APISENSE® : une plate-forme répartie pour la conception,le déploiement et l'exécution de campagnes de collecte de données sur des terminaux intelligents

Nicolas Haderer

Département de formation doctorale en informatique
UFR IEEA

École doctorale SPI Lille

APISENSE® : une plate-forme répartie pour la conception, le déploiement et l'exécution de campagnes de collecte de données sur des terminaux intelligents

THÈSE

présentée et soutenue publiquement le 05/11/2014

pour l'obtention du

Doctorat de l'Université Lille I
(spécialité informatique)

par

Haderer Nicolas

Composition du jury

Rapporteurs :	Chantal Taconet, *TELECOM SudParis, France*
	Ernesto Exposito, *INSA Toulouse, France*
Examinateurs :	Stéphane Frénot, *INSA Lyon, France*
	Luigi Lancieri, *Université Lille I, France*
Directeur :	Lionel Seinturier, *Université Lille I, France*
Co-encadrant :	Romain Rouvoy, *Université Lille I, France*

Laboratoire d'Informatique Fondamentale de Lille – Inria Lille - Nord Europe
UMR Lille 1/CNRS 8022

RÉSUMÉ

Le *mobile crowdsensing* est une nouvelle forme de collecte de données exploitant *la foule* de terminaux intelligents déjà déployés à travers le monde pour collecter massivement des données environnementales ou comportementales d'une population. Ces dernières années, ce type de collecte de données a suscité l'intérêt d'un grand nombre d'acteurs industriels et académiques dans de nombreux domaines tels que l'étude de la mobilité urbaine, la surveillance de l'environnement, la santé ou l'étude des comportements socioculturels. Cependant, le *mobile crowdsensing* n'en n'est qu'à ses premiers stades de développement, et de nombreux défis doivent encore être relevés pour pleinement profiter de son potentiel. Ces défis incluent la protection de la vie privée des utilisateurs, les ressources énergétiques limitées des terminaux mobiles, la mise en place de modèles de récompense et de déploiement adaptés pour recruter les utilisateurs les plus à même de collecter les données désirées, ainsi que faire face à l'hétérogénéité des plateformes mobiles disponibles.

Dans cette thèse, nous avons cherché à réétudier les architectures des systèmes dédiés au *mobile crowdsensing* pour adresser les limitations liées au développement, au déploiement et à l'exécution de campagnes de collecte de données. Les différentes contributions proposées sont articulées autour APISENSE®, la plate-forme résultante des travaux de cette thèse. Plus particulièrement, APISENSE® propose un environnement distribué favorisant le développement et le déploiement rapides de campagnes de collecte, tout en prenant en considération des contraintes telles que la protection de la vie privée des utilisateurs ou encore les ressources énergétiques limitées de leurs terminaux. Pour atteindre ces objectifs, APISENSE® repose sur le modèle de composant SCA et sur l'ingénierie des lignes de produits logiciels, offrant ainsi un environnement modulaire et facilement configurable pour supporter une grande diversité de campagnes de collecte. Un modèle de programmation de haut niveau a également été proposé permettant de s'affranchir de toute la complexité liée aux développements d'applications de collecte mobiles.

APISENSE® a été utilisé pour réaliser une campagne de collecte de données déployée auprès d'une centaine d'utilisateurs au sein d'une étude sociologique, et évalué à travers des expériences qui démontrent la validité, l'efficacité et le passage à échelle de notre solution.

ABSTRACT

Mobile crowdsensing is a new form of data collection that takes advantage of millions smart devices already deployed throughout the world to collect massively environmental or behavioral data from a population. Recently, this type of data collection has attracted interest from a large number of industrials and academic players in many areas, such as the study of urban mobility, environmental monitoring, health or the study of sociocultural attitudes. However, mobile crowdsensing is in its early stages of development, and many challenges remain to be addressed to take full advantage of its potential. These challenges include privacy, limited energy resources of devices, development of reward and recruitment models to select appropriates mobile users and dealing with heterogeneity of mobile platforms available.

In this thesis, we aim to reconsider the architectural design of current mobile crowdsensing systems to provide a simple and effective way to design, deploy and manage data collection campaigns. The main contributions of this thesis are organize around APISENSE®, the resulting platform of this research. In particular, APISENSE® offers a distributed environment based on the SCA component model and software product line engineering , providing a modular and flexible environment to support a wide variety of data collection campaigns. Futhemore, APISENSE® takes into account constraints, such as protecting the privacy of users or limited energy resources devices. We also propose a high-level programming model to get rid of the complexity associated with the development of mobile data collection applications.

APISENSE® has been used to carry out a data collection campaign deployed over hundred of users in a sociological study and evaluated through experiments demonstrating the validity, effectiveness and scalability of our solution.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

1

INTRODUCTION

1.1 CONTEXTE

La dernière génération de terminaux intelligents tels que les smartphones ou les tablettes tactiles est rapidement devenue omniprésente dans notre quotidien. On recense, actuellement, plus d'un milliard de smartphones à travers le monde et ce nombre ne va cesser d'augmenter [42]. Non seulement dotés d'une grande capacité de calcul et d'une connexion Internet en continu, ces terminaux sont désormais programmables et équipés d'un grand nombre de capteurs sophistiqués permettant de prendre des images, capter des sons, mesurer la température ou la pression atmosphérique tout en donnant la position géographique. De ce fait, leurs usages permettent de générer de nombreuses traces numériques d'activités, capables d'identifier nos habitudes, nos relations sociales ainsi que l'environnement dans lequel on évolue.

De par leurs fortes présences dans notre société et leurs hautes prouesses technologiques, ces terminaux ont ouvert la voie à une nouvelle forme de collecte de données à grande échelle, récemment appelée le *Mobile Crowd Sensing* (MCS) [27]. Plus particulièrement, le MCS fait référence à l'ensemble des applications qui impliquent des citoyens ordinaires, munis de leurs terminaux intelligents, afin de collecter et de partager des données dans le but de mesurer des phénomènes d'un intérêt commun. Ces dernières années, de nombreuses recherches ont été menées pour tenter d'exploiter le potentiel du MCS dans de nombreux domaines tels que l'étude de la mobilité urbaine ou des comportements sociaux, la surveillance de l'environnement ou la santé [39, 11]. Quelques exemples d'applications novatrices dans ce secteur comprennent la collecte et le partage de données concernant la qualité de l'air [19], le trafic routier [51], la pollution sonore [47], les performances cyclistes [20], ou encore des informations relatives aux prix des produits de consommation [17].

Comme la revue de la littérature l'indique, le MCS n'en n'est qu'à ses premiers stades de développement, et de nombreux défis doivent encore être relevés pour pleinement profiter de son potentiel [39, 27, 66]. Ces défis sont particulièrement liés à l'implication d'individus rationnels, qui sont au centre du système de collecte de données. Cela demande de prendre en considération de nombreux aspects non-fonctionnels qui ne sont pas directement liés à la collecte de données difficiles en elle-même, rendant par conséquent le développement de ces systèmes encore difficile. Plus particulièrement, il est nécessaire de prendre en compte les points suivants :

ÉNERGIE : Bien que la dernière génération des terminaux intelligents continue de fournir plus de puissance de calcul, de mémoire, d'espace de stockage et de capteurs de plus en plus sophistiqués, les ressources énergétiques de ces terminaux restent limitées. Sans une gestion rigoureuse de l'énergie consommée par les applications de collecte, la batterie de ces terminaux peut s'épuiser en quelques heures, décourageant ainsi les individus à installer ces applications.

RESPECT DE LA VIE PRIVÉE : Sans un mécanisme de protection approprié, les applications mobiles peuvent se devenir de parfaits espions, en révélant potentiellement des informations privées sur leur propriétaire. Il est par exemple possible qu'elles puissent enregistrer des conversations intimes ou prendre des photos d'une scène privée, d'identifier des trajets quotidiennement empruntés par le propriétaire du terminal et d'observer ses principaux lieux de vie tels que son domicile ou son lieu de travail. Ce point représente également un frein majeur pour l'acceptation de ces applications par les individus. D'un autre coté, certaines de ces données peuvent se révéler indispensables pour l'objectif de la collecte de données.

COMPROMIS : Prendre en compte ces deux derniers points demande généralement le développement d'algorithmes sophistiqués permettant de faire un compromis entre l'énergie consommée par les applications mobiles, leur niveau d'intrusion dans la vie privée des individus et la qualité des données collectées.

DIVERSITÉ : De plus, les développeurs doivent également faire face à la diversité des systèmes d'exploitations mobiles disponibles sur le marché (*par ex.* Android, iOS). Pour déployer une application auprès d'un grand nombre d'individus, les développeurs doivent généralement étudier et développer une application pour chacun de ces systèmes, entraînant par conséquent un coût important (*c.-à-d.* temps de développement, monétaire).

PASSAGE Á L'ÉCHELLE : Finalement, ces systèmes doivent également faire face au défi de collecter une grande quantité de données reportées par des utilisateurs répartis à travers le monde. De ce fait, déployer une application auprès de plusieurs millions d'utilisateurs qui collectent et propagent continuellement des données vers un serveur serait inefficace, et demanderait des ressources importantes en terme de CPU, de mémoire et d'espace de stockage de l'infrastructure hébergeant l'application serveur pour stocker et analyser ces données. En outre, ces applications peuvent générer un lot considérable de données qui peuvent être inutiles, surtout si la collecte vise uniquement une région spécifique ou une population particulière. Dans ce contexte, il est nécessaire de mettre en place un modèle de déploiement adapté pour recruter les bons utilisateurs au bon moment, capable de collecter seulement les données utiles à la campagne de collecte.

À ce jour, beaucoup d'efforts ont principalement portés sur la réalisation de systèmes monolithiques, difficilement réutilisables dans un contexte non anticipé [6]. Ces systèmes ont été conçus complètement indépendamment les uns des autres, ne partageant aucun module logiciel commun, alors qu'ils ont à faire face à des problématiques communes. La mise en place d'une nouvelle campagne de collecte demande généralement de développer un nouveau système à partir de zéro, pouvant entraîner la négligence de certains aspects non fonctionnels, comme protéger la vie privée des individus, à cause de contrainte de temps ou une expertise limitée dans ce domaine. Ce manque d'approche réutilisable, également pointé du doigt dans la littérature [39, 27, 60, 66], entrave ainsi l'adoption du MCS par de nombreux acteurs intéressés par ce type de collecte.

Dans cette thèse, nous avons cherché à réétudier les architectures des systèmes dédiées au MCS pour adresser les limitations identifiées ci-dessus liées au développement, au déploiement et à l'exécution d'une campagne de collecte de données. Plus particulièrement, l'objectif des travaux de cette thèse est de proposer une plate-forme générique, favorisant le développement et le déploiement rapides de campagnes de collecte de données pouvant être menées dans une grande variété de domaines.

