de logiciels à installer sur le terminal, afin de rendre l'application compatible avec un maximum de terminaux. Ainsi, nous nous reposons principalement sur les technologies Java et Web, car elles permettent le chargement dynamique d'applications sur les terminaux mobiles. L'application exploite un modèle client / serveur. Le terminal de l'utilisateur (client) est un Pocket PC équipé d'un récepteur GPS. L'application est exécutée directement dans le navigateur Web du terminal. Elle consiste en une page Web affichant le contexte photographique de l'utilisateur. Cette page contient également une applet Java qui surveille la position et l'orientation de l'utilisateur à l'aide du récepteur GPS. L'applet Java envoie régulièrement la position de l'utilisateur au serveur, qui évalue si la page Web doit être mise à jour. Autrement dit, le serveur évalue si le contexte de l'utilisateur a changé. Le terminal de l'utilisateur communique avec le serveur via une interface de communication 802.11b (WiFi).

Du côté du serveur, l'application est exécutée par un servlet Java, qui génère dynamiquement les pages Web en fonction de la position et de l'orientation de l'utilisateur. Nous utilisons le serveur Web Jetty[1], qui est un serveur Web et un conteneur de servlet entièrement écrit en Java. Nous avons choisi ce serveur Web car il est bien plus simple à configurer que le couple Apache / Tomcat. Pour les essais, nous avons créé une collection de photos du campus de l'Université de Rennes 1. Les photos sont stockées dans une base de données et étiquetées à l'aide des champs EXIF [2]. Les champs EXIF sont un ensemble de méta-données ajoutées aux photos créées avec les appareils photo numériques. Ils permettent de stocker des informations telles que le temps de pose, l'ouverture de la focale, les coordonnées géographiques, le modèle de l'appareil... Avec cette approche, du point de vue logiciel, l'utilisateur n'a besoin que d'un simple navigateur Web pour exécuter l'application. L'architecture logicielle du navigateur est illustrée sur la figure 10.1.

Si nous envisageons une utilisation plus globale de notre application, elle devrait être exécutée sur des Smart Phones avec une connection globale vers le serveur, comme une connection GPRS ou UMTS. Le terminal devrait également être capable de surveiller sa position. Les services de localisation commencent à apparaître sur les téléphones mobiles, ainsi la limitation actuelle pour le déploiement d'une telle application est plus financière que technique. En effet les communications GPRS ou UMTS restent encore très chères.

[1] jetty.mortbay.org/jetty
[2] www.exif.org

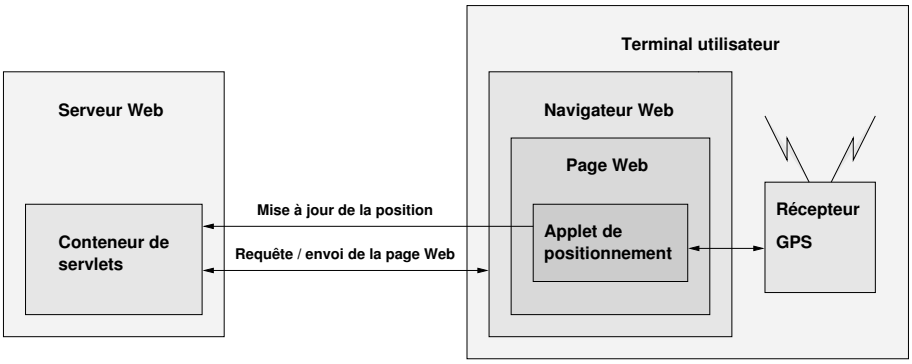

FIG. 10.1 – Architecture logicielle du navigateur de photos

10.2.2 Interface

L'interface du navigateur de photos est présentée sur la figure 10.2. Elle est principalement composée d'un tableau contenant les photos proches. Afin que l'utilisateur puisse localiser rapidement dans l'espace physique le contenu des photos présentées par le navigateur, ce tableau est une représentation directe du contexte de l'utilisateur ; son organisation géométrique est analogue à la structure géométrique du contexte. Par exemple, la photo située en haut de l'écran représente des objets ou des bâtiments situés devant l'utilisateur ; la photo située en bas à droite de l'écran représente ce qui est derrière l'utilisateur sur sa droite. Nous avons choisi de ne présenter que neuf photos car l'écran du navigateur a une taille réduite. Ce tableau de photos est mis à jour dynamiquement au fur et à mesure des déplacements de l'utilisateur, selon sa position et son orientation.

Pour filtrer les photos présentées, l'utilisateur peut entrer un mot clé en haut de l'interface. Dans ce cas là, son contexte est calculé selon la dimension physique et la dimension textuelle. Dans la partie suivante nous présentons une manière simple de calculer ce contexte. Dans l'exemple montré sur la figure 10.2 l'utilisateur a entré le mot clé neige, ce qui implique qu'il désire obtenir des photos de son voisinage sous la neige.

Lorsque l'utilisateur navigue physiquement à travers la collection de photos, la case centrale du tableau contient une rose des vents représentant la position de l'utilisateur dans le tableau. Lorsque l'utilisateur sélectionne une photo dans le tableau, il démarre une navigation virtuelle et la position de la photo sélectionnée devient sa position virtuelle. Dans ce cas, la case centrale du tableau contient la photo associée à la position virtuelle courante. Si l'utilisateur sélectionne une autre photo, il saute à une nouvelle position virtuelle et la tableau est mis à jour. La représentation du contexte sous forme d'un tableau permet à l'utilisateur de savoir quelle direction il est en train de suivre, lors de sa navigation virtuelle. Pour naviguer virtuellement vers la gauche, l'utilisateur doit juste cliquer sur la photo située dans la case de gauche. Ici nous pouvons observer que le contenu du contexte, mais également sa structure géométrique présentent un intérêt

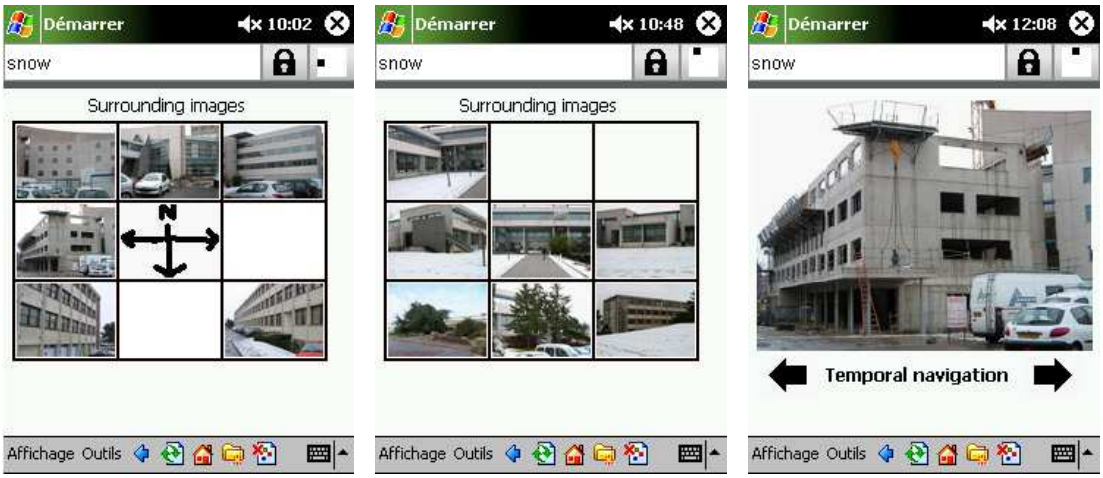

FIG. 10.2 – Image a : capture d'écran de la navigation physique. La position de l'utilisateur est représentée par une croix. Image b : capture d'écran de la navigation virtuelle. Image c : capture d'écran de la navigation temporelle.

pour l'utilisateur.

Lorsque l'utilisateur navigue virtuellement à travers la collection de photos et trouve une destination intéressante, il peut verrouiller cette position virtuelle en tant que prochaine destination. Dans ce cas, le compas situé en haut à droite de l'interface indique en permanence la direction à suivre pour atteindre cette destination. Ce compas est simplement représenté par un carré noir se déplaçant autour d'un carré blanc, en fonction de la direction relative de la destination. De cette manière, l'utilisateur peut préparer son trajet avec la navigation virtuelle, revenir ensuite à la navigation physique et atteindre sa destination. Ce mode de navigation présente l'avantage d'être plus intuitif qu'une navigation avec carte. En effet, grâce au tableau de photos et au compas l'utilisateur visualise directement sur son écran la direction à suivre pour rejoindre sa destination. Avec une carte, l'utilisateur est obligé de faire en permanence la correspondance entre la carte et son environnement physique.

Pendant une navigation virtuelle, si l'utilisateur sélectionne la photo centrale, il obtient alors une version en plus haute résolution de la photo et peut commencer une navigation temporelle depuis cette photo. Dans ce cas la position physique de la photo centrale représente encore la position virtuelle de l'utilisateur. La date de création de la photo représente la position temporelle de l'utilisateur. La navigation temporelle permet à l'utilisateur de se déplacer dans le passé ou le futur, en partant de la photo courante. Avec les deux flèches située en bas de l'interface, l'utilisateur peut sauter à la photo la plus proche dans chacune des directions. Par exemple avec la flèche de droite, il saute à la photo qui a été prise juste après la photo courante, à la même position physique.

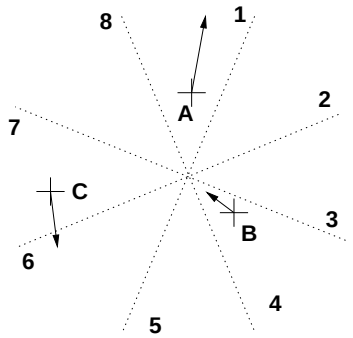

FIG. 10.3 – Filtrage physique. L'espace autour de l'utilisateur est divisé en huit secteurs et trois photos sont proches de lui : A, B, C. Les photos sont associées à un secteur en fonction de leur orientation. La photo A est associée au secteur 1, la photo B au secteur 8 et la photo C au secteur 5.

10.2.3 Construction du contexte

Nous considérons que l'utilisateur navigue à travers une collection de photos où la densité physique des photos est importante. Ainsi, le contexte de l'utilisateur peut contenir de nombreuses photos, Cependant en raison de la taille limitée de l'écran, le navigateur ne peut présenter qu'un maximum de huit photos, plus une neuvième lors des navigations virtuelles. Ainsi, nous devons filtrer les photos pour trouver celles à afficher.