1.3 OBJECTIF DU MANUSCRIT

Comme nous l'avons décrit dans la section précédente, le développement d'applications utilisant le MCS comme source de collecte de données est une tâche complexe, nécessitant une grande expertise. Toutefois, pour ouvrir cette forme de collecte à de nombreux acteurs académiques et industriels, nous proposons dans cette thèse une plate-forme générique qui facilite le développement et le déploiement de ces applications de collecte. Plus particulièrement, nous adressons dans cette thèse les points suivants :

Simplifier le développement des applications de collecte

Le premier défi concerne le développement des applications de collecte. Actuellement, ce développement nécessite une grande expertise dans les systèmes d'exploitation des terminaux mobiles disponibles sur le marché. Pour simplifier leur développement, il est nécessaire de proposer un modèle de programmation nouveau fournissant une abstraction complète de ces systèmes d'exploitation. Dans ce contexte, les principaux défis sont de proposer un modèle : i) assez général pour supporter une grande variété d'activités de collecte qui peuvent impliquer une grande variété de capteurs, ii) permettant de minimiser l'expertise nécessaire dans les technologies mobiles afin d'accéder aux fonctionnalités offertes par les capteurs des terminaux mobiles, ainsi que leurs collectes et leurs propagations vers l'infrastructure serveur, iii) et finalement portable pour être en mesure de s'exécuter sur les différents systèmes d'exploitation. D'autre part, il est nécessaire de fournir un environnement fiable assurant la confidentialité des utilisateurs, efficace pour ne pas empêcher une utilisation normale des terminaux mobiles et faciles d'utilisation pour les utilisateurs voulant participer aux collectes de données.

Mise en œuvre d'une plate-forme générique de collecte de données

Le deuxième défi concerne l'architecture de la plate-forme responsable du déploiement des applications de collecte, de la persistance et de l'analyse des données collectées. La plupart des plate-formes proposées actuellement sont souvent monolithiques, donnant très peu ou pas de possibilités de personnalisation. Cependant, les applications de collectes peuvent être menées dans une grande variété de domaines, nécessitant diffé-

rents services de l'application serveur pour analyser et stocker les données collectées ou encore déployer ces applications. Pour prendre en compte cette diversité, la flexibilité et l'extensibilité doivent être les propriétés clés de l'architecture mise en œuvre. Cela demande par conséquent de proposer un modèle identifiant les points communs et les différentes exigences des applications de collecte qui peuvent être réalisées. Ce modèle doit s'appuyer sur les bonnes pratiques de l'ingénierie logiciel pour fournir un cadre logiciel modulaire et configurable, et être utilisé simplement par les différents acteurs voulant mener de nouvelles collectes de données pour leurs permettre d'exprimer leurs exigences. D'autre part, la plate-forme doit être capable de passer à l'échelle, c'est-à-dire d'être capable d'impliquer un grand nombre d'utilisateurs mobiles ainsi qu'un grand nombre d'acteurs voulant définir de nouvelles collectes d'un grand volume de données.

1.4 CONTRIBUTIONS

En réponse à ces défis, ces travaux ont abouti à la définition, l'implémentation et l'évaluation d'une plate-forme baptisée APISENSE®. La finalité de APISENSE® est de permettre une mise en place rapide de campagnes de collectes de données à travers des terminaux mobiles intelligents. Nous proposons ici une vue d'ensemble de la solution proposée illustrée par la figure 1.

APISENSE® distingue deux rôles évoluant dans la plate-forme. Le premier, appelé simplement *utilisateur*, peut être un acteur voulant définir et déployer de nouvelles campagnes de collecte de données à travers des terminaux mobiles. Les utilisateurs peuvent utiliser un nœud de collecte dédié (DataGathering Node), qui peut être déployé sur une infrastructure publique ou privée selon leur exigence, et se servir des services fournis par celui-ci pour définir de nouvelles tâches de collecte grâce à un langage de script dédié, déployer la tâche à travers un sous-ensemble d'individus ainsi qu'exploiter ou exposer les données collectées (*par ex.* visualisation, analyse).

Le second rôle, appelé *participant*, est un individu possédant un terminal mobile. Les participants peuvent utiliser une application mobile dédiée pour télécharger les tâches de collecte, les exécuter dans un environnement sécurisé et automatiquement propager les données collectées.

Dans APISENSE®, la mise en relation entre les nœuds de collecte et les participants est assurée par le serveur central (Central Server). Dans un sens, le serveur central peut être perçu comme un magasin dédié aux tâches de collecte. Typiquement, son rôle est

d'assurer le déploiement des tâches de collecte soumises par les nœuds de collecte, et également d'assurer une première couche d'anonymat des participants.

Les objectifs présentés dans la section précédente sont adressés avec APISENSE® de la manière suivante :

- À l'issue d'une étude approfondie des architectures présentées dans la littérature (*cf.* chapitre 2 section 2.3), nous avons proposé un modèle permettant de représenter la variabilité des systèmes responsables du développement et du déploiement de campagnes de collectes de données (*cf.* chapitre 3 section 4.4). Ce modèle est ensuite fourni aux utilisateurs leur permettant de définir des exigences selon la spécificité des campagnes qu'ils veulent mener. Une application serveur dédiée (*c.-à-d. DataGathering Node* dans la figure 1) est ensuite générée suite aux exigences définies, fournissant un ensemble de services permettant de développer et de déployer de nouvelles campagnes de collecte, d'assurer la persistance des données collectées par les dispositifs mobiles ainsi que de connecter des services additionnels pour extraire ou traiter ces données.
- Nous avons défini un modèle de programmation de haut niveau permettant de s'affranchir de toute la complexité liée aux développements mobiles. Basée sur un langage de script, cette abstraction a été conçue pour être assez générale pour supporter une grande variété d'applications de collecte, facilement accessible pour des utilisateurs ayant très peu d'expertise en programmation mobile, et facilement portable pour être exécutée dans différents environnements mobiles (*par ex.* Android, iOS, Window Mobile)(*cf.* chapitre 3 section 3.3).
- Pour assurer l'exécution des campagnes de collecte définies par notre modèle de programmation, nous proposons un environnement d'exécution dédié aux utilisateurs mobiles. Cet environnement est responsable du téléchargement, de l'exécution des applications de collectes et de la propagation des données collectées (*cf.* chapitre 3 section 3.4). Principalement, cet environnement met l'accent sur la consommation énergétique liée à l'exécution d'une campagne de collecte et la protection de la vie privée des utilisateurs.
- Nous proposons des modèles et des algorithmes dédiés à l'optimisation de l'exécution de campagnes de collecte. L'optimisation proposée réside en deux points. Pour le premier, nous proposons un modèle de déploiement de tâches de collecte contextuel, permettant d'attribuer une application de collecte à un individu en fonction de propriétés temporelles, géographiques et de capacité

de détection. Pour le second, nous proposons un algorithme permettant de coordonner l'exécution des tâches de collecte, entre les dispositifs répondant aux mêmes propriétés, afin d'équilibrer les coûts énergétiques entre les terminaux et de diminuer la redondance des données collectées(*cf.* chapitre 4 section 4.5).

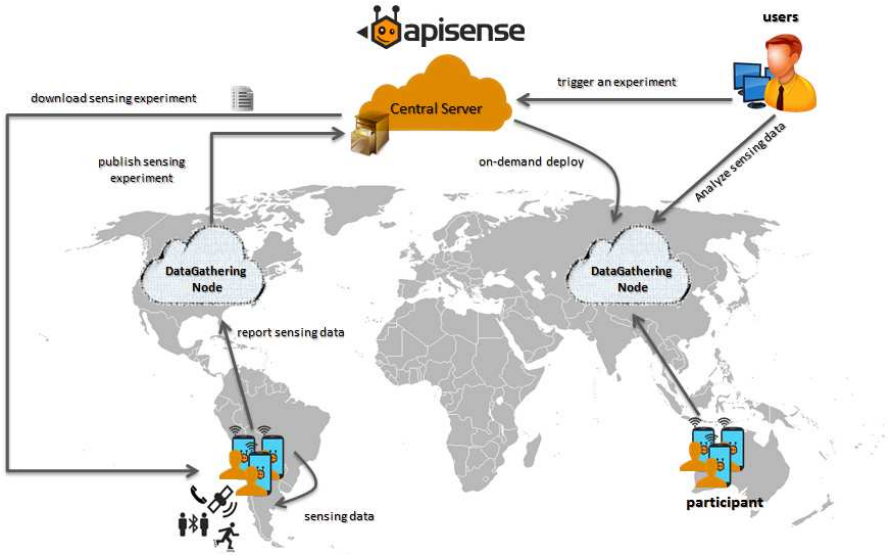

FIGURE 1 – Vue d'ensemble de APISENSE®

1.5 PLAN DU MANUSCRIT

Ce manuscrit est divisé en quatre parties. La première partie donne une vue d'ensemble de l'état de l'art dans le domaine du *Mobile crowdsensing*. La seconde partie décrit les contributions de ce manuscrit. La troisième porte sur la validation des contributions proposées. Et finalement la quatrième conclut ce manuscrit et décrit les perspectives issues des travaux de cette thèse. Plus particulièrement, la suite de ce document est organisé comme suit.

Partie 1 : État de l'art

Chapitre 1 : Systèmes de collecte de données Ce chapitre a pour objectif de donner dans un premier un temps une vison plus approfondie du *Mobile crowdsensing* ainsi que ses problématiques. Nous décrivons par la suite des travaux proches à ceux proposés dans cette thèse et nous identifions leurs limitations.

Partie 2 : Contributions

Chapitre 3 : Collecte mobile de données Ce chapitre traite de l'environnement mobile de APISENSE®. Il décrit tout d'abord le modèle de programmation proposé permettant de minimiser le coût du développement des applications de collecte de données. Par la suite, il présente l'architecture de l'environnement mobile dédié à l'exécution des applications de collecte.

Chapitre 4 : Collecte répartie de données Ce chapitre présente l'architecture et les choix d'implémentation de l'environnement serveur APISENSE® permettant de déployer des applications de collecte ainsi que d'exploiter les données collectées.

Partie 3 : Validations

Chapitre 5 : Pratiques culturelles et usages de l'informatique connectée Ce chapitre présente une campagne de collecte déployée auprès d'une centaine d'utilisateurs, réalisée au sein d'une étude sociologique nommée PRATIC (Pratiques Culturelles et Usages de l'Informatique Connectée).

Chapitre 6 : Performance et efficacité de APISENSE® présente une validation quantitative sur les performances de APISENSE®

Partie 4 : Conclusion

Chapitre 7 : Conclusion Finalement, ce chapitre conclut ce manuscrit et expose les perspectives des travaux présentés.

1.6 PUBLICATIONS

Les travaux de cette thèse ont été réalisés au sein de l'équipe SPIRALS commune à Inria et à l'Université Lille 1 au sein de l'UMR LIFL (Laboratoire d'Informatique Fondamentale de Lille). Ils ont donné lieu aux publications scientifiques suivantes.

Conférence International

- *Dynamic Deployment of Sensing Experiments in the Wild Using Smartphones*. Nicolas Haderer, Romain Rouvoy and Lionel Seinturier. In 13th International IFIP

Conference on Distributed Applications and Interoperable Systems (DAIS), pages 43-56.

- *A Federated Multi-Cloud PaaS Infrastructure*. Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, Lionel Seinturier. In 5th IEEE International Conference on Cloud Computing (2012), pages 392-399.

Chapitre de livre

- *A Cloud-based Infrastructure for Crowdsourcing Data from Mobile Devices*. Nicolas Haderer, Fawaz Paraiso, Christophe Ribeiro, Philippe Merle, Romain Rouvoy, Lionel Seinturier Wenjun Wu. Cloud-based Software Crowdsourcing, Springer, 2014 (To appear)

Workshop

- *A preliminary investigation of user incentives to leverage crowdsensing activities*. Nicolas Haderer, Romain Rouvoy and Lionel Seinturier. 2nd International IEEE PerCom Workshop on Hot Topics in Pervasive Computing (PerHot) (2013), pp. 199-204.
- *Towards Multi-Cloud Configurations Using Feature Models and Ontologies*. Clément Quinton, Nicolas Haderer, Romain Rouvoy and Laurence Duchien. In Proceedings of the 1st International Workshop on Multi-Cloud Applications and Federated Clouds, Multi-Cloud'13. Prague, Czech Republic, 22 April 2013, pp. 21-26.

Vulgarisation scientifique

- *APISENSE : Crowd-Sensing Made Easy*. Nicolas Haderer, Romain Rouvoy, Christophe Ribeiro, Lionel Seinturier. ERCIM News, ERCIM, 2013, Special theme : Mobile Computing, 93, pp. 28-29.
- *Le capteur, c'est vous !* Nicolas Haderer, Christophe Ribeiro, Romain Rouvoy, Simon Charneau, Vassili Rivron, Alan Ouakrat, Sonia Ben Mokhtar, Lionel Seinturier L'Usine Nouvelle, L'Usine Nouvelle, 2013, 3353, pp. 74-75
- *Campagne de collecte de données et vie privée*. Nicolas Haderer, Miguel Nuñez Del Prado Cortez, Romain Rouvoy, Marc-Olivier Killijian and Matthieu Roy. 3ème Journées du GDR CNRS GPL (2012), pp. 253-254.

Rapport de recherche

- *AntDroid : A distributed platform for mobile sensing*. Nicolas Haderer, Romain Rouvoy, Lionel Seinturier. [Research Report], 2012, pp. 27. RR-7885

Première partie

État de l'art

2

SYSTÈMES DE COLLECTE DE DONNÉES

Sommaire

2.1 MOBILE CROWDSENSING

2.1.1 *Qu'est ce que le Mobile crowdsensing ?*

En 2010, Ganti et al. définissent le *Mobile crowdsensing* comme *la participation d'un groupe d'individus, disposant de terminaux mobiles intelligents qui, collectivement, partagent des informations pour la mesure ou la cartographie de phénomènes d'un intérêt commun"*. Typiquement, le MCS offre de nombreux avantages par rapport aux méthodes classiques de collecte de données telles que les réseaux de capteurs (WSNs), qui peuvent impliquer des coûts importants liés à l'installation d'un grand nombre de capteurs statiques et de leurs maintenances. En effet, des millions de dispositifs mobiles sont déjà déployés dans la nature, portés par des utilisateurs dans leur vie quotidienne. Avec la généralisation des magasins d'applications (*par ex.* App Store, Google Play), il est désormais possible pour des petites organisations comme des équipes de recherche ou des petites entreprises de rapidement délivrer une application mobile auprès de cette masse d'utilisateurs. Par exemple, au lieu d'installer un ensemble de caméras le long des routes, il est possible de collecter des données du trafic routier et de détecter les embouteillages en utilisant le GPS des dispositifs mobiles des conducteurs.

Un autre avantage de l'utilisation de ces dispositifs est le nombre de capteurs multimodaux inclus dans ces appareils, permettant une collecte d'information contextuelle de haut niveau. En effet, le GPS peut fournir la position d'un utilisateur. Le microphone, quand il n'est pas utilisé pour les communications téléphoniques, peut être utilisé comme capteur acoustique. L'accéléromètre peut être, quant à lui, utilisé pour identifier les modes de transport des utilisateurs, leur activité journalière (*par ex.* temps passé en position assise, à dormir, à marcher, à courir) mais aussi détecter des dégâts importants sur une route. De nouveaux capteurs peuvent également facilement être ajoutés à ceux initialement préinstallés, en utilisant des connexions sans fils comme le Bluetooth pour communiquer entre eux. Quelques exemples sont les bracelets connectés qui peuvent mesurer la fréquence cardiaque [1] d'un individu, ou les capteurs qui mesure la qualité de l'air [19]. Pour finir, le dernier avantage est de pouvoir inclure les utilisateurs dans le processus de collecte, bénéficiant ainsi de l'intelligence humaine pour collecter des données sémantiquement complexes à identifier à partir de simples capteurs. Par exemple, un utilisateur peut facilement identifier la dégradation d'installations

1. http://pulseon.fi/

publiques, prendre une photo de ces dégâts et l'envoyer à la collectivité territoriale concernée [35].

Grâce à ces avantages, le MCS a suscité l'intérêt d'un grand nombre d'acteurs industriels et académiques dans des domaines tels que l'étude de la mobilité urbaine ou des comportements sociaux, la surveillance de l'environnement et la santé [39, 11]. Cependant, l'utilisation des dispositifs mobiles possédés par des individus introduit également de nouvelles problématiques, comme la préservation de la vie privée des utilisateurs, la grande consommation énergétique des applications de collecte ou alors comment inciter les utilisateurs à installer et exécuter ces applications. Dans la suite de cette section, nous présentons tout d'abord une vue d'ensemble de ces applications avant de décrire avec plus de précision ces problématiques.

2.1.2 *Classification des applications*

Dans la littérature, il existe un grand nombre d'applications de collecte de données, mettant en évidence l'utilisation du MCS dans une grande variété de domaines. Globalement, ces applications peuvent être classées selon deux critères : le sujet à observer et le mode d'acquisition des données. Le classement selon le sujet à observer se scinde, lui encore, en deux dimensions : les applications de collecte *personnelle* et les applications de collecte *communautaire*. Les applications de collecte personnelle visent à observer et analyser le comportement d'un individu (e.g., activité physique, mode de transport), tandis que les applications de collecte communautaire visent à observer des phénomènes (à plus grande échelle) sur l'environnement (*par ex.* qualité réseaux, pollution atmosphérique ou sonore) ou sur une infrastructure publique (e.g., trafic routier, place de parking disponible, dégâts de matériel public). Le classement selon le mode d'acquisition de données peut également se scinder en deux modalités appelées : collecte participative et collecte opportuniste. Dans la collecte participative, l'utilisateur est directement impliqué dans la prise de décision de la collecte (i.e., en décidant quand, comment et quelles données doivent être collectées). Au contraire, dans la collecte opportuniste, elle est complètement automatisée par l'application, sans nécessiter une intervention de l'utilisateur.

Collecte personnelle ou communautaire

Dans les systèmes de collecte personnelle, l' objectif est d'observer des phénomènes liés à un individu. Dans ce type de système, toutes les données collectées sont généralement couplées avec un identifiant unique, permettant de suivre l'évolution d'un utilisateur spécifique à travers le temps. Elles sont généralement rendues à l'utilisateur sous forme d'un rapport détaillé sur son activité, partagées à travers un réseau social ou alors agrégées à travers plusieurs utilisateurs afin d'identifier divers comportements. Ce type de collecte est généralement utilisé dans les domaines de la santé, du sport ou de l'environnement.

Dans le domaine de l'environnement par exemple, PEIR [51](*Personal Environmental Impact Report*) est un système permettant aux utilisateurs de se servir de leurs dispositifs mobiles afin de déterminer leur exposition aux polluants présents dans leur environnement. *PEIR* collecte continuellement les données GPS vers un serveur pour y effectuer une série de traitement consistant à segmenter les différents déplacements des utilisateurs, identifier les modes de transports (*par ex.* bus, voiture) et calculer la quantité de gaz carbonique émise lors des déplacements. Pour le calcul de l'émission du gaz carbonique, PEIR couple les informations des déplacements avec plusieurs bases de données : celles des principales entreprises polluantes, et celles de la météo et du trafic routier. Sur un site Internet spécifique, les utilisateurs peuvent ensuite consulter les rapports de leurs déplacements, les aidants à modifier leurs comportements et protéger leur santé.

Dans le domaine du sport, BikeNet [20] fournit aux cyclistes une application leur permettant de mesurer leurs expériences sportives. BikeNet combine les données du GPS et de l'accéléromètre pour calculer la vitesse, les calories perdues ainsi que la qualité des routes empruntées. Les données collectées peuvent être visualisées par les cyclistes eux-mêmes, ou être agrégées avec d'autres utilisateurs afin de construire une carte complète des pistes cyclables.

Les systèmes de collecte personnelle sont également beaucoup utilisés dans le cadre d'expériences scientifiques. Par exemple, EmotionSense [56] est un système dédié aux études de psychologie sociale. Il tente d'établir une relation entre le bien-être d'un utilisateur et ses interactions sociales ou ses activités. Pour identifier ces informations, EmotionSense analyse continuellement trois types d'informations. Premièrement, les conversations téléphoniques des utilisateurs pour mesurer leurs émotions durant leurs appels. Deuxièmement, les périphériques Bluetooth voisins pour identifier les

personnes se trouvant régulièrement à proximité. Et finalement l'accéléromètre pour identifier l'activité régulière de l'utilisateur.

Dans un autre contexte, HealthAware [28] est une application visant à observer l'obésité des utilisateurs. Avec cette application, les utilisateurs peuvent prendre en photo leur nourriture, ce qui permet au système de leur fournir en retour des informations sanitaires complémentaires sur cette nourriture. HealthAware collecte également les données de l'accéléromètre pour mesurer l'activité physique quotidienne des utilisateurs. Le système fournit en retour un rapport complet sur l'activité effectuée, et rappelle aux utilisateurs l'importance de garder une activité physique quotidienne pour être en bonne santé.