Nous déterminons les photos proches en définissant une zone autour de l'utilisateur et en récupérant les photos contenues dans cette zone. Ensuite, nous devons filtrer cet ensemble de photos pour trouver la photo à mettre dans chaque case du tableau.

Chaque case du tableau est associée à un secteur angulaire défini en fonction de l'orientation de l'utilisateur. Par exemple la cellule du haut du tableau est associée à un secteur qui s'étend devant l'utilisateur. L'espace physique est ainsi divisé en huit secteurs égaux. Les photos sont ensuite placées dans les secteurs en fonction de l'orientation de la prise de vue. L'orientation de la prise de vue correspond à l'orientation de l'utilisateur lorsqu'il a pris la photo. De cette manière lorsque l'utilisateur regarde sur sa droite il voit ce que la photo située dans la case de droite montre. Ce mécanisme est illustré sur la figure 10.3. Après avoir associé chaque photo à un secteur, nous recherchons pour chaque secteur la photo la plus proche de l'utilisateur et la stockons dans la case correspondante.

Nous pouvons remarquer que la position des photos n'est pas utilisée pour trouver le secteur auquel elles appartiennent. Par exemple, considérons la photo B sur la figure 10.3. Cette photo est physiquement localisée dans le secteur 4, mais d'après son orientation le sujet de la photo est probablement situé dans le secteur 8. C'est pourquoi nous utilisons l'orientation des photos pour les associer à un secteur donné. Ce problème "d'inversion de case" vient du fait que le sujet d'une photo ne se trouve pas à la position de la photo.

Nous pouvons également utiliser la dimension textuelle pour déterminer le contexte de l'utilisateur. Dans ce cas l'utilisateur entre un mot clé k dans l'interface. Une photo est considérée comme proche si son commentaire textuel contient le mot clé et si elle est proche de l'utilisateur.

Pour calculer la distance entre l'utilisateur et une photo nous utilisons la formule suivante : $d = d_{phy} + d_{text}$ avec d_{phy} qui représente la distance physique et d_{text} la distance textuelle. Ces deux distances sont normalisées. Une photo est incluse dans le contexte de l'utilisateur si la distance qui les sépare est inférieure à 20 mètres. Ainsi, lorsqu'un le mot clé k n'est pas présent dans les commentaires d'une photo, nous avons $d_{text} = 20$; si le mot clé est présent nous avons $d_{text} = 0$. De cette manière, pour une photo située à 12 mètres de l'utilisateur et dont les commentaires contiennent k, nous avons $d = 12$. Cependant, pour une photo localisée à un mètre de l'utilisateur et dont les commentaires ne contiennent pas k, nous avons $d = 1 + 20 = 21$. Cette seconde photo, bien que plus proche physiquement de l'utilisateur que la première, sera donc considérée comme plus éloignée. La balance entre distance physique et distance textuelle peut être ensuite ajustée en adaptant la valeur maximale de la distance textuelle.

10.3 Expérimentations

Pour évaluer la conception du navigateur et décider des changements et améliorations à apporter nous avons effectué une évaluation préliminaire de l'application.

L'évaluation du tableau à montré que le sujet des photos présentées doit toujours être visible pour l'utilisateur. La première version du tableau essayait de remplir toute les cases. Cette approche impliquait que certaines photos étaient présentées à l'utilisateur alors que leur sujet était trop loin pour être visible. L'évaluation nous à montré qu'il vaut mieux laisser une case vide, que de présenter une photo dont le sujet n'est pas visible. L'utilisateur s'attend à voir autour de lui ce qui est présenté sur l'écran. Ainsi, afficher une photo avec un sujet invisible n'avait pas de sens pour l'utilisateur et rendait l'application difficile à comprendre.

Les évaluations sur le terrain ont également montré que l'écran ne doit pas être mis à jour trop souvent. Lorsque la collection de photo est dense, le contexte de l'utilisateur change en permanence. Pour être cohérent avec la position de l'utilisateur, l'écran du navigateur devrait être mis à jour chaque fois que le contexte change. Cependant, les évaluations ont montré que lorsque l'écran est mis à jour trop souvent, l'utilisateur passe la plupart de son temps à regarder le navigateur. Le navigateur devient alors trop intrusif et dérangeant. Nous avons évalué qu'un temps de rafraîchissement de 4 à 5 seconde est un bon compromis. L'utilisateur peut ainsi se focaliser sur sa visite et consulter juste de temps en temps le navigateur.

Actuellement l'orientation de l'utilisateur est surveillée en interpolant la position de l'utilisateur. Ainsi, les changements d'orientation ne sont détectés que lorsque l'utilisateur se déplace. En raison de la précision du GPS, les changements d'orientation ne sont détectés que tous les deux ou trois mètres. Cette approche n'est pas adaptée pour une application telle que le navigateur. En effet, lorsque l'utilisateur tourne sur lui-même,

l'affichage n'est pas mis à jour car l'utilisateur ne se déplace pas assez pour que les changements d'orientation soient détectés. Nous envisageons de recourir à un compas électronique pour résoudre ce problème.

Finalement, les évaluations sur le terrain ont montré que le contenu des photos est également important pour la navigation physique. Lorsque l'utilisateur navigue physiquement, il est concentré sur sa visite. Ainsi, lorsqu'il détourne son attention pour regarder le navigateur il doit pouvoir identifier rapidement le contenu des photos. Les photos qui contiennent un point de repère clair (*landmark*) sont les plus faciles à identifier. Les points de repère sont également utiles pour la navigation virtuelle. Lorsque l'utilisateur recherche une destination, il verrouille habituellement sa destination sur un point de repère clairement identifiable. L'importance des points de repère pour naviguer dans l'espace physique est étudiée dans [20].

10.4 Bilan

Dans ce chapitre nous avons présenté un navigateur permettant à l'utilisateur de naviguer à travers une collection de photos. Ce navigateur représente le contexte de l'utilisateur par l'ensemble des photos qui sont proches de l'utilisateur. Le navigateur repose sur deux modes de navigation : la navigation physique et la navigation virtuelle. L'utilisateur navigue physiquement lorsqu'il se déplace physiquement. L'utilisateur navigue virtuellement, lorsqu'il se déplace à travers le système d'information en sautant de photo en photo. Le navigateur permet également à l'utilisateur de combiner les deux modes de navigation. Grâce à la navigation physique l'utilisateur peut atteindre un point précis du système d'information correspondant à sa position géographique courante. Ensuite, l'utilisateur peut explorer les alentours grâce à la navigation virtuelle.

Le navigateur propose plusieurs fonctionnalités à l'utilisateur. La navigation physique permet par exemple à l'utilisateur de visiter une ville à une autre époque ou une autre saison. A partir de la position courante de l'utilisateur, la navigation virtuelle lui permet par exemple de planifier son trajet ou de contourner des obstacles physiques. Le navigateur permet également à l'utilisateur de naviguer virtuellement selon la dimension temporelle. Ce mécanisme de navigation temporelle lui permet d'obtenir différentes versions du lieu courant.

Nous avons implémenté ce navigateur et effectué une première série d'expérimentations. Les principaux résultats nous ont montré que le contenu des photos est particulièrement important. Tout d'abord le sujet des photos doit être visible pour l'utilisateur. De plus, il est plus facile d'effectuer la correspondance entre l'écran du navigateur et l'espace physique lorsque les photos contiennent un point de repère clairement identifiable. Les expérimentations nous ont également montré que le contenu de l'écran ne doit pas être mis à jour trop souvent, pour ne pas détourner l'attention de l'utilisateur de son activité courante.

Conclusion et perspectives

Bilan

Dans cette thèse nous avons étudié le rôle de la géométrie dans la construction d'applications d'informatique diffuse. Dans un premier temps nous avons étudié le rôle de la géométrie dans la programmation d'applications. Puis dans un second temps nous avons étudié le rôle de la géométrie dans la construction et la navigation à travers un système d'information.

Programmation des applications

Dans le chapitre 5 nous avons présenté la programmation spatiale qui permet au programmeur de synchroniser les applications sur la détection d'interactions physiques, directement au niveau des opérations de lecture de données. La programmation spatiale repose sur deux mécanismes : i) le mécanisme de mémoire spatiale ; ii) le mécanisme de synchronisation physique. Le mécanisme de mémoire spatiale permet aux entités physiques de publier et d'accéder aux données portées par les autres entités. Le mécanisme de synchronisation physique permet de synchroniser le programme exécuté par une entité sur la découverte d'une donnée particulière. D'un point de vue physique, découvrir une donnée revient à détecter la rencontre entre l'entité qui cherche la donnée et l'entité qui porte la donnée. Dans sa version initiale, la programmation spatiale souffre de deux grandes limitations. D'une part, elle ne nous permet pas de prendre en compte précisément la structure géométrique de l'espace physique. D'autre part, elle ne nous permet pas de synchroniser les applications sur des interactions physiques autres que des rencontres entre entités physiques.

Dans le chapitre 6 nous avons présenté une extension géométrique pour la programmation spatiale. Cette extension nous permet de définir précisément les configurations géométriques des interactions physiques à détecter. Cette extension nous permet d'une part d'associer une forme géométrique à chaque donnée et d'autre part d'associer une zone d'adressage aux opérations de lecture de données. Grâce au mécanisme de synchronisation physique le programmeur décrit explicitement les rencontres entre entités dans le code de son application. Désormais, il décrit non seulement les rencontres entre entités, mais également leur configuration géométrique.

Dans sa version initiale, la programmation spatiale ne permet pas de synchroniser les programmes exécutés par les entités sur des interactions physiques du type “départ

d'une entité". Dans le chapitre 7 nous avons présenté deux nouvelles opérations : i) l'opération `readOnce` qui détecte l'arrivée d'une nouvelle donnée dans une zone de l'espace physique ; ii) l'opération `lostOne`ăqui détecte le départ d'une donnée précédemment lue avec `readOnce`. Ces deux opérations sont particulièrement utiles pour synchroniser les applications sur les arrivées et départs d'entités du même type, comme l'application de calcul du prix d'un chariot.

Dans le chapitre 8 nous avons présenté l'opération `take` qui nous permet de transférer de manière atomique une donnée d'une entité à une autre. Dans le contexte de la programmation spatiale, nous avons vu que l'opération `take` ne peut pas être atomique. Le protocole de cette opération repose sur une contrainte géométrique pour minimiser le nombre d'opérations non-atomiques. Cette contrainte géométrique définit une distance minimum entre les deux entités qui participent à l'opération, au-delà de laquelle l'opération ne peut pas démarrer. Grâce à une série d'évaluations nous avons pu observer que cette contrainte géométrique permet de grandement diminuer le nombre d'opérations non-atomiques. Sans contrainte nous avons entre 12 et 14 pour cent d'opérations non-atomiques et aucune opération non-atomique avec une contrainte de 90 mètres.

Accès et navigation dans un système d'information

Dans le chapitre 9 nous avons étudié la notion de contexte dans un système d'information. Nous avons proposé une définition du contexte basée sur la notion de proximité. Dans un système d'information le contexte de l'utilisateur est représenté par l'ensemble des informations qui sont proches de l'utilisateur. La géométrie joue un rôle directe dans la construction du contexte de l'utilisateur. En effet, les systèmes d'information sont structurés selon une ou plusieurs dimensions et les informations proches de l'utilisateur sont déterminées selon ces dimensions. Un système d'information peut être organisé selon la dimension physique, thématique, temporelle...

Le contexte de l'utilisateur est également structuré selon des propriétés géométriques. Ces propriétés géométriques caractérisent la position des informations qui composent le contexte, relativement à la position de l'utilisateur. Par exemple, dans le chapitre 10 nous avons présenté un navigateur de photos qui permet à l'utilisateur de naviguer dans un système d'information composé de photos numériques. Ce système d'information est organisé selon les dimensions physique, temporelle et textuelle. Selon la dimension physique le contexte de l'utilisateur est représenté par l'ensemble des photos physiquement proches de lui. La structure du contexte physique de l'utilisateur est représentée par la position géographique des photos relativement à l'utilisateur. Cette application affiche le contexte de l'utilisateur sur l'écran de son terminal. L'organisation des photos sur l'écran est analogue à la structure géométrique du contexte : les photos affichées en haut de l'écran sont situées devant l'utilisateur, les photos affichées en bas à gauche sont situées derrière et à gauche de l'utilisateur.

Le contexte de l'utilisateur peut être utilisé pour naviguer virtuellement dans le système d'information en sautant d'information en information. Lorsque l'utilisateur sélectionne une information présente dans son contexte, il saute à la position de cette information et son contexte est mis à jour en fonction de sa nouvelle position. Le naviga-

teur de photos repose fortement sur ce mécanisme de navigation virtuelle. Il permet par exemple à l'utilisateur d'explorer son environnement proche pour chercher sa prochaine destination. Il lui permet également de contourner les obstacles physiques : lorsque l'utilisateur est devant un bâtiment fermé, il peut effectuer quelques sauts virtuels en direction du bâtiment pour entrer virtuellement à l'intérieur de celui-ci. Ici, la structure géométrique du contexte participe directement à la navigation. En effet, lorsque l'utilisateur sélectionne par exemple une photos en haut de l'écran de son terminal, il sait qu'il se déplace virtuellement en avant. Ainsi, en organisant l'affichage de manière analogue à la structure géométrique du contexte, nous permettons à l'utilisateur de choisir la direction qu'il suit lors de ses navigations virtuelles.

Perspectives

Le modèle de programmation est actuellement exécuté sur des calculateurs de type PDA ou téléphones mobiles. Or la programmation spatiale suppose que l'ensemble des entités physiques qui participent à l'application embarquent un calculateur sans fil. Nous ne pouvons embarquer que des calculateurs de petite taille et aux capacités réduites sur les entités. Il serait donc nécessaire de porter SPREAD sur ce type d'architecture, tout en tenant compte des contraintes propres à ces architectures, notamment du point de vue de la consommation énergétique. Actuellement, les opérations de lecture en mémoire reposent sur un mécanisme d'attente active qui diffuse régulièrement des requêtes afin de découvrir une donnée. Cette approche est coûteuse en énergie. En effet, la découverte d'une nouvelle donnée est conditionnée par les mouvements des entités : lorsqu'une opération est bloquée en attente d'une donnée, au moins une entité doit se déplacer pour que l'opération découvre une nouvelle donnée. Ainsi, après une première requête de donnée, il est inutile de diffuser des requêtes de données tant qu'aucune entité ne se déplace. Il serait donc intéressant d'étudier un nouveau protocole contrôlé par les mouvements des entités, de sorte que lorsqu'aucune entité ne se déplace aucune requête de données ne soit diffusée.

Actuellement, nous ne pouvons pas programmer de synchronisation physique entre plus de deux entités. Par exemple dans le cas d'Ubibus, pour l'arrêt du bus nous ne pouvons pas définir une synchronisation impliquant le bus, le piéton et l'abri bus. Cette synchronisation détecterait la proximité *s*imultanée des trois participants. Pour l'instant cette synchronisation doit être décomposée en deux synchronisations : la première synchronisation implique l'abri bus et le piéton et détecte l'arrivée du piéton à l'abri bus ; la seconde synchronisation implique l'abri bus et le bus et valide l'arrivée du bus à l'abri bus. Afin de garantir que le piétons est toujours à l'abri bus lorsque le bus s'arrête, ces deux synchronisations devraient se dérouler en une seule et même étape atomique. Ce type d'interaction physique nécessite l'étude de nouvelles opérations, permettant de synchroniser les programmes de trois entités ou plus.

Pour le navigateur de photos, il nous reste à effectuer des essais utilisateurs afin d'évaluer les différentes fonctionnalités. Après analyse, les résultats de ces essais seraient à prendre en compte dans le cadre d'un nouveau cycle de développement.

Le système d'information sur lequel repose le navigateur est actuellement une simple base de données. Il nous faut donc concevoir un système d'information permettant aux utilisateurs de partager et d'accéder aux photos depuis l'environnement physique. La construction de ce système d'information ouvre plusieurs perspectives de recherche. Du point de vue du stockage des photos, nous pourrions stocker les photos directement dans l'espace physique là où elles sont créées. En effet, dans le cas de la navigation physique les utilisateurs accèdent aux photos à proximité de l'endroit où elles ont été prises. Inversement, dans le cas de la navigation virtuelle, les utilisateurs n'accèdent pas aux photos là où elles ont été créées. Nous devrions donc étudier une solution de stockage permettant à la fois un accès à distance et un accès local rapides. Afin de satisfaire ces contraintes antagonistes, nous pourrions par exemple étudier les architectures à base d'infostations [32] et de caches locaux. Un cache local est associé à une zone de l'espace physique représentée par la zone de communication de l'infostation et sert les utilisateurs situés dans cette zone. Au sein de telles architectures, nous devrions également étudier les politiques de pré-chargement et de conservation des données en cache, en fonction de la trajectoire des utilisateurs.

Annexe A

Évaluation des opérations readOnce et lostOne

Dans le chapitre 7 nous avons présenté les opérations `readOnce` et `lostOne`. La réalisation de ces opérations dépend à la fois de SPREAD et des technologies de la plateforme d'exécution, c'est pourquoi nous n'avons présenté que les résultats principaux des évaluations dans le chapitre 7. Dans cette annexe, nous présentons l'évaluation des opérations `readOnce` et `lostOne`.

Nous nous reposons toujours sur l'application de calcul du prix d'un chariot. Nous effectuons ces évaluations sur des PDA de type Pocket PC. Dans cette évaluation nous utilisons deux Pocket PC : *Cad* et *Prod*. Le pocket PC *Cad* correspond au chariot et *Prod* correspond aux produits. Idéalement, nous devrions utiliser un Pocket PC par produit, cependant de telles conditions expérimentales sont difficiles à reproduire. Il est en effet très long de mettre en œuvre des évaluations impliquant plusieurs Pocket PC, or cette évaluation nécessiterait plusieurs dizaines de Pocket PC.

Dans cette annexe, nous évaluons dans un premier temps la réactivité de l'opération `readOnce`. Dans un second temps, nous évaluons la réactivité de l'opération `lostOne`.

A.1 Évaluation de l'opération readOnce

Dans cette partie nous évaluons la réactivité de l'opération `readOnce`. Dans la première partie, nous présentons le protocole de l'évaluation. Nous verrons notamment que lorsque le nombre de tuples reçus par l'entité qui exécute le `readOnce` est trop grand, le temps de réaction augmente de manière importante, en raison d'un problème de congestion. Dans la seconde partie, nous présentons comment adapter la période des requêtes de tuples afin d'éviter cette congestion.

A.1.1 Évaluation du temps de réaction

Dans cette première évaluation, le chariot connaît initialement l'ensemble des tuples qu'il contient et cherche à en détecter un nouveau. Nous évaluons le temps de réaction

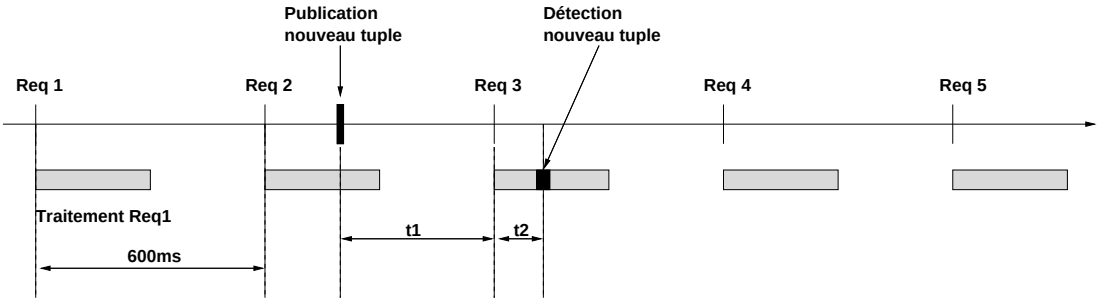

FIG. A.1 – Temps de réaction de l'opération `readOnce`

de l'opération, qui correspond au temps écoulé entre l'insertion d'un nouveau produit dans le chariot et la détection de ce produit par le chariot.

A.1.1.1 Protocole de l'évaluation

Le protocole de l'évaluation de l'opération `readOnce` est illustré sur la figure A.1. Dans un premier temps, *Cad* lit l'ensemble des produits qu'il contient grâce à une séquence d'opérations `readOnce`. Une fois l'ensemble des produits lus, le programme de *Cad* est bloqué sur la détection d'un nouveau tuple. Pour détecter un nouveau tuple, *Cad* diffuse à intervalle fixe une requête de tuples. Sur la figure la période des requêtes est de 600 millisecondes. Une fois le programme de *Cad* bloqué sur attente d'un nouveau tuple, *Prod* publie un nouveau tuple T_n à une date aléatoire pour simuler l'insertion d'un produit dans le chariot. Nous mesurons ensuite le temps de réaction t_r, qui correspond au temps écoulé entre la publication et la détection du tuple.