Les systèmes de collecte communautaire visent, quant à eux, à observer des phénomènes à plus grande échelle, sur l'environnement ou sur des infrastructures publiques afin d'améliorer la vie d'une communauté (*par ex.* toutes les personnes habitant dans une ville, les étudiants d'une université). Quelques scénarios types de ce genre de collecte comprennent la mesure de la pollution atmosphérique en milieu urbain, ou l'observation des réseaux routiers afin de détecter les embouteillages ou des dégâts importants sur la chaussée.

CommonSense [19] est un exemple d'application dédié à l'observation de la pollution atmosphérique. L'application mobile de CommonSense combine les données GPS avec des données fournies par un capteur externe mesurant la qualité de l'air ambiant. La communication entre le mobile et le capteur externe est assurée par une connexion Bluetooth. Les capteurs externes ainsi que l'application sont déployés auprès de la population, qui collectivement propage leurs positions ainsi que la qualité de l'air vers un serveur. Les données sont ensuite agrégées et peuvent être visualisées sur un site Internet spécifique. Similairement, NoiseTube [47] utilise directement les microphones des dispositifs mobiles pour mesurer la pollution sonore en milieu urbain.

Un autre type de collecte communautaire implique la mesure de phénomènes en relation avec des infrastructures publiques. Par exemple, Nericell [50] est un projet qui a pour objectif de mesurer différents phénomènes issus du trafic routier. Ce projet analyse continuellement la position d'un utilisateur qui, couplée avec les informations issues de l'accéléromètre, permet de détecter la dégradation d'une route ou des freinages d'urgences et, couplée avec le microphone, permet de détecter un fort trafic à partir du bruit ambiant.

Dans un autre contexte, LiveCompare [17] est une application qui permet de comparer différents prix d'un même article se trouvant dans des magasins situés à proximité

d'un utilisateur. Avec cette application, les utilisateurs peuvent se servir de leur dispositif pour prendre en photo le code-barres d'un article. Le code est ensuite décrypté, et envoyé vers un serveur avec la position de l'utilisateur et la photo. En échange, LiveCompare fournit aux utilisateurs les différents prix disponibles pour ce même produit, en fonction des magasins se trouvant à proximité.

Un autre exemple d'application est CreekWatcher [2]. Cette application surveille les niveaux et la pollution des eaux, à l'aide des rapports des utilisateurs, comme des photos prises à divers endroits au bord des étendues d'eau, ou des messages texte sur le quantité de déchets. Ces informations peuvent ensuite être exploitées par des commissions de contrôle des eaux.

Collecte participative ou opportuniste

Indépendant du sujet à observer, un facteur déterminant du succès d'une application est le niveau d'implication des utilisateurs dans le processus de collecte [38]. Typiquement, le processus de collecte peut prendre deux formes appelées collecte *opportuniste* [39] et *participative* [7].

Dans celle dite opportuniste, la collecte de données est complètement automatisée par l'application mobile (*par ex.* "collecter la position toutes les cinq minutes"). Cette approche a le principal avantage de minimiser l'intervention humaine dans le processus de collecte. Cela est particulièrement utile dans le cadre d'une collecte communautaire, permettant d'assurer la propagation de données régulièrement sur le serveur (sans que l'utilisateur soit contraint à accomplir des tâches quotidiennes sur son dispositif). Cependant, ces applications sont plus difficiles à développer, et consomment également beaucoup de ressources énergétiques. En effet, la difficulté principale pour développer ce type d'application est de déterminer dans quel contexte se trouve le dispositif mobile, afin d'assurer une certaine qualité des données collectées. Par exemple dans l'application *NoiseTube* [47], qui a pour objectif d'observer la pollution sonore, il est nécessaire de mesurer le niveau sonore uniquement quand le dispositif est hors de la poche ou du sac de l'utilisateur. Dans ce cas, cette issue peut être résolue en utilisant le capteur de lumière, mais par conséquent, nécessite la consommation de ressources plus conséquentes.

2. http://www.ibm.com/smarterplanet/us/en/water_management/article/creek_watch.html

Au contraire, la collecte dite participative nécessite une plus grande implication de la part des utilisateurs. Par exemple, dans l'application *LiveCompare* [17], les utilisateurs doivent manuellement prendre une photo du produit qu'ils veulent comparer. Le principal avantage de cette approche est qu'elle permet de bénéficier de l'intelligence humaine pour réaliser des opérations plus complexes. Si nous reprenons l'exemple de application *NoiseTube* [47], l'approche participative peut permettre de résoudre facilement le problème du contexte du dispositif, en demandant aux utilisateurs de prendre une mesure du niveau sonore manuellement lorsque leur dispositif est hors de leur poche. Cependant, l'inconvénient de cette approche est que la qualité des données, ainsi que la fréquence à laquelle elles sont collectées, dépend fortement de l'enthousiasme des utilisateurs pour l'application.

2.1.3 *Discussion*

Dans cette section, nous avons présenté une vue d'ensemble du MCS, ses principales caractéristiques ainsi que les domaines d'applications utilisant le MCS comme source d'approvisionnement de données. Cependant, jusqu'à ce jour, les applications développées ont principalement été élaborées dans un contexte applicatif spécifique, et difficilement réutilisable. En effet, ces applications ont été conçues complètement indépendamment des unes des autres, ne partageant aucun module logiciel commun, alors qu'elles ont à faire face à des problématiques communes. Ce manque de solutions réutilisables, largement pointé du doigt dans la littérature ces dernières années [39, 27, 60, 66], rend non seulement le développement d'une application spécifique difficile, mais il y a aussi la problématique du recrutement des participants qui doit être repris de zéro pour chaque application.

Avec la popularité croissante du MCS dans de nombreuses communautés scientifiques, fournir une plate-forme générique permettant un rapide développement et un déploiement d'une large variété de campagnes de collecte de données devient une nécessité [27]. C'est donc dans cette lignée que s'inscrivent les travaux de cette thèse. Cependant, réaliser une telle plate-forme comporte de nombreux défis, que nous présentons dans la section suivante.

2.2 LES CHALLENGES CLÉS DU *mobile crowdsensing*

Dans cette section, nous identifions les principaux défis à relever afin proposer une plateforme générique dédiée aux développements et aux déploiements de campagne de collecte de données. Principalement, nous identifions cinq défis : *Sécurité et vie privée*, *Gestion des ressources*, *Hétérogénéité des équipements et des OS*, *Diffusion et passage à l'échelle de applications*, *Implication et incitation des usagers*. Pour chacun des défis, nous décrivons brièvement la problématique ainsi que les différentes solutions envisagées dans l'état de l'art.

2.2.1 *Sécurité et Vie privée*

Respecter la vie privée des utilisateurs est peut-être la responsabilité la plus fondamentale d'une plate-forme de collecte. Les utilisateurs sont naturellement sensibles aux données collectées sur leurs dispositifs, spécialement si ces informations comprennent leurs localisations, des images sensibles ou alors des communications audios. En effet, de nombreux travaux [36] ont montré que de nombreuses données, même si elles peuvent paraître anodines prises individuellement, peuvent révéler de nombreuses informations sensibles sur les habitudes ou les relations sociales d'un individu lorsqu'elles sont agrégées dans la durée. Par exemple, les données de géolocalisation collectées avec le GPS, couplées avec des données temporelles peuvent être utilisées pour inférer les trajets quotidiens empruntés par un utilisateur, ou encore ses principaux lieux de vies tels que son domicile ou son lieu de travail [26].

Afin de protéger la vie privée des utilisateurs, de nombreuses solutions ont été envisagées dans la littérature. Christin et al. [11] donne une bonne vue d'ensemble des techniques utilisées selon le contexte de la collecte, qu'elle soit personnelle ou communautaire. Typiquement, ces techniques peuvent consister à utiliser un alias pour assurer les communications entre le serveur et le dispositif mobile, à perturber volontairement les données collectées avant de les propager vers le serveur (en modifiant par exemple les coordonnées GPS), à agréger les données entre n utilisateurs partageant des propriétés communes (*par ex.* n participant situés dans le même quartier), ou alors à échanger alternativement les traces de mobilités entre plusieurs utilisateurs.

Cependant, bien que ces approches permettent d'assurer une certaine confidentialité des utilisateurs, elles impliquent également une dégradation des données qui sont collectées. Dans ce cas, il est nécessaire de faire un compromis entre qualité et confidentialité. Ce compromis peut être fait en effectuant un maximum de traitement directement dans le dispositif mobile. Par exemple, NERICELL [50] adopte cette approche en analysant toutes les données issues de l'accéléromètre et du microphone localement, et propage uniquement des informations de haut niveau sur le serveur (*par ex.* fort trafic).

Une autre solution envisagée par Christin et al. [11] est de laisser la possibilité aux utilisateurs de définir leurs exigences en matière de vie privée. En effet, la notion de vie privée peut énormément varier d'un individu à un autre. Par exemple, certain utilisateur peuvent être retissant à partager leur position contraire à d'autre. Dans ce contexte, il est nécessaire de fournir aux utilisateurs des moyens leur permettant de spécifier les données qu'ils veulent partager, dans quelles circonstances (*par ex.* temps, position), et surtout dans quel but.

2.2.2 *Gestion des ressources*

Alors que les smartphones continuent de fournir plus de puissance de calcul, de mémoire, d'espace de stockage et de capteurs de plus en plus sophistiqués, les ressources énergétiques de ces dispositifs restent limitées. En effet, certaines applications déployées [49] montrent que la durée de vie de la batterie d'un dispositif, peut être réduite de 20 heures à 6 heures si certaines ressources sont exploitées activement. Par exemple les ressources du CPU pour traiter un grand volume de données (*par ex.* traitement de données audio), les ressources de certains capteurs comme le GPS, ou encore des ressources de communications (*par ex.* 3G, WIFI) pour partager en temps réel les données collectées.

Un aspect intéressant pour réduire la consommation énergétique de ces applications est la multimodalité de certains capteurs. En effet, cela peut permettre de faire un compromis entre la qualité des données, en terme de fréquence d'échantillonnage et de précision, et le coût énergétique lié à leurs acquisitions. L'exemple le plus courant est l'acquisition de la position d'un utilisateur, qui peut être obtenue soit par le GPS, impliquant une grande coût énergétique, mais fournissant une grande précision (5-20 mètres), soit par WiFi ou par la triangularisation GSM, qui sont moins précis

(20-1000 mètres), mais ont une taxe énergétique moins importante. À partir de ce constat, de nombreux travaux ont été proposés dans la littérature, essayant d'alterner l'utilisation de ces différents capteurs pour faire le plus efficacement ce compromis. Par exemple, *EnLoc* [14] adapte la fréquence d'échantillonnage et les capteurs haute et basse qualité en fonction de la batterie de l'utilisateur. *Matador* [8] propose également un algorithme adaptatif, mais pour déterminer si l'utilisateur se trouve dans une zone précise. *SenseLoc* [34] active le GPS uniquement si l'utilisateur se déplace en utilisant l'accéléromètre pour identifier son activité.

Cependant, un inconvénient de ces approches est qu'elles ont été conçues pour un contexte applicatif spécifique, et qu'elles ne sont donc pas forcement adaptées pour un autre. De plus, ces approches restent limitées lorsque de multiples applications coexistent dans un même dispositif. Par exemple, les applications dédiées à l'observation de la pollution sonore et du trafic routier nécessitent toutes les deux des données de géolocalisation. Dans ce contexte, ces applications effectuent leurs propres échantillonnages des données GPS, pouvant entraîner une surconsommation énergétique du dispositif mobile. Cela limite ainsi les utilisateurs à participer à un nombre limité d'applications.

2.2.3 *Hétérogénéité des équipements et des OS*

Un autre aspect à prendre en considération est l'hétérogénéité des systèmes d'exploitation et des équipements disponibles. En effet, selon *Gartner Inc.* [3], les ventes de *smartphones* en 2013 ont totalisé plus de 1,8 milliards d'unité, réparties principalement entre trois systèmes d'exploitation (OS), Android [4] avec 78,4% de part de marché, iOS [5] avec 15,6 % et Windows Phone [6] avec 3,2%. Bien que ce marché en pleine effervescence — soit une augmentation 3.5% depuis 2012 —, confirme le potentiel du *Mobile crowdsensing* pour la collecte de données à grande échelle, cela induit aussi un long processus de développement pour prendre en considération tous ces systèmes d'exploitation.

Pour le développement d'une application sur un OS spécifique, un ensemble d'outils et d'APIs sont fournis aux développeurs. Ces APIs sont implémentées généralement dans des langages de programmation différents, par exemple Java pour Android et

3. Gartner Inc. http://www.gartner.com/newsroom/id/2665715
4. Android : http://www.android.com
5. iOS : http://www.apple.com/fr/ios
6. Windows Phone : http://www.windowsphone.com

Objective-C ou Apple Swift[7] pour iOS. Ceci implique qu'une application développée pour l'un de ces OS est incompatible avec les autres. Par conséquent, cela demande aux développeurs d'étudier et de développer une application spécifique pour chaque OS disponible, afin de pouvoir impliquer un maximum d'utilisateurs dans le système de collecte. Cette diversité a été soulignée par l'expérience menée par Balan et coll. [4], qui ont mis plus six mois pour le développement et le déploiement d'une expérience de collecte à travers 15,000 taxis à Singapour.

Pour faire face à cette hétérogénéité, également identifiée par [60], il est nécessaire de concevoir les applications de collecte dans un langage de haut niveau. Dans ce contexte, le défi consiste à proposer une abstraction assez générale pour supporter le développement d'applications variées (opportuniste et participative), qui peut également impliquer une grande variété de capteurs, portables pour supporter la diversité des plate-formes mobiles (*c.-à-d.* Android, iOS, Windows Mobile), et finalement accessibles pour minimiser l'expertise nécessaire dans les technologies mobiles.

2.2.4 *Diffusion et passage à l'échelle des applications*

La diffusion des applications est un aspect crucial pour assurer le passage à l'échelle d'une plate-forme de collecte.

Généralement, la diffusion des applications mobiles est assurée par les magasins traditionnels proposés par les plate-formes mobiles (*par ex.* Google Play pour Android, Apple Store pour iOS). Cependant, une fois disponible dans un magasin, l'application peut être potentiellement téléchargée par des millions d'utilisateurs à travers le monde. Ces utilisateurs peuvent alors générer un lot considérable de données qui peuvent être inutiles, surtout si la collecte vise uniquement une région spécifique ou une population particulière. De plus, la période de prolifération des mises à jours de ces applications vers les utilisateurs peut prendre plusieurs heures jusqu'à plusieurs jours selon le magasin utilisé (*par ex.* quelques heures pour Google Play jusqu'à plusieurs heures pour Apple Store). Dans le cadre du MCS, cette période est beaucoup trop longue, spécialement si l'objectif est de collecter des données lors d'un événement ne durant que quelque jours, comme un festival ou une fête nationale. En effet, lors de ces manifestations, après avoir récolté les premières données, il peut s'avérer nécessaire de

7. https://developer.apple.com/swift

mettre rapidement à jour l'application afin d'améliorer la qualité de la collecte ou de remédier à des défauts de programmation.

Dans ce cadre, il est nécessaire de développer de nouveaux mécanismes pour déployer ces applications de collecte [39]. Ces mécanismes devront prendre en compte plusieurs considérations. La première est de maîtriser le déploiement de ces applications vers les participants les plus à même à collecter les données désirées, en limitant le nombre d'utilisateurs impliqués dans la collecte, ou en ne visant qu'une population particulière (*par ex.* région géographique, tranche d'âge), tout en assurant leurs confidentialités.

La deuxième considération est d'assurer un rapide déploiement de leurs mises à jour pour adresser les exigences évolutives liées aux premières données collectées, ou tout simplement pour corriger des erreurs de programmation.

La dernière, aussi envisagée par [27, 60], est de permettre de coordonner l'exécution des campagnes à travers plusieurs dispositifs, notamment dans les campagnes de collecte communautaire. En effet, généralement ces applications collectent périodiquement (toutes les x secondes) des données sans prendre en considération si un autre dispositif placé à proximité exécute la même application. Cela peut ainsi entraîner une forte duplication de données collectées, en particulier en milieu urbain où il peut y avoir une forte densité d'utilisateurs. Dans ce contexte, effectuer une coordination entre ces dispositifs pourrait permettre de diminuer la quantité de données redondantes sur le serveur, et également équilibrer les charges énergétiques entre les dispositifs, réduisant ainsi l'énergie globale consommée lors de l'exécution de la campagne.

2.2.5 *Implication et incitation des usagers*

Inévitablement, sans une participation adéquate des utilisateurs, il peut être très difficile d'obtenir la masse de données critique pour l'application. Cependant, comme nous l'avons mentionné dans les sous-sections précédentes, en utilisant une application de collecte, les utilisateurs consomment de nombreuses ressources de leurs dispositifs, et peuvent potentiellement s'exposer à la divulgation d'informations confidentielles. Dans ce contexte, un utilisateur peut ne pas être intéressé à participer à l'application, à moins de recevoir une récompense adaptée.

Dans ce contexte, de nombreux travaux ont porté sur la réalisation d'un modèle économique permettant d'attribuer des récompenses financières aux utilisateurs en fonction de leur participation. Ces modèles peuvent être statiques [57], en attribuant

un prix fixe aux données partagées par les utilisateurs, ou dynamiques [40] en faisant varié le prix des données en fonction de la couverture des premières données partagée, pour inciter par exemple les utilisateurs à se déplacer dans des zones peu couvertes.

Cependant, bien que ces modèles puissent favoriser la participation des utilisateurs et améliorer la qualité des données obtenues, collecter des données à grande échelle peut nécessiter un budget considérable, qui n'est pas forcément accessible par exemple pour des équipes de recherche. Dans ce contexte, de nombreux modèles alternatifs, non monétaires, ont été explorés dans la littérature ces dernières années. Ces sources de motivation peuvent par exemple s'inscrire dans le cadre d'une démarche citoyenne ou scientifique en aidant à diminuer l'émission de gaz carbonique dans la nature par exemple [51]. Ces sources peuvent également prendre la forme d'un jeu, en simulant une compétition entre les utilisateurs, en partageant ses performances sur les réseaux sociaux [49], ou en attribuant des récompenses virtuelles. Une dernière source de motivation est d'obtenir un bénéfice direct en partageant des données, par exemple dans l'application LiveCompare [17], les utilisateurs peuvent bénéficier des prix scannés par les autres utilisateurs.

2.2.6 *Discussion*

Le principal objectif d'une plate-forme générique de collecte de données est de permettre la fédération d'une large communauté d'utilisateurs, avec des niveaux d'expertises et des exigences différentes. En effet, les campagnes de collecte de données peuvent être de différents types, qui peuvent être de type communautaire ou personnel, et peuvent également impliquer différents niveaux de participation de la part des utilisateurs (*c.-à-d.* collecte opportuniste et participative). Comme nous en avons discuté tout au long de cette section, la mise en place d'une campagne demande de faire de nombreux compromis. Que ce soit sur la qualité des données, leur quantité, la préservation de la vie privée des utilisateurs ou encore la consommation énergétique des applications de collecte. Nous avons également vu que dans la littérature, de nombreux modèles ou algorithmes ont été proposés pour adresser ces points. Cependant, aucun de ces modèles ne représente une solution idéale, et ne peut être appliqué seulement dans un contexte applicatif spécifique (*par ex.* population visée, type de de données collectées).

Dans ce contexte, nous pensons que la flexibilité et l'extensibilité doivent être les propriétés clés de l'architecture logicielle d'une plate-forme générique. La flexibilité fait

référence à la capacité de la plate-forme à s'adapter à la diversité des campagnes de collecte qui peuvent être menées. Cette diversité peut faire référence à différents types de données qui peuvent être collectées (*c.-à-d.* impliquant de nombreux capteurs), aux technologies utilisées pour les sauvegarder (*c.-à-d.* base de données), et les structurer (*c.-à-d.* méthode d'indexation) et les analyser. Quant à l'extensibilité, elle fait référence à la capacité de la plate-forme pour intégrer de nouvelles contributions, afin de faire face aux exigences des utilisateurs non anticipées. Dans ce cadre, la plate-forme pourrait permettre également à des scientifiques travaillant sur une problématique spécifique du MCS (*par ex.* vie privée, modèle de récompense, etc.), de se focaliser sur leur domaine d'expertise sans avoir à redévelopper un système complet et donc participer à l'amélioration de la plate-forme.

Dans la section suivante, nous faisons une étude comparative des différentes plate-formes ayant reçu beaucoup d'attention dans la littérature ces dernières années, et ayant ce même objectif.

2.3 TRAVAUX CONNEXES

Comme nous l'avons vue dans la section précédente, le développement d'application de collecte de données demande de faire face à de nombreux défis. Dans cette section, nous décrivons un certain nombre de travaux existants, proposant des plate-formes ayant pour objectif de simplifier leurs développements et leurs déploiements. Pour chaque travail présenté, nous décrivons brièvement l'architecture générale adoptée, et nous les confrontons par rapport aux défis définis dans la section précédente.