Sur la figure A.1, nous avons $t_r = t1 + t2$. Le temps $t1$ correspond au temps qui sépare la publication de T_n par *Prod* du lancement d'une nouvelle requête de tuples par *Cad*. Sachant que T_n est publié aléatoirement entre deux requêtes de tuples, la valeur moyenne de $t1$ est donc égale à la moitié de la période des requêtes de tuples.

Le temps $t2$ correspond au temps écoulé entre le lancement de la requête et la réception de T_n. Pour cette évaluation, $t2$ est indépendant du nombre total de tuples reçus par l'entité. En effet, en raison des structures de données utilisées par SPREAD pour stocker les tuples, lorsqu'une entité reçoit une requête de tuples, elle envoie en premier les tuples qu'elle a publié en dernier. Ainsi, *Cad* reçoit en premier le tuple correspondant au produit qui vient d'être inséré dans le chariot. Le temps de réaction moyen $t_r = t1 + t2$ doit donc être constant, quelque soit le nombre de tuples reçus. Si nous avions un calculateur par produit, le nouveau tuple ne serait pas forcément le premier reçu par *Prod* et le temps $t2$ vaudrait alors en moyenne la moitié du temps de traitement de tous les tuples reçus.

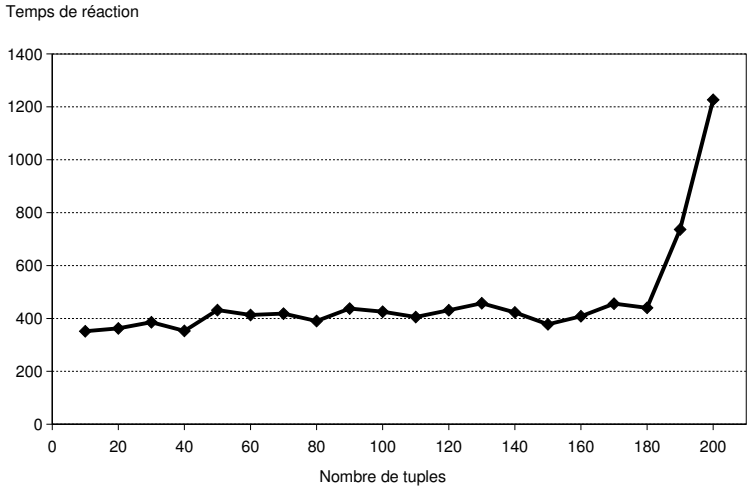

FIG. A.2 – Temps de réaction de l'opération `readOnce` pour une période de 600 millisecondes

A.1.1.2 Temps de réaction pour une période de 600 millisecondes

Nous évaluons le temps de réaction moyen t_r en fonction du nombre total de produits présents dans le chariot. Nous fixons la période des requêtes à 600 millisecondes. Les résultats sont présentés sur la figure A.2. Pour chaque point, nous mesurons le temps de réaction de 200 opérations. La courbe est divisée en deux zones distinctes. Lorsque le nombre de tuples total est inférieur à 180, le temps de réaction moyen est compris entre 350 et 450 millisecondes. Le temps de réaction moyen n'est pas constant, mais nous observons clairement un comportement en palier. Lorsque le nombre total de tuples total est supérieur à 180, le temps de détection augmente de manière importante.

Au-delà de 180 tuples, le temps de traitement des tuples par *Cad* est supérieur à la période des requêtes de nouveaux tuples. Nous avons alors un décalage temporel qui apparaît entre l'envoi des requêtes et leur traitement effectif (cf. figure A.3). Ce décalage entraîne une congestion au niveau de l'entité *Cad* qui reçoit les tuples, congestion qui est elle-même responsable de la perte de nombreux tuples. Les tuples peuvent être perdus lors de la traversée de chacune des couches du système : le réseau, le système d'exploitation, la machine d'exécution Java. Nous n'avons pas accès au fonctionnement interne de ces couches, il nous est donc difficile d'identifier la cause exacte de ces pertes de tuples.

La congestion entraîne la perte de tuples lorsque le nombre de tuples total est supérieur à 180. Chaque perte du tuple T_n retarde sa détection de 600 millisecondes. Lorsque la quantité de tuples perdus augmente, T_n peut être perdu lors de plusieurs requêtes de tuples successives, retardant d'autant sa détection. Afin de préserver la réactivité de l'application, lorsque la periode des requêtes est de 600 millisecondes, l'opération `readOnce` ne doit donc pas être utilisée avec plus de 180 tuples.

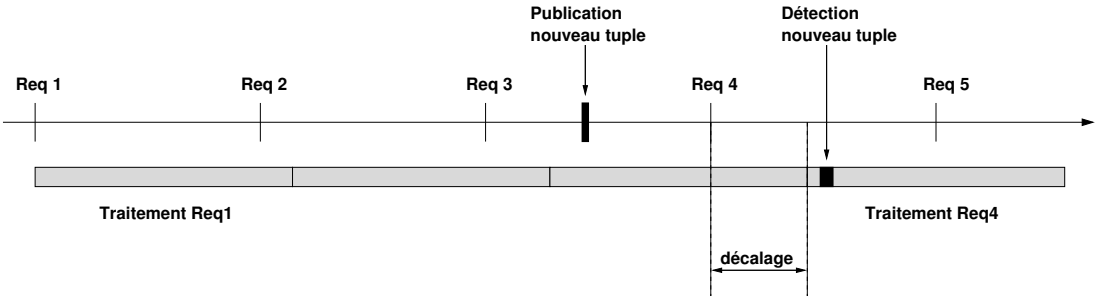

FIG. A.3 – Illustration du décalage temporel lorsque le temps de traitement des tuples est supérieur à la période des requêtes de l'opération `readOnce`.

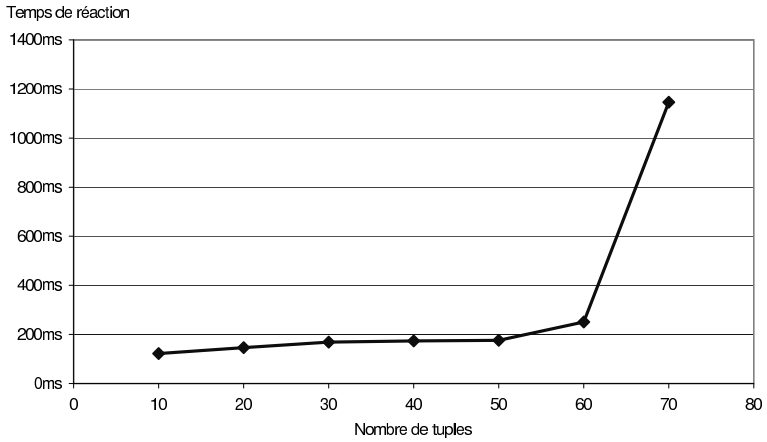

FIG. A.4 – Temps de réaction de l'opération `readOnce` pour un période de 150 millisecondes

A.1.1.3 Temps de réaction pour une période de 150 millisecondes

Pour étudier l'impact de la période des requêtes de tuples sur le temps de réaction, nous exécutons à nouveau l'évaluation précédente avec une période de 150 millisecondes. Les résultats sont présentés sur la courbe A.4. Nous observons également deux zones distinctes. Lorsque le nombre total de tuples est inférieur à 60, le temps de réaction est plus court et compris entre 170 et 230 millisecondes. Au delà de 60 tuples, le temps de réaction augmente très rapidement. Ici, en raison de la période des requêtes plus courte, le problème de congestion apparaît d'autant plus tôt.

Nous pouvons donc réduire le temps de réaction de l'opération `readOnce` en réduisant la période des requêtes. Cependant, avec une période plus faible, le nombre de tuples total que l'opération peut gérer est également plus faible.

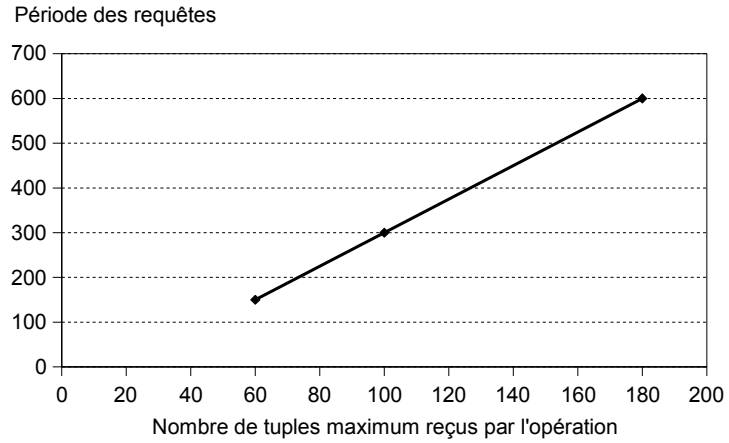

FIG. A.5 – Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `readOnce`

A.1.2 Limitation de la perte des tuples

Pour éviter de perdre des tuples, *Cad* ne devrait relancer une nouvelle requête de tuples que lorsqu'il a fini le traitement de la requête précédente. Cependant, en raison de la mobilité des entités, *Cad* ne peut savoir à l'avance le nombre de tuples que renverra chaque requête. C'est pourquoi, nous sommes obligés de découpler l'envoi des requêtes de leur traitement.

Afin d'éviter le problème de congestion, nous pouvons adapter la période des requêtes en fonction du profil de l'application. Si nous pouvons évaluer à l'avance le nombre maximum M de tuples que devra traiter *Cad*, nous pouvons minimiser la période des requêtes tout en évitant la congestion.

Nous cherchons donc maintenant à déterminer la période minimum des requêtes évitant la congestion, en fonction du nombre de tuples reçus par l'entité qui exécute l'opération. Pour cela, nous cherchons le nombre M de tuples à partir duquel le temps de traitement dépasse la période des requêtes. Ce point correspond au nombre de tuples pour lequel le temps de réaction augmente de manière importante. Sur la figure A.2, M vaut 180 et la période est de 600 millisecondes. Sur la figure A.2, M vaut 60 et la période est de 150 millisecondes. Nous exécutons à nouveau l'évaluation précédente afin d'obtenir M pour une périodes de 300 millisecondes. La figure[1] A.5 présente les résultats. Nous pouvons observer que la courbe est linéaire. Cette courbe nous permet donc de déterminer la période des requêtes en fonction du nombre maximal de tuples qu'aura à traiter une opération au cours d'une application.

[1]Les figures A.5 et A.8 sont identiques aux figures 7.2 et 7.3 présentées dans le chapitre 7. Nous les présentons à nouveau ici dans un souci de lisibilité.

A.1.3 Analyse des résultats

Nous avons évalué le temps de réaction de l'opération `readOnce` lors de la publication d'un nouveau tuple. La première évaluation nous montre que, si le nombre de tuples traités par l'opération `readOnce` est inférieur à un maximum M, le temps de réaction de l'opération est stable quel que soit le nombre de tuples traités par l'opération. Lorsque le nombre de tuples traités est supérieur à M, le temps de réaction augmente très rapidement en raison d'un problème de congestion.

Grâce à la seconde évaluation, nous pouvons adapter la période des requêtes de tuples en fonction du nombre maximum de tuples qu'une entité gère pendant l'application. La réactivité de l'opération peut donc être maximisée en fonction de ce nombre maximum de tuples. Cette adaptation suppose bien sûr de pouvoir évaluer à l'avance ce nombre maximum de tuples.

Actuellement, la réactivité obtenue limite le champ d'application de l'opération `readOnce` aux applications destinées aux êtres humains. Les temps de réaction obtenus ne nous permettent pas d'envisager l'utilisation de l'opération `readOnce` lorsque la réactivité doit être très élevée, comme dans le contexte de chaînes de production. Cependant, du point de vue d'un être humain, la réactivité obtenue est acceptable. Lorsque l'opération doit traiter au maximum 80 tuples par requête, nous pouvons régler la période des requêtes sur 150 millisecondes pour un temps de réaction compris entre 170 millisecondes et 230 millisecondes. Dans le cas de l'application du chariot par exemple, à moins que l'utilisateur ne surveille constamment l'afficheur du chariot, le prix qu'il lira sera donc toujours cohérent avec le contenu du chariot.

A.2 Évaluation de la réactivité de l'opération lostOne

Dans la partie précédente nous avons évalué la réactivité de l'opération `readOnce`. Dans cette partie, nous évaluons la réactivité de l'opération `lostOne`. Dans un premier temps, nous évaluons le temps de réaction du `lostOne` en fonction du nombre de tuples traités par l'opération. Tout comme pour l'opération `readOnce`, lorsque le nombre de tuples traités par l'opération dépasse un maximum, nous observons une dégradation des performances, due à la congestion de l'entité qui exécute l'opération. Dans un second temps, nous présentons une approche similaire à celle utilisée pour l'opération `readOnce` pour éviter cette congestion.

A.2.1 Évaluation du temps de réaction

Nous reconsidérons le scénario dans lequel le chariot calcule son prix au fur et à mesure des insertions et des retraits d'articles. Au début de l'évaluation, le chariot connaît l'ensemble des produits qu'il contient, nous retirons un article et mesurons le temps pris par le chariot pour détecter ce retrait. Nous rappelons qu'un tuple est déclaré comme disparu lorsque son estampille est âgée de plus de deux fois la période des requêtes de tuples.

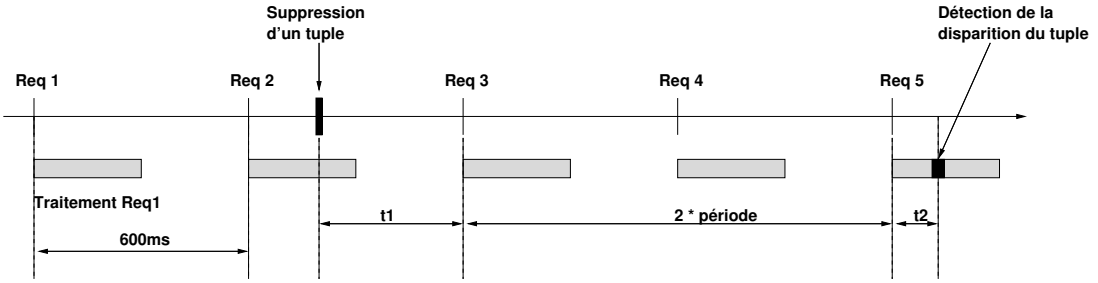

FIG. A.6 – Temps de réaction de l'opération `lostOne`

Le protocole de l'évaluation est présenté sur la figure A.6. Comme précédemment, nous effectuons cette évaluation à l'aide de deux Pocket PC. Le Pocket PC *Cad* représente le chariot et le Pocket PC *Prod* représente les produits. Initialement, *Cad* a lu l'ensemble des tuples publiés par *Prod* et son programme est bloqué sur une opération `lostOne`. Après un temps aléatoire, *Prod* supprime l'un de ses tuples pour simuler le retrait d'un article du chariot. Nous mesurons le temps de réaction t_r, qui correspond au temps écoulé entre la suppression du tuple par *Prod* et la détection de sa disparition par *Cad*. Nous avons $t_r = t1 + 2 * période + t2$. Le temps $t1$ représente le temps écoulé entre la suppression du tuple et la prochaine requête de tuples. Pour cette évaluation, la période des requêtes de tuples pour mettre à jour les estampilles est de 600 millisecondes, $t1$ vaux donc en moyenne $période/2$, soit 300 millisecondes. Le temps $t2$ correspond au temps de parcours des estampilles, afin de déterminer celles qui sont âgées de plus de deux périodes. Ici, nous avons donc $t_r > 2.5 * période = 1500ms$.

Les résultats sont présentés sur la figure A.7. Nous observons que le temps de réaction est stable et compris entre 1550 millisecondes et 1620 millisecondes. Lorsque le nombre de tuples traités par l'opération `lostOne` est supérieur à 160, le temps de traitement des tuples reçus par l'opération est supérieur à la période des requêtes de tuples, provoquant par la même un problème de congestion similaire à celui posé par l'opération `readOnce`.

Des pertes de tuples apparaissent donc au-delà de 160 tuples traités par l'opération `lostOne`. Lorsqu'un tuple est perdu plusieurs fois d'affiler, son estampille n'est plus mise à jour, *Cad* déclare alors ce tuple comme disparu et débloque l'opération `lostOne`. Dans ce cas là, nous avons une incohérence entre l'état du programme et l'état du monde physique, car un tuple toujours présent est considéré comme disparu. Au-delà de 160 tuples, nous ne sommes plus en mesure d'évaluer correctement le temps de réaction moyen de l'opération `lostOne`, car plus de 90% des opérations se débloquent sans qu'aucun tuple n'ait encore disparu. Nous considérons que l'opération `lostOne` n'est plus utilisable au-delà de 160 tuples.

A.2.2 Limitation de la perte des tuples

Dans la partie précédente, nous avons vu que, lorsque le nombre de tuples reçus par l'entité qui exécute l'opération `lostOne` est trop grand, des tuples sont perdus.

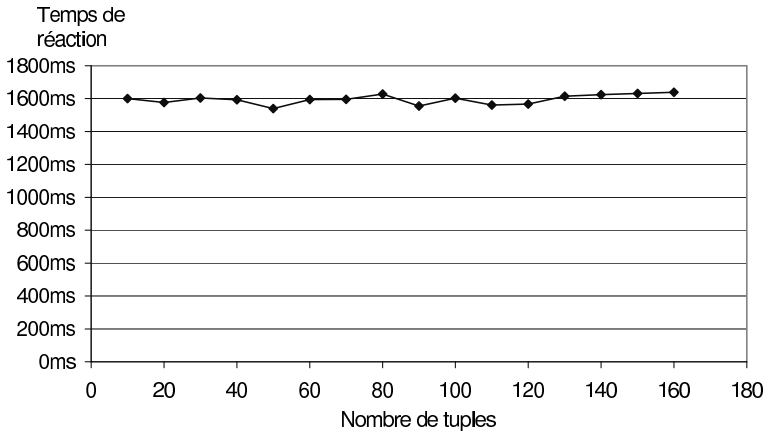

FIG. A.7 – Temps de réaction de l'opération `lostOne` pour une période de 600 millisecondes

Ces pertes de tuples débloquent des opérations `lostOne` sans que ces tuples n'aient réellement disparus ; ils sont toujours présents physiquement sur l'entité *Prod*.

Tout comme pour l'opération `readOnce`, nous ne pouvons pas savoir à l'avance le nombre de tuples que recevra une requête de tuples. Ainsi, nous ne pouvons pas adapter dynamiquement la période des requêtes. Cependant, si nous connaissons par avance le nombre de tuples maximal M que recevra Cad, nous pouvons déterminer la période minimale des requêtes qui permet d'éviter la congestion de Cad. Grâce à l'évaluation précédente, nous savons que si M vaut 160 la période des requêtes doit être de 600 millisecondes. Nous recherchons la valeur de M pour des périodes de 150 millisecondes et 300 millisecondes. Pour cela, il nous suffit de rechercher la valeur de M à partir de laquelle les opérations `lostOne` se débloque sans qu'un tuple n'ait disparu. Nous présentons les résultats sur la figure A.8.

Comme la courbe présentée sur la figure A.5, cette courbe est également linéaire. Elle nous permet de déterminer la période des requêtes en fonction du nombre maximum de tuples qu'à traiter une opération `lostOne`.

A.2.3 Analyse des résultats

La première évaluation nous montre que, si le nombre de tuples traités par l'opération `lostOne` est inférieur à un maximum M, le temps de réaction de l'opération est constant quel que soit le nombre de tuples traités. Lorsque le nombre de tuples traités est supérieur à M les opérations sont débloquées sans qu'un tuple n'ait réellement disparu, en raison du problème de congestion.

Nous remarquons également que le temps de réaction de l'opération `lostOne` est nettement supérieur à celui de l'opération `readOnce`, pour la même période des requêtes de tuples. Ceci est dû au fonctionnement de `lostOne` qui nécessite d'attendre un temps minimum avant de pouvoir confirmer qu'un tuple a réellement disparu. Si nous cherchons

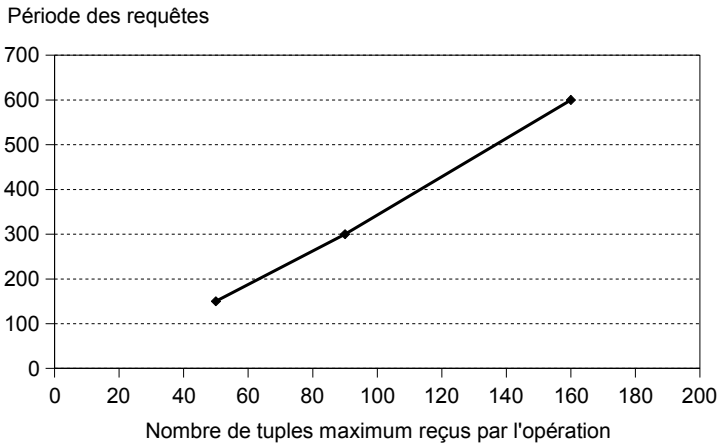

FIG. A.8 – Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `lostOne`

à réduire ce temps pour améliorer la réactivité, nous augmentons le risque d'avoir des opérations `lostOne` débloquées sans qu'un tuple n'ait réellement disparu, menant à des incohérences entre l'état des programmes des entités et l'état du monde physique.