2.3.1 *Funf Open Sensing Framework*

FUNF [1] est une plate-forme open source dédié aux développements d'applications de collecte de données mobiles basées sur le système d'exploitation Android. Initialement, FUNF a été développé par le MIT MEDIA LAB, afin de proposer aux scientifiques un outil plus générique pour les aider à facilement concevoir leurs applications de collecte.

Le concept de base proposé par FUNF est appelé `Probe`. Typiquement, un `Probe` est un module logiciel (*c.-à-d.* portion de code Android) assurant la capture de données d'un capteur physique ou logique (*par ex.* GPS, température, contact). Pour le développement

des applications, Funf met à disposition un service appelé FunfInABox, accessible via une interface web. L'interface propose un formulaire qui permet aux scientifiques de sélectionner et configurer un ensemble de `Probe` à inclure dans l'application mobile (*cf.* figure 2). Le formulaire validé, FunfInABox procède alors à la génération de l'application et de la mettre à disposition des scientifiques à travers un compte Dropbox, un service de stockage et de partage de fichier en ligne. L'application peut également être configurée pour propager périodiquement les données collectées vers le même compte Dropbox, ou sur un serveur HTTP. Pour des raisons de sécurité, toutes les données collectées sont encryptées, et toutes les données sensibles telles que les contacts ou les SMS sont automatiquement hachés. Funf fournit également un ensemble de scripts qui peuvent être utilisés par les scientifiques pour décrypter les données, les insérer dans une base de données relationnelle et visualiser les données.

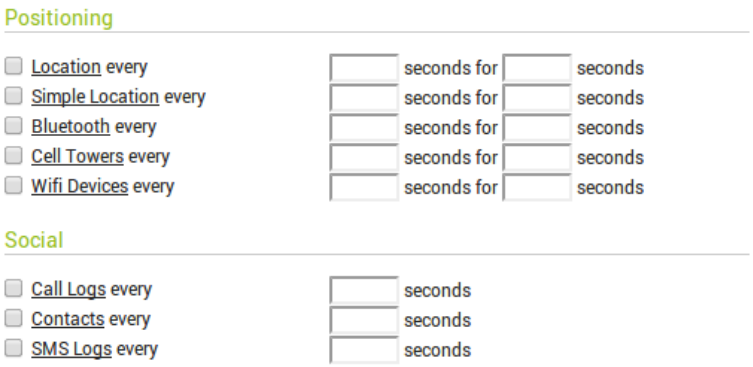

Figure 2 – Développement d'une application mobile avec FunfInABox

Funf se diffère principalement par sa simplicité. En effet, l'utilisation d'un formulaire web permet de rapidement développer une nouvelle application sans nécessiter toutes sortes d'expertise en programmation. Cependant, les applications qui peuvent être développées restent très basiques, limitées à la collecte périodique de données brutes de capteurs, et ne supportent pas les interactions avec les utilisateurs. De plus, les applications générées ne proposent pas aux utilisateurs de mécanismes leur permettant de contrôler les données qui sont collectées sur leurs dispositifs ni de retour sur celles-ci. D'autres parts, Funf ne fournissent pas de service spécifique pour aider les utilisateurs à déployer leurs applications vers les dispositifs mobiles. Ils sont alors contraints d'utiliser les magasins d'applications, devant faire face aux problématiques discutées précédemment(*cf.* section 2.2.4).

2.3.2 *MyExperience : A System for In situ Tracing and Capturing of User Feedback on Mobile Phones*

Jon Froehlich et al. [25] proposent une approche plus flexible pour développer des applications de collecte avec MyExperience.

```
<myexperience>
<!--Activate phone call & cell related sensors-->
<sensor name="PhoneCall" type="PhoneCallSensor"/>
<sensor name="CellStrength"
        type="CellSignalStrengthSensor"/>

<!--Create the phone call completed trigger-->
<trigger name="PhoneCallCompletedTrigger">
 <script>
 phoneCall = GetSensor("PhoneCall");
 if(phoneCall.State=="Completed")
   RunAction("PhoneQualitySurvey");
 </script>
</trigger>

<!--A survey action to ask about call quality-->
<action name="PhoneQualitySurvey" type="Survey">
 <prop name="TimeOutInterval">00:30</prop>
 <prop name="EntryQuestionId">CallQuality</prop>
</action>

<!--Define the call quality question-->
<question id="CallQuality" text="Please rate
   the voice quality of that phone call.">
   <prop name="ImageFile">cellnetwork.png</prop>
   <response widget="RadioButtonList">
    <option>Bad</option>
    <option>Poor</option>
    <option>Fair</option>
    <option>Good</option>
    <option>Excellent</option>
   </response>
</question>
</myexperience>
```

Listing 2.1 – MyExperience : Exemple de configuration de collecte de données

MyExperience facilite la définition d'une expérience collecte en proposant une abstraction appelée *Sensors Triggers* et *Actions* au-dessus du langage descriptif XML. Dans cette abstraction, les *triggers* combinent les flux des données fournis par les *sensors* et une expression logique pour déterminer quand une *action* doit être déclenchée.

Pour capturer l'expérience d'un utilisateur, cette abstraction supporte la définition de questionnaires, qui peuvent être présentés à l'utilisateur périodiquement, ou après un évènement spécifique. Le listing 2.1 montre par exemple une expérience affichant à un utilisateur un questionnaire lui demandant la qualité de sa communication vocale après un appel téléphonique. Cependant, l'abstraction proposée a plusieurs limitations. La première est que les triggers qui exécutent les actions ne peuvent pas exécuter une seconde action en fonction du résultat de la première. Cela empêche par exemple de déclencher un nouveau questionnaire en fonction des premières réponses fournies. La deuxième concerne l'expressivité de l'abstraction qui reste limitée aux balises XML définies au préalable. Cette limitation par exemple empêche le développement d'algorithmes contextuels complexes (*par ex.* inférence de l'activité d'un utilisateur) complexe. Et finalement, MyExperience reste limité aux collectes participatives.

L'architecture de la plate-forme MyExperience est composée de deux parties spécifiques : i) une application mobile Windows Mobile, responsable de de l'exécution des fichiers XML et ii) une application serveur, responsable de la persistance des données collectées. L'application mobile supporte également la propagation automatique des données vers le serveur. La propagation des données est effectuée lorsqu'une connexion réseau est détectée, et utilise le protocole HTTPS pour sécuriser le transfert des données. L'application fournit également une option permettant d'utiliser un algorithme de cryptographie SHA-1 pour hacher les données confidentielles des utilisateurs (*par ex.* SMS, contacts, numéro de téléphone). Cependant, MyExperience ne fournit pas de mécanisme particulier pour aider les scientifiques à déployer leurs expériences. L'application mobile ne supportant l'exécution que d'une seule expérience, cela empêche par exemple aux scientifiques de bénéficier des applications déjà installées chez les utilisateurs pour déployer leurs propres expériences.Néanmoins, MyExperience fournit plusieurs mécanismes permettant la mis à jours des expériences à distance, en envoyant la nouvelle expérience par SMS ou un message électronique aux utilisateurs concernés. Cependant, cela nécessite que les scientifiques aient connaissance du numéro de téléphone ou de l'adresse électronique des utilisateurs, ce qui met en péril l'anonymat des utilisateurs.

2.3.3 *Medusa : A programming framework for crowd-sensing applications*

Medusa, présenté par Ra et al. [55], est une plate-forme de programmation et de distribution de tâches de collecte participatives. Medusa s'adresse principalement aux utilisateurs finaux, sans expertise en programmation. Un exemple typique d'application présentée est celui du citoyen journaliste, permettant au journaliste de recruter des citoyens mobiles pour récolter des vidéos et des commentaires d'un évènement spécifique (*par ex.* accident, feu de forêt).

Dans Medusa, la spécification d'une nouvelle tâche consiste à définir une séquence d'actions qui doivent être exécutées par les mobiles. Pour spécifier cette séquence d'actions, Medusa propose MedScript, une abstraction de haut niveau basé sur XML. MedScript propose deux niveaux d'abstractions : les stages et les connecteurs.

Les stages permettent de définir une action élémentaire qui doit exécutée par le dispositif. MedScript distingue deux type de stages : i) les stages consistant à extraire des données des capteurs (*par ex.* GPS, accéléromètre) appelés `SPC-Stage`, et les stages nécessitant une intervention humaine (*par ex.* prendre une vidéo, documenté une photo) appelés `Hit-Stage`. Chaque stage inclut également plusieurs paramètres, comme une date d'expiration, un contexte d'exécution (*par ex.* région spécifique ou période de la journée) et également inclure une récompense financière pour inciter les utilisateurs à exécuter les stages.

Par exemple, le listing 2.2 décrit l'application du citoyen journaliste. Cette application comporte quatre stages et deux connecteurs définissant l'enchaînement suivant : `Recruit -> TakePicture -> GetSummary -> UploadData` . Le premier stage permet de recruter les utilisateurs, le deuxième demande aux utilisateurs de prendre une photo dans une région spécifique, le troisième demande aux utilisateurs de commenter la photo prise et finalement le dernier permet d'envoyer le commentaire et la photo vers le serveur.

```
1  <app>
2    <name>Citizen-Journalst</name>
3    <stage>
4      <name>Recruit</name> <type>HIT</type>
5      <binary>recruit</binary>
6      <config>
7      <stmt>Citizen Journalist Demonstration</stmt>
8      <expiration>18:00:00 12/16/2011</expiration>
9      <reward>.05</reward>
10     <output>W_WID</output>
```

```
11      </config>
12    </stage>
13    <stage>
14     <name>GetSummary</name> <type>SPC</type>
15     <binary>medusalet_videosummary</binary>
16     <trigger>immediate</trigger> <review>none</review>
17     <config>
18       <input>IMAGE</input>
19       <output>SUMMARY</output>
20     </config>
21    </stage>
22    <stage>
23      <name>TakePicture</name> <type>SPC</type>
24      <binary>medusalet_mediagen</binary>
25      <trigger>location=34.020259|-118.290131|40, user-initiated</trigger>
26      <config>
27        <params>-t image</params>
28        <output>IMAGE</output>
29      </config>
30    </stage>
31    <stage>
32      <name>UploadData</name> <type>SPC</type>
33      <binary>medusalet_uploaddata</binary>
34      <trigger>none</trigger> <review>textdesc</review>
35      <config>
36         <input>SUMMARY</input>
37      </config>
38     </stage>
39
40     <connector>
41       <src>Recruit</src>
42       <dst> <success>TakePicture</success> <failure>Hiring</failure> </dst>
43     </connector>
44     <connector>
45       <src>TakePicture</src>
46       <dst> <success>GetSummary</success> <failure>Hiring</failure> </dst>
47     </connector>
48     <connector>
49       <src>GetSummary</src>
50       <dst> <success>UploadData</success> <failure>Hiring</failure> </dst>
51     </connector>
52   </app>
```

Listing 2.2 – Le journaliste citoyen développé en MedScript language

Pour le déploiement des tâches de collecte, l'architecture de Medusa (*cf.* figure 3) est composé d'un ensemble de service exécuté sur une infrastructure serveur, et une

application mobile distribuée auprès d'un ensemble d'utilisateurs. Nous décrivons brièvement le rôle de ces composants.