La seconde évaluation nous permet d'adapter la période des requêtes de tuples en fonction du nombre maximum de tuples à traiter par l'opération `lostOne`. Cette période détermine directement la réactivité de l'opération, qui sera supérieure à deux périodes et demie.

Du point de vue de l'utilisateur la réactivité est cette fois bien perceptible. Pour l'application du chariot, il est donc possible que l'utilisateur retire un article du chariot et regarde l'écran de contrôle du chariot avant que le prix n'ait été mis à jour. Cependant, l'incohérence entre le contenu du chariot et le prix affiché n'est que transitoire : une fois le produit détecté, le prix affiché sera cohérent avec le contenu du chariot.

A.3 Bilan

Dans cette annexe, nous avons évalué la réactivité des opérations `readOnce` et `lostOne`. Nous avons vu que le temps de réaction des opérations est stable et directement fonction de la période des requêtes de tuples. De plus, si le nombre de tuples reçus par ces opérations est trop grand, un problème de congestion apparaît sur l'entité qui exécute l'opération. Finalement, pour chaque opération, nous avons évalué la période des requêtes afin d'éviter le problème de congestion, en fonction du nombre maximal de tuples traités par l'opération.

Du point de vue de l'utilisateur, la réactivité des opérations est acceptable. Cependant, la réactivité actuelle ne nous permet pas d'envisager d'utiliser ces opérations pour réaliser des applications nécessitant une réactivité élevée, comme des chaînes de production.

Annexe B

Similink

Dans le chapitre 9, nous avons présenté une nouvelle définition pour le contexte de l'utilisateur basée sur la notion de proximité. Dans ce chapitre, nous présentons l'application contextuelle Similink qui est une illustration directe de cette définition. Similink n'est pas directement une application d'informatique diffuse. Similink permet à l'utilisateur de naviguer sur le Web selon la dimension thématique et la dimension temporelle en plus de la dimension topologique habituelle.

B.1 Présentation

Dans cette partie nous commençons par présenter le principe de fonctionnement de Similink. Nous poursuivons en présentant deux scénarios typiques d'utilisation. Finalement, nous détaillons le schéma de navigation proposé par Similink.

B.1.1 Principe de fonctionnement

Similink est une application qui permet à l'utilisateur de naviguer sur le Web selon plusieurs dimensions. Le Web est un système d'information composé de pages qui sont liées les unes aux autres par des liens hypertextes. Ces liens permettent à l'utilisateur de naviguer dans le système d'information en sautant de page en page. Ce mécanisme de navigation est contextuel. Lorsqu'une personne ajoute des liens dans une pages Web qu'elle crée, elle définit le contexte de cette page. Autrement dit, elle définit les pages qui en sont proches. A partir de ce contexte, l'utilisateur peut sauter vers une autre page. Nous considérons que l'utilisateur navigue selon la dimension topologique.

Actuellement, nous naviguons sur le Web uniquement via les liens présents dans les pages. Sur certaines pages les possibilités de navigation sont donc limitées, en raison du manque de liens. Similink propose à l'utilisateur des liens supplémentaires, qui lui offrent de nouvelles directions de navigation. Ces liens sont organisés selon deux dimensions, la dimension thématique et la dimension temporelle. Par exemple, sur le site d'un logiciel, Similink propose à l'utilisateur des liens vers des sites de logiciels concurrents, mais également des liens vers les pages précédemment visitées.

Similink propose à l'utilisateur des liens vers des pages qui sont proches thématiquement de la page courante. Ces liens sont déterminés à l'aide d'un moteur de recherche, qui renvoie les pages proches textuellement de la page courante. Le texte contenu dans une page définit son thème, nous considérons donc que des pages proches selon la dimension textuelle sont également proches selon la dimension thématique.

En plus de la dimension topologique habituelle, Similink propose donc à l'utilisateur de naviguer selon la dimension thématique et la dimension temporelle. Ainsi, le contexte de l'utilisateur est représenté par trois ensembles de liens, chacun déterminé selon une dimension. Ces dimensions sont indépendantes l'une de l'autre, le contexte de l'utilisateur est donc représenté par l'union de ces trois ensembles. Il contient des pages qui sont proches thématiquement *ou* temporellement *ou* selon la dimension topologique. Lorsque l'utilisateur sélectionne un lien présent dans son contexte, il saute à la page pointée par ce lien et son contexte est mis à jour.

B.1.2 Scénarios d'utilisation

Similink permet à l'utilisateur d'effectuer des recherches sur le Web par navigation (cf. 9.2.4), afin d'explorer rapidement une zone thématique du Web. Considérons un utilisateur cherchant une conférence dans le domaine des interactions homme machine. L'utilisateur commence par aller sur la page d'une conférence qu'il connaît dans ce domaine. Lorsqu'il se trouve sur cette page, Similink lui propose des liens vers des pages qui sont proches thématiquement. Parmi ces liens, se trouvent plusieurs liens vers d'autres conférences du même domaine. L'utilisateur choisit l'un de ces liens et saute donc à la page correspondante. Arrivé sur cette nouvelle page, Similink lui propose de nouveaux des liens vers des conférences du domaine voulu. Le contexte thématique de l'utilisateur contient la page d'où vient l'utilisateur, des pages qui étaient déjà proposées sur la page précédente, mais également des liens vers des conférences qui n'étaient pas proposés précédemment. Ainsi, en sautant de page en page selon la dimension thématique, l'utilisateur peut trouver rapidement plusieurs conférences dans un domaine voulu.

Considérons maintenant un utilisateur localisé sur la page Web de la tour Eiffel. Selon la dimension thématique, Similink lui propose des liens vers plusieurs autres lieux touristiques de Paris. L'utilisateur choisit alors de sauter vers le site de parc d'attraction Disneyland. Arrivée sur ce site, Similink lui propose des liens vers d'autres attractions touristiques et notamment vers d'autres parcs d'attraction. Grâce à la dimension temporelle, l'utilisateur revient sur la page de la Tour Eiffel. Cette fois, il choisit de sauter vers la page du musée d'Orsay. Arrivé sur cette page, Similink lui propose plusieurs liens vers d'autres musées parisiens. Nous voyons ici qu'en fonction des liens choisis, l'utilisateur peut sélectionner la direction thématique qu'il suit. Ainsi, dans le premier cas l'utilisateur à choisit une direction "ludique", tandis que dans le second cas il choisit une direction "culturelle".

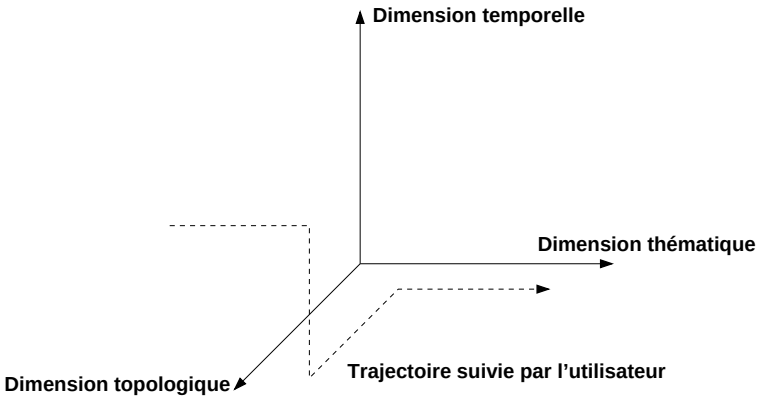

FIG. B.1 – Exemple de navigation multidimensionnelle sur le Web

B.1.3 Schéma de navigation

Similink représente le contexte de l'utilisateur selon trois dimensions : la dimension thématique, la dimension temporelle et la dimension topologique. Ainsi, lorsque l'utilisateur consulte une page Web, il peut choisir parmi trois dimensions pour se déplacer à la page suivante. Par exemple, l'utilisateur peut arriver à une page en se déplaçant selon la dimension thématique et poursuivre sa navigation selon la dimension topologique. La figure B.1 illustre ce schéma de navigation selon plusieurs dimensions.

Le contexte de l'utilisateur est composé de liens vers d'autres pages, qui sont proches de la page courante. Parmi ces liens, seuls les liens contenus dans le texte de la page, existent initialement. Similink permet donc de créer dynamiquement des liens entres les pages, qui offrent de nouvelles directions de navigation à l'utilisateur.

Actuellement, Similink ne repose que sur les dimensions thématique, temporelle et topologique. Cependant, nous pourrions envisager d'ajouter la dimension physique au contexte de l'utilisateur. Les entités de l'espace physique seraient alors associées à des liens et le contexte de l'utilisateur contiendrait également les liens physiquement proches de l'utilisateur. L'utilisateur pourrait donc atteindre une page en se déplaçant physiquement, puis naviguer selon la dimension thématique pour trouver des pages similaires à cette page. Par exemple, en arrivant devant un produit dans un magasin, l'utilisateur découvrirait physiquement la page Web de ce produit. Il aurait accès ensuite grâce à la dimension thématique à des pages Web de produits du même type.

B.2 Réalisation

Similink est une extension pour le navigateur Web Firefox. Le mécanisme d'extension de Firefox permet aux développeurs tiers d'ajouter des fonctionnalités au navigateur. Ces fonctionnalités sont directement disponibles depuis l'interface graphique du navigateur. Les extensions sont programmées avec le langage Javascript.

Similink se présente sous la forme d'un panneau latéral, affiché à gauche de la fenêtre

FIG. B.2 – Interface graphique de Similink

de navigation (cf. figure B.2). Ce panneau est divisé en deux et présente le contexte thématique de l'utilisateur d'une part, et son contexte temporel d'autre part. Les liens thématiques et temporels sont ordonnés en fonction de leur proximité relativement à la page courante. Similink ne présente pas le contexte topologique car les liens sont déjà présents dans la page.