Le `Interpreter` est le point d'entré de MEDUSA, il accepte les tâches décrites en MEDSCRIPT, vérifie leur validité et les envoies aux `Task Tracker`.

Le `Task Tracker` est responsable de coordonner l'exécution des tâches vers les dispositifs mobiles. Pour chaque tâche, le `Task Tracker` lui associe une instance dédiée responsable du suivie des différents stages associés à la tâche. Dans ce cas, si la tâche doit être exécuté par 50 utilisateurs (*c.-à-d.* collecter 50 images dans l'exemple présenté), 50 instances seront alors instanciées dans le `Task Tracker`. Pour chaque stage inclus dans la tâche, l'instance associé à la tâche notifie l'application mobile assignée lui demandant d'exécuter le stage. Une fois le stage complété, l'application mobile alerte le `Task Tracker` et attend une instruction pour exécuter le stage suivant. Une fois tous les stages terminés, le `Task Tracker` averti l'initiateur de la tâche et rend les données disponibles dans le `Data Repository`.

Le `Worker Manager` sert principalement d'interface avec le service Amazon Mechanical Turk (AMT) [8], qui est utilisé pour la phase de recrutement. Lorsque le `Task Tracker` instancie une nouvelle tâche, une description de la tâche ainsi que sa rémunération est postée sur le service AMT. Les participants peuvent alors se connecter sur le service AMT avec leur application mobile, et s'inscrire auprès de la tâche qu'il les intéresse. Une fois inscrit, le `Worker Manager` notifie le `Task Tracker`, qui peut alors procéder à l'exécution des stages. AMT est également utilisé pour assurer les communications (*c.-à-d.* par SMS) entre le `Task Tracker` et les applications mobiles, et la rémunération des utilisateurs. Cette approche permet d'assurer l'anonymat des utilisateurs, dans le sens où MEDUSA n'a pas besoin de connaître l'identité réelle des utilisateurs pour communiquer avec eux et les récompenser.

Le `Stage Library` représente la partie extensible de MEDUSA. Chaque stage supporté par MEDSCRIPT est associé à un exécutable Android stocké dans le `Stage Library`. Les exécutables sont alors téléchargés par les dispositifs mobiles avant d'exécuter le stage associé. Cette approche permet d'étendre les capacités offertes par MEDSCRIPT sans avoir à recompiler l'application mobile et la redéployer.

L' `Application mobile` est composée de deux services appelés `StageTracker` et `MedBox`. Le `Stage Tracker` est responsable des communications entre l'ap-

8. Amazon mechanical turk : https://www.mturk.com

plication mobile et les services du serveur. Ces communications comprennent le téléchargement des exécutables des différents stages, d'analyser les SMS envoyés par AMT et propager les données vers le serveur. La MedBox est responsable de l'exécution des stages dans le dispositif mobile. Un stage peut être exécuté soit directement après une notification du `Stage Tracker`, soit en fonction du contexte de l'utilisateur (*c.-à-d.* zone géographique ou période de la journée). L'application mobile fournit également plusieurs mécanismes pour contrôler les ressources consommées par l'application mobile et assurer la confidentialité des participants. Pour la gestion des ressources, Medusa laisse la possibilité aux participants de définir des limites en terme de CPU, réseau ou mémoire utilisées lors de l'exécution des stages. Pour assurer la confidentialité, l'application mobile demande une confirmation aux utilisateurs avant de propager les données sur le serveur. Cela permet aux utilisateurs de vérifier si les données collectées peuvent être compromettantes vis-à-vis de leurs vies privées.

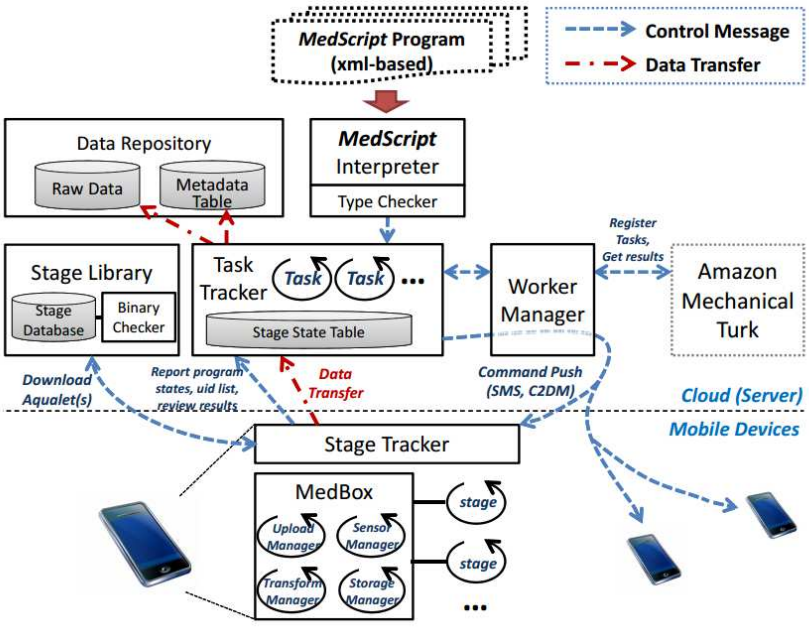

Figure 3 – Architecture de Medusa [55]

Medusa propose certains aspects intéressants à l'égard des considérations discutées dans la section précédente. Ces aspects sont l'intégration d'un service tiers comme AMT, pour assurer l'anonymat des utilisateurs, la possibilité d'intégrer des récom-

penses financières, ou encore proposer une application mobile générique favorisant la participation des utilisateurs à plusieurs tâches de collecte. Cependant MEDUSA a plusieurs limitations. La première réside lors de la phase de recrutement des utilisateurs. En effet, si nous reprenons l'exemple du citoyen journaliste, il faudrait que, idéalement, seul les utilisateurs se situant à proximité de l'événement couvert par le journaliste, puissent être capables de s'inscrire à la tâche. Or, bien que MEDUSA supporte la notion de contexte d'exécution (*par ex.* exécuter un stage dans une zone géographique), celle-ci est interprétée uniquement par l'application mobile après la phase d'enregistrement. Ainsi, de nombreux utilisateurs peuvent s'enregistrer sans qu'ils ne soient à proximité de l'événement visé.

Une autre limitation réside dans l'abstraction du modèle de programmation proposé, qui reste limité à la définition de tâche participatives et communautaires. Par exemple, cette abstraction ne permet pas de définir des tâches suivant l'évolution d'un utilisateur à travers une longue période (*par ex.* "collecter toutes les 5 minutes la position du GPS"). Et finalement, la dernière limitation réside dans les mécanismes d'extension de MEDUSA. En effet, pour étendre les fonctionnalités offertes par MEDSCRIPT, les développeurs ont besoin de développer un nouveau module Java compatible Android. Ce mécanisme a deux inconvénients. Le premier concerne la sécurité. En effet, un développeur mal intentionné peut développer un stage envoyant des SMS à l'insu des participants, ou récupérer des données confidentielles (*par ex.* contacts) et les envoyer vers un service tiers. Et le deuxième rend exclusive MEDUSA uniquement pour Android, empêchant de faire face à l'hétérogénie des OS mobiles.

2.3.4 *PRISM : Platform for Remote Sensing using Smartphones*

PRISM [16] est une plate-forme adressant particulièrement trois points qui sont la généralité, la sécurité et le passage à l'échelle.

Pour adresser la généralité, PRISM propose une application mobile permettant le téléchargement de tâches de collecte écrites en code natif. Cette approche permet de faciliter la réutilisation de code et de donner une totale flexibilité pour le développement de tâches complexes et variés. Dans ce cadre, PRISM supporte le développement de tâche participatives et opportunistes. Néanmoins, cette approche a plusieurs limitations. Premièrement, le développement de code natif demande une grande expertise en programmation et dans les plate-formes mobiles. PRISM ne fournit pas d'API de

plus haut pour simplifier leurs développements. Deuxièmement, le développement de code natif ne permet pas de faire face à l'hétérogénéité des OS mobiles. Le code développé reste alors exclusivement exécutable par OS mobile supporté par PRISM, en l'occurrence Windows Mobile. Et finalement, permettre à des développeurs tiers de déployer du code natif sur les dispositifs mobiles peut également poser des problèmes de sécurité et de confidentialité vis-à-vis des participants.

Pour limiter les problèmes de sécurité, PRISM implémente un mécanisme d'interposition, empêchant les tâches d'accéder à des fonctions critiques de l'appareil. L'application mobile observe également les ressources énergétique et réseaux consommées par les tâches, et stoppe leurs exécutions si elles dépassent une valeur limite. Cela permet de renforcer la sécurité de l'appareil, en évitant que des tâches épuisent totalement la batterie des utilisateurs. Pour permettre aux utilisateur de contrôler les données collectées durant l'exécution des tâches, PRISM fournit un mécanisme de contrôle d'accès comprenant trois niveaux : i) *No Sensors* empêchant tous accès, ii) *Location Only* permettant l'accès uniquement aux capteurs de position et *All Sensors* autorisant tous les accès.

En ce qui concerne le passage à l'échelle du système, PRISM propose une approche permettant de maîtriser le déploiement des tâches de collecte à un sous-ensemble d'utilisateurs. La figure 4 décrit l'architecture globale de PRISM comprenant deux parties spécifiques : i) une application mobile pour Window Mobile, responsable de l'exécution des tâches soumise par le serveur, et ii) le serveur PRISM, acceptant des tâches d'applications tierces pour les déployer auprès des applications mobiles.