Les liens proches thématiquement sont déterminés à l'aide du moteur de recherche Google. Les liens proches temporellement sont extraits des structures de données internes de Firefox.

A côté de chaque lien, Similink présente une icône qui est extraite du code source de la page correspondante. Cette icône facilite la navigation, car elle permet à l'utilisateur de distinguer et d'identifier plus facilement les liens proposés par Similink.

Afin d'améliorer la réactivité des mises à jour de contexte lorsque l'utilisateur navigue, Similink repose sur un mécanisme de cache qui sauve pour chaque page visitée les pages proches thématiquement. Ce mécanisme économise donc des requêtes au moteur de recherche pour les pages que l'utilisateur visite régulièrement. Une entrée dans le cache est mise à jour au bout d'une semaine. Cette mise à jour est "paresseuse", dans le sens où elle n'est faite que lorsque l'utilisateur visite la page correspondante.

B.3 Bilan

Dans cette partie nous avons présenté Similink, qui permet à l'utilisateur de naviguer sur le Web selon la dimension thématique et la dimension temporelle. Similink est une application directe de la définition de contexte présentée dans le chapitre 9. Similink propose un mécanisme de recherche par navigation, qui permet à l'utilisateur d'explorer rapidement une zone thématique du Web. Similink permet également à l'utilisateur de choisir la direction thématique qu'il emprunte.

Similink a été intégré dans le navigateur Web Firefox. Nous avons rendu Similink disponible sur le Web. Les principaux retours d'utilisateurs concernent l'interface graphique, qui prend selon eux trop de place. Nous envisageons donc de modifier l'interface afin de réduire son encombrement.

Bibliographie

[1] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide : A Mobile Context-aware Tour Guide. *ACM Wireless Networks*, 3 :421–433, 1997.

[2] M. Banâtre, P. Couderc, J. Pauty, and M. Becus. Ubibus : Ubiquitous Computing to Help Blind People in Public Transport. In *Mobile HCI 2004*, pages 310–314, 2004.

[3] B. Brumitt, J. Krumm, B. Meyers, and S. Shafer. Ubiquitous computing and the role of geometry. *Personal Communications*, 7(5) :41–43, 2000.

[4] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving : Technologies for Intelligent Environments. In *Handheld and Ubiquitous Computing (HUC'00)*, September 2000.

[5] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2) :225–267, 1996.

[6] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[7] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide : some issues and experiences. In *CHI*, pages 17–24, 2000.

[8] P. Couderc and M. Banâtre. Ambient Computing Applications : An Experience with the SPREAD Approach. In *Annual Hawaii International Conference on System Sciences (HICSS'03)*, 2003.

[9] P. Couderc and M. Banâtre. Spreading the web. In Springer, editor, *Personnal Wireless Communications (PWC'03)*, pages 375–384, 2003.

[10] Paul Couderc. *Mobilité contextuelle dans les système d'information.* PhD thesis, Université de Rennes 1, 2000.

[11] G. Cugola and G. Picco. PeerWare : Core Middleware Support for Peer-To-Peer and Mobile Systems, 2001.

[12] D. López de Ipiña, P. R. S. Mendonça, and A. Hopper. TRIP : A Low-Cost Vision-Based Location System for Ubiquitous Computing. *Personal and Ubiquitous Computing*, 6(3) :206–219, 2002.

[13] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness, Conference on Human Factors in Computing Systems (CHI'00)*, 2000.

[14] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16 :2–4, 2001.

[15] M. H. Dunham, A. Helal, and S. Balakrishnan. A Mobile Transaction Model That Captures Both the Data and Movement Behavior. *Mobile Networks and Applications*, 2(2) :149–162, 1997.

[16] F. Ganovelli and F. Ponchio and C. Rocchini. Fast tetrahedron-tetrahedron overlap algorithm. *ACM Journal of Graphics Tools*, 7(2), 2003.

[17] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1) :80–112, 1985.

[18] H.-W. Gellersen, M. Beigl, and Holger Krull. The MediaCup : Awareness Technology Embedded in an Everyday Object. In *Handheld and Ubiqutious Computing (HUC'99)*, 1999.

[19] L. Girod, V. Bychkobskiy, J. Elson, and D. Estrin. Locating Tiny Sensors in Time and Space : A Case Study. In *International Conference on Computer Design (ICCD'02)*, 2002.

[20] J. Goodman, P. Gray, K. Khammampad, and S. Brewster. Using Landmarks to Support Older People in Navigation. In *Mobile HCI*, 2004.

[21] K Goto and Y Kambayashi. A New Passenger Support System for Public Transport using Mobile Database Access. In *VLDB*, 2002.

[22] J. Gray. Notes on Data Base Operating Systems. In *Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, 1978.

[23] C. G Gurrin, G. J. F. Jones, H. Lee, N. O'Hare, A. F. Smeaton, and N. Murphy. Mobile Access to Personal Digital Photograph Archives. In *Mobile HCI*, 2005.

[24] R. H. Güting. An Introduction to Spatial Database Systems. *The VLDB Journal - The International Journal on Very Large Data Bases*, 3(4) :357–399, 1994.

[25] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8) :57–66, August 2001.

[26] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, pages 93–104, 2000.

[27] L.E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.W. Gellersen. Smart-Its Friends : A Technique for Users to Easily Establish Connections between Smart Artefacts. In *International Conference on Ubiquitous Computing (UbiComp'01)*, 2001.

[28] S. S. Intille, K. Larson, J. S. Beaudin, J. Nawyn, E. Munguia Tapia, and P. Kaushik. A living laboratory for the design and evaluation of ubiquitous computing technologies. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*, 2004.

[29] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next Century Challenges : Mobile Networking for "Smart Dust". In *International Conference on Mobile Computing and Networking (MobiCom'99))*, pages 271–278, 1999.

[30] T. Kindberg. Implementing physical hyperlinks using ubiquitous identifier resolution. In *Proceedings of the 11th international conference on World Wide Web*, pages 191–199. ACM Press, 2002.

[31] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, Places, Things : Web Presence for the Real World, 2000.

[32] U. Kubach and K. Rothermel. Exploiting Location Information for Infostation-Based Hoarding. . In *International Conference on Mobile Computing and Networking (MobiCom'01)*, pages 15–27, 2001.

[33] M. Langheinrich, F. Mattern, K. Römer, and H. Vogt. First Steps Towards an Event-Based Infrastructure for Smart Things. In *Ubiquitous Computing Workshop (PACT 2000)*, 2000.

[34] I. MacColl, D. Millard, C. Randell, and A. Steed. Shared Visiting in EQUATOR City. In *International Conference on Collaborative virtual environments*, pages 88–94. ACM Press, 2002.

[35] M. P. G. Oliveira, E. Bauzer M., and C. A. Davis. Planning the Acoustic Urban Environment : a GIS-centered approach. In *Proceedings of the seventh ACM International Symposium on Advances in Geographic Information Systems*, pages 128–133. ACM Press, 1999.

[36] A. Padovitz, S. Wai Loke, and A. Zaslavsky. Towards a theory of context spaces. In *Pervasive Computing and Communications Workshops PERCOMW'04*, 2004.

[37] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *International Symposium on Wearable Computers (ISWC'98)*, pages 92–99, 1998.

[38] J. Pauty, M. Banâtre, and P. Couderc. Logical versus Physical Programming for Ubiquitous Computing. In *Workshop on Intelligent Solutions in Embedded Systems (WISES'03)*, June 2003.

[39] J. Pauty, P. Couderc, and M. Banâtre. Atomic Token Passing in the Context of Spontaneous Communications. In *Workshop on Applications and Services in Wireless Networks (ASWN'05)*, 2005.

[40] J. Pauty, P. Couderc, and M. Banâtre. Spatial Programming : Using the Physical World as a Computing System. In *UbiPhysics : Designing for Physically Integrated Interaction (UBICOMP 2005 workshops)*, 2005.

[41] J. Pauty, P. Couderc, and M. Banâtre. Using Context to Combine Virtual and Physical Navigation. Technical Report 5496, INRIA, February 2005.

[42] J. Pauty, P. Couderc, and M. Banâtre. Using Context to Navigate Through a Photo Collection. In *Mobile HCI'05*, 2005.

[43] J. Pauty, P. Couderc, and M. Banâtre. Architectures de Systèmes pour l'Informatique Diffuse. *Technique et Science Informatiques*, 25(2), 2006.

[44] E. Peytchev and C. Claramunt. Experiences in Building Decision Support Systems for Traffic and Transportation GIS. In *Proceedings of the ninth ACM International Symposium on Advances in Geographic Information Systems*, pages 154–159. ACM Press, 2001.

[45] G. P. Picco, A. L. Murphy, and G.-C. Roman. LIME : Linda Meets Mobility. In *International Conference on Software Engineering (ICSE'99)*, pages 368–377, 1999.

[46] G. P. Picco, A. L. Murphy, and G.-C. Roman. Developing mobile computing applications with LIME. In *International Conference on Software Engineering (ICSE'00)*, pages 766–769, 2000.

[47] The VINT Project. The ns manual. http ://www.isi.edu/nsnam/ns/doc/.

[48] J. M. Rabaey, M. J. Ammer, J. L. da Silva Jr, and D. P. S. Roundy. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. *IEEE Computer*, pages 42–48, 2000.

[49] M. Raynal. Revisiting the Non-Blocking Atomic Commitment Problem in Distributed Systems. In *IPPS IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pages 116–133, April 1997.

[50] J. Rekimoto and Y. Ayatsuka. Cybercode : Designing augmented reality environments with visual tags. In *Proceedings of Designing Augmented Reality Environments (DARE'00)*, pages 1–10. ACM Press, 2000.

[51] J. Rekimoto and K. Nagao. The World Through the Computer : Computer Augmented Interaction with Real World Environments. In *ACM Symposium on User Interface Software and Technology (UIST'95)*, pages 29–36, 1995.

[52] M. Rohs and P. Zweifel. A Conceptual Framework for Camera Phone based Interaction Techniques. In *PERVASIVE*, 2005.

[53] G. Roussos, J. Tuominen, L. Koukara, O. Seppala, P.s Kourouthanasis, G. Giaglis, and J. Frissaer. A Case Study in Pervasive Retail. In *Proceedings of the second international workshop on Mobile commerce*, pages 90–94. ACM Press, 2002.

[54] K. Römer and T. Schoch. Infrastructure concepts for tag-based ubiquitous computing applications. In *Workshop on Concepts and Models for Ubiquitous Computing (UbiComp'02)*, September 2002.

[55] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. Information Processing and Management. *Information Processing and Management*, 24(5) :513–523, 1988.

[56] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94)*, Santa Cruz, CA, US, 1994.

[57] A. Schmidt. Implicit Human Computer Interaction Through Context. *Personal Technologies*, 4(2), 2000.

[58] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to Context than Location. *Computers and Graphics*, 23(6) :893–901, 1999.

[59] Frank Siegemund and Christian Floerkemeier. Interaction in pervasive computing settings using bluetooth-enabled active tags and passive rfid technology together with mobile phones. In *International Conference on Pervasive Computing and Communications (PerComm'03)*, pages 378–387, March 2003.

[60] D. Touzet, F. Weis, and M. Banâtre. Sensing and filtering surrounding data : The persend approach. In *Mobile HCI Workshop on Mobile and Ubiquitous Information Access*, pages 283–297, 2003.

[61] A. Troël, F. Weis, and M. Banâtre. Découverte automatique entre terminaux mobiles communicants. Technical Report 1570, IRISA, 2003.

[62] D. Wagner, T. Pintaric, and F. Ledermann. Towards massively multi-user augmented reality on handheld devices. In *PERVASIVE*, 2005.

[63] X. Wang. Ontology-Based Context Modeling and Reasoning using OWL. In *Context Modeling and Reasoning Workshop at PerCom*, 2004.

[64] R. Want, K. P. Fishkin, A. Gujar, and B. L. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 370–377. ACM Press, 1999.

[65] A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. *IEEE Personnel Communications*, 4(5) :42–47, 1997.

[66] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM, Back to the Real World, Special issue on Computer Augmented Environments*, 36(7) :75–84, 1993.

Table des figures

1.1 Capture du contexte depuis l'environnement physique. Les données captées sont d'abord regroupées puis analysées pour déduire le contexte de l'utilisateur 17

2.1 Un espace de données partagées physiquement 22
2.2 Un guide virtuel purement diffus 23
2.3 Capture du contexte dans WebWalker 27
2.4 Partage des informations dans PeerWare 28

3.1 Adressage physique basé sur des étiquettes électroniques. Les adresses sont distribuées dans l'espace physique et permettent d'accéder indirectement aux données 32
3.2 Approche basée sur le marquage électronique 33

5.1 Illustration du problème de la taille du volume occupé par les données. Gauche : avec un grand volume. Droite : avec un petit volume. En utilisant un volume trop grand, la porte peut s'ouvrir alors que l'utilisateur ne veut pas la prendre. 45
5.2 Une entité peur accéder à une donnée si elle se trouve à l'intérieur de la forme correspondante 46
5.3 Exemple de synchronisation physique. Le programme de l'entité statique est synchronisé sur la découverte d'un tuple porté par l'entité mobile et donc sur la rencontre de cette entité. 47
5.4 Protocole de l'opération `read`. L'opération interroge d'abord l'espace de tuples local et, si elle ne trouve pas de tuple correspondant, interroge ensuite les entités voisines. 50
5.5 Illustration du problème de la forme des données. Gauche : les données sont sphériques, le crayon est considéré comme étant à l'intérieur du pot. Droite : les données ont une forme adaptée à la forme de l'objet correspondant, le crayon est considéré comme étant à l'extérieur du pot. 53

6.1 Formes disponibles pour les tuple 56
6.2 Adressage géométrique : un tuple est accessible lorsque la forme du tuple est en intersection avec la zone d'adressage 57

6.3 Définition d'une zone de paiment pour calculer le prix d'un chariot . . . 58
6.4 Décomposition en tétraèdres d'une boite, d'un cône et d'un secteur. Le cylindre est approché par deux boites, elles-mêmes décomposées en tétraèdres. 59

7.1 Le chariot calcule lui-même son prix, en fonction des insertions et retraits de produits 64
7.2 Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `readOnce` 69
7.3 Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `lostOne` 70

8.1 Utilisation de l'opération `take` pour réaliser l'application du taxi 72
8.2 L'opération `take` peut se terminer dans quatre états différents 73
8.3 Protocole de l'opération `take` 75
8.4 Une contrainte géométrique garantit un temps de communication minimum entre les entité A et B 78
8.5 Scénario utilisés dans la simulation 78
8.6 Taux de défaillance sans réémission. Scénario 1 : les entités se rapprochent 79
8.7 Taux de défaillance avec deux réémissions. Scénario 1 : les entités se rapprochent. 81
8.8 Taux de défaillance en fonction de la zone où les opérations sont démarrées, avec 2 réémissions. Scénario 2 : les entités s'éloignent. 82

9.1 Contexte physique d'une personne 91
9.2 Exemple d'une navigation virtuelle à travers un ensemble de photos . . . 95
9.3 Construction d'un contexte multidimensionnel 100
9.4 Construction d'un contexte multidimensionnel avec une dépendance entre les deux dimensions 101

10.1 Architecture logicielle du navigateur de photos 108
10.2 Image a : capture d'écran de la navigation physique. La position de l'utilisateur est représentée par une croix. Image b : capture d'écran de la navigation virtuelle. Image c : capture d'écran de la navigation temporelle. 109
10.3 Filtrage physique. L'espace autour de l'utilisateur est divisé en huit secteurs et trois photos sont proches de lui : A, B, C. Les photos sont associées à un secteur en fonction de leur orientation. La photo A est associée au secteur 1, la photo B au secteur 8 et la photo C au secteur 5. 110

A.1 Temps de réaction de l'opération `readOnce` 118
A.2 Temps de réaction de l'opération `readOnce` pour une période de 600 millisecondes 119
A.3 Illustration du décalage temporel lorsque le temps de traitement des tuples est supérieur à la période des requêtes de l'opération `readOnce`. . 120

A.4 Temps de réaction de l'opération `readOnce` pour un période de 150 millisecondes 120
A.5 Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `readOnce` 121
A.6 Temps de réaction de l'opération `lostOne` 123
A.7 Temps de réaction de l'opération `lostOne` pour une période de 600 millisecondes 124
A.8 Adaptation de la période des requêtes de tuples en fonction du nombre de tuples maximum gérés par l'opération `lostOne` 125

B.1 Exemple de navigation multidimensionnelle sur le Web 129
B.2 Interface graphique de Similink 130

Publications

Journal

[1] J. Pauty, P. Couderc, and M. Banâtre. Architectures de Systèmes pour l'Informatique Diffuse. *Technique et Science Informatiques*, 25(2), 2006.

Conférences

[2] J. Pauty, P. Couderc, and M. Banâtre. Using Context to Navigate Through a Photo Collection. In *Mobile HCI'05*, 2005.

[3] M. Banâtre, P. Couderc, J. Pauty, and M. Becus. Ubibus : Ubiquitous Computing to Help Blind People in Public Transport. In *Mobile HCI'04*, pages 310–314, 2004.

[4] J. Pauty and G. Cabillic. A Checkpoints Mechanism for Mobile Java Applications. In *Parallel and Distributed Computing and Networks (PDCN'04)*, 2004.

Workshops

[5] J. Pauty, P. Couderc, and M. Banâtre. Spatial Programming : Using the Physical World as a Computing System. In *In UbiPhysics : Designing for Physically Integrated Interaction (UBICOMP 2005 workshops)*, 2005.

[6] J. Pauty, P. Couderc, and M. Banâtre. Atomic Token Passing in the Context of Spontaneous Communications. In *Workshop on Applications and Services in Wireless Networks (ASWN'05)*, 2005.

[7] J. Pauty, M. Banâtre, and P. Couderc. Logical versus Physical Programming for Ubiquitous Computing. In *Workshop on Intelligent Solutions in Embedded Systems (WISES'03)*, June 2003.

Rapports de recherche

[8] J. Pauty, P. Couderc, and M. Banâtre. Using Context to Combine Virtual and Physical Navigation. Technical Report 5496, INRIA, February 2005.

[9] J. Pauty, P. Couderc, and M. Banâtre. Atomic Token Passing in the Context of Spontaneous Communications. Technical Report 5445, INRIA, January 2005.

[10] J. Pauty, P. Couderc, and M. Banâtre. Synthèse des Méthodes de Programmation en Informatique Contextuelle. Technical Report 5094, INRIA, January 2004.

[11] J. Pauty and G. Cabillic. Local Checkpointing for Embedded Java Applications. Technical Report 4826, INRIA, 2003.

Résumé

L'informatique diffuse a pour objectif d'assister implicitement l'utilisateur dans ses tâches quotidiennes. Une application d'informatique diffuse est utilisée directement depuis l'espace physique et dépend de son organisation géométrique. Les services fournis par les applications d'informatique diffuse dépendent souvent par exemple de la position géographique de l'utilisateur ou de la proximité relative d'objets physiques.

Dans cette thèse, nous étudions dans un premier temps le rôle de la géométrie dans le cadre de la programmation des applications d'informatique diffuse. Nous proposons un modèle de programmation basé sur la synchronisation des programmes sur les mouvements des personnes et objets physiques. Afin de conserver la cohérence entre l'état des programmes et l'état du monde physique, ce modèle de programmation nous permet de définir précisément la configuration géométrique des synchronisations, directement au niveau du code des applications.

Dans un second temps, nous étudions le rôle de la géométrie dans la construction, l'organisation et la navigation dans un système d'information. Nous proposons une définition du contexte dans un système d'information qui repose sur la structure géométrique du système. Nous présentons ensuite l'utilisation de ce contexte pour naviguer virtuellement et physiquement dans le système d'information.

Mots clés Informatique diffuse, programmation géométrique, navigation contextuelle.

Abstract

The goal of ubiquitous computing is to support implicitly the user during his everyday tasks. A ubiquitous computing application is used directly inside the physical space and depends on the geometric organization of the physical space. The services provided be a ubiquitous computing application often depend on the geographical location of the user or the relative proximity of physical objects.

In this PhD thesis, we study the role of geometry in the development of ubiquitous computing applications. We propose a programming model that is based on synchronizing programs on the movements of people and physical objects. To keep the consistency between the programs' state and the physical space's state, this programming model enables us to precisely define the geometric configuration of the synchronizations.

We also study the role of geometry to construct, organize and navigate through an information system. We propose a definition of context in an information system that relies on the geometric structure of the system. Afterward, we present how to use this context to navigate physically and virtually through the information system.

Keywords Ubiquitous computing, geometric programming, context-aware navigation.