Dans PRISM, le déploiement des taches de collecte est assuré via approche *push-based*, c'est-à-dire permettant au serveur de pousser les applications directement vers les dispositifs mobiles. Cette approche a le principal avantage d'assurer un déploiement rapide des applications, sans que les utilisateurs soient obligés de se connecter régulièrement sur le serveur pour voir si une nouvelle tâche est disponible. Pour permettre aux développeurs de définir un sous-ensemble de dispositifs pour déployer leurs tâches, PRISM fournit une API comportant deux niveaux de raffinements (*cf.* listing 2.3).

```
1 // set up the first level coarse-grained predicate
2 L1pred = new FirstLevelPredicate();
3 L1pred.location = <desired location>;
4 L1pred.radius = <desired coarse radius>;
5 L1pred.stationary = false;
6 L1pred.cameraPresent = true;
7 L1pred.numOfPhones = <desired number of phones>;
```

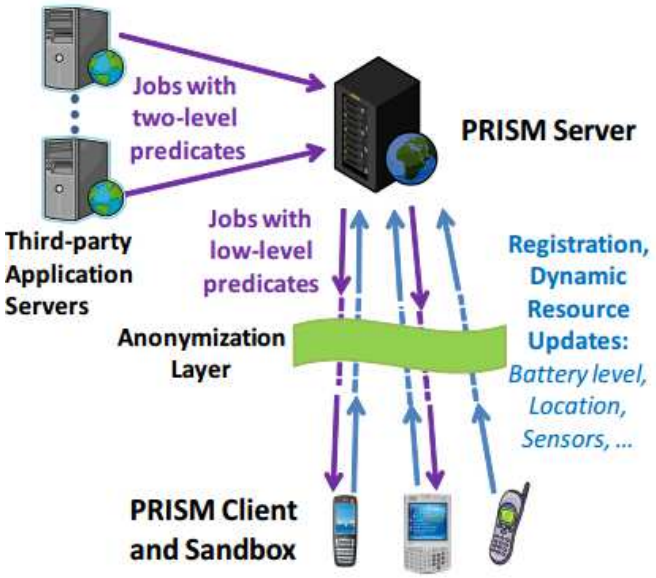

FIGURE 4 – Architecture de PRISM [16]

```
8  // set up the second level fine-grained predicate
9  L2pred = new SecondLevelPredicate();
10 L2pred.location = <desired location>;
11 L2pred.radius = <desired fine radius>;
12 // set up the application with the predicates
13 PRSIMapp = new PRISMApplication();
14 PRSIMapp.Init();
15 PRISMapp.SetPredicates(L1pred, L2pred);
16 PRISMapp.SetBinary(<path to 'phoneapp.exe'>);
17 PRISMapp.DistributeToPhones();
18 // read and process data sent by phones
19 while (appData = PRISMapp.GetData()) {
20 <process the received data>;
21 }
```

Listing 2.3 – Exemple d'application développée avec PRISM

Le premier niveau permet de spécifier les capteurs nécessaires pour exécuter la tâche, le nombre de mobiles désirés et une vaste région géographique. Le deuxième niveau permet quant à lui de définir contexte plus précis (*par ex.* lorsque l'utilisateur se déplace, se trouve dans une zone précise). Pour être en mesure de déployer les tâches de collecte, PRISM requiert une connaissance complète des dispositifs mobiles ainsi que leurs mobilités. Cette connaissance est assurée par une phase d'enregistrement assurée par les dispositifs mobiles. Lors de l'enregistrement, les dispositifs mobiles

reportent deux sortes d'informations comprenant des données statiques (*par ex.* capteurs disponibles) et des informations dynamiques (*par ex.* position et niveau de batterie restant). Cependant, cela nécessite que les dispositifs envoient constamment leurs positions pour permettre au serveur d'avoir une vision en temps réel de leur répartition. Cela peut causer par conséquent un trafic réseau important si de nombreux dispositifs sont connectés au serveur, et également négliger la confidentialité des utilisateurs. Lorsqu'une nouvelle tâche est soumise au serveur PRISM, celui-ci compare le premier niveau de raffinement avec tous les dispositifs enregistrés au préalable, et la déploie uniquement aux dispositifs correspondants au premier niveau de raffinement. Le deuxième niveau de raffinement sera ensuite interprété par l'application mobile, qui déclenchera l'exécution de la tâche en fonction des contraintes spécifiées.

2.3.5 *Bubble-sensing*

Bubble-sensing [45] est un système dédié aux déploiements de tâches de collecte participative et communautaire. Ce système focalise principale sur les méthodes de déploiements des tâches de collecte qui doivent être exécutées dans une région spécifique. La figure 5 illustre son architecture.

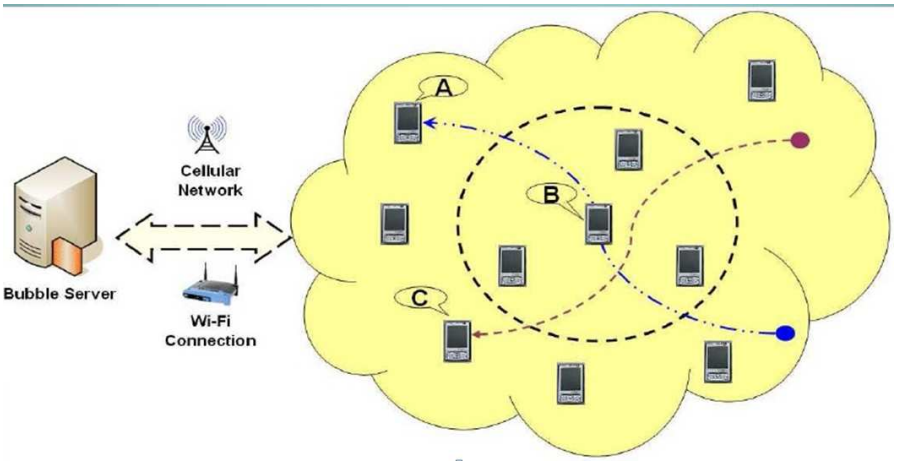

Figure 5 – Architecture de Bubble-sensing [45]

Dans le modèle proposé par Bubble-sensing, la création d'une tâche est initiée par le dispositif d'un participant (dispositif *A* dans la figure 5). Le dispositif joue alors le rôle de *bubble anchor*, et est responsable de la maintenance de la tâche de collecte dans la zone géographique d'intérêt. Une tâche est définie par un tuple : i) *action* qui définie

la tâche à réalisé par les participants (*par ex.* prendre une photo, une vidéo), *région* qui définie une zone géographique représentant la *bulle* où la tâche doit être exécutée, et une *durée* qui définie la date limite ou l'action doit être réalisé. La tâche est disséminée périodiquement, par des communications à courte distance (*par ex.* Bluetooth, Wifi direct), jusqu'à ce qu'un autre dispositif mobile reçoive et accepte d'exécuter la tâche en question (dispositif *C*). Les données collectées sont ensuite envoyées sur le *bubble-server*. Dans le cas ou l'initiateur de la tâche quitte la zone géographique initialement définie, un message est alors disséminé à tous les autres dispositifs se trouvant à proximité. Dans le cas ou un autre dispositif reçoive le message, il devient alors le *bubble anchor* (dispositif *B*) responsable de la maintenance de la tâche dans la bulle.

En utilisant cette approche, Bubble-sensing permet d'éviter aux dispositifs de reporter continuellement leur position auprès d'une entité centrale. Cependant, cela implique qu'il y est constamment la présence de participants volontaire pour jouer le rôle de *bubble-anchor* dans la zone géographique d'intérêt, sinon la tâche de collecte serait alors perdu.

2.3.6 *Pogo, a Middleware for Mobile Phone Sensing*

Pogo [6] est une plate-forme proposée par Brouwers et al., qui a pour objectif d'aider les scientifiques à développer et déployer des tâches de collecte à grande échelle.

Trois types d'acteurs sont identifiés dans Pogo. Les participants propriétaires des dispositifs mobiles, les scientifiques et un administrateur. Les participants mobiles exécutent une application mobile dédiée, développée pour Android. En installant l'application mobile, les participants acceptent de partager les ressources de leurs dispositifs pour exécuter des tâches développées par les scientifiques. Les scientifiques, quant à eux, peuvent exécuter une application sur leur ordinateur pour développer de nouvelles tâches et les déployer. Et finalement l'administrateur est responsable d'assigner un ensemble de participants aux scientifiques. Cette assignation est effectuée à partir d'un serveur central, qui garde également une trace de tous les dispositifs et des données qu'ils partagent. Cependant, les auteurs donnent très peu d'informations sur cette phase d'assignation, si elle est manuelle ou automatisée. Une fois un participant assigné à un scientifique, Pogo assure le déploiement des tâches par une approche *push-based*. Cette approche permet d'assurer un rapide déploiement des mises à jour des tâches, mais implique également un manque de transparence sur les données qui sont

collectées sur les dispositifs des participants ni avec qui ils les partagent. Nous pensons que ce manque de transparence peut freiner leurs participations, surtout qu'aucun mécanisme d'incitation n'est proposé.

Dans Pogo, les tâches de collecte peuvent être développées en JavaScript. Pour simplifier le développement des tâches, Pogo propose une abstraction basée sur le pattern *publish-subcribe* [21], permettant d'effectuer des communications asynchrone entre entre les scripts et les capteurs. La figure 2.4 illustre un exemple de tâches reportant toutes les minutes les points d'accès WiFi sur les ordinateurs des scientifiques. Comparée aux approches précédentes, l'utilisation du JavaScript comporte de nombreux avantages. Tout d'abord, cela permet de bénéficier de l'expressivité d'un langage générale, permettant ainsi de définir des algorithmes contextuels complexes. Ensuite, l'utilisation du JavaScript permet également de favoriser la réutilisation de code. Et finalement de nombreux projet open sources proposent des moteurs d'exécutions pour l'exécution de code JavaScript sur Android [9], iOS [10] ou Window Mobile [11], permettant d'assurer la portabilité du code développée vers diverses plate-formes. Cependant, l'abstraction proposée par Pogo ne permet pas d'interagir avec les participants, restant exclusivement réservé à la collecte opportuniste de données.

```
1 function start(){
2
3       subscribe('wifi-scan', function(msg){
4
5             publish(msg,'wifi-scan')
6
7       } , { interval : 60 * 1000 }) }
```

Listing 2.4 – Exemple d'application développée avec Pogo

Pogo protège la vie privée des utilisateurs finaux en cachant leur identité aux scientifiques. En outre, les utilisateurs finaux conservent le contrôle de leurs données et sont en mesure de contrôler les capteurs utilisés par l'application. Pogo prend également la consommation d'énergie en considération en coordonnant la transmission des données pour les faire coïncider avec les transmissions d'autres applications. Cela permet d'éviter d'activer les interfaces réseaux constamment qui engendre un grand coût énergétique.

9. Android : https://developer.mozilla.org/enUS/docs/Rhino
10. iOS :https://github.com/phoboslab/JavaScriptCore-iOS
11. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey

2.3.7 *AnonySense : A System for Anonymous Opportunistic Sensing*

Shin et al. introduisent une plate-forme appelée AnonySense [62], adressant majoritaire les problématiques de confidentialité dans les systèmes de collecte opportuniste.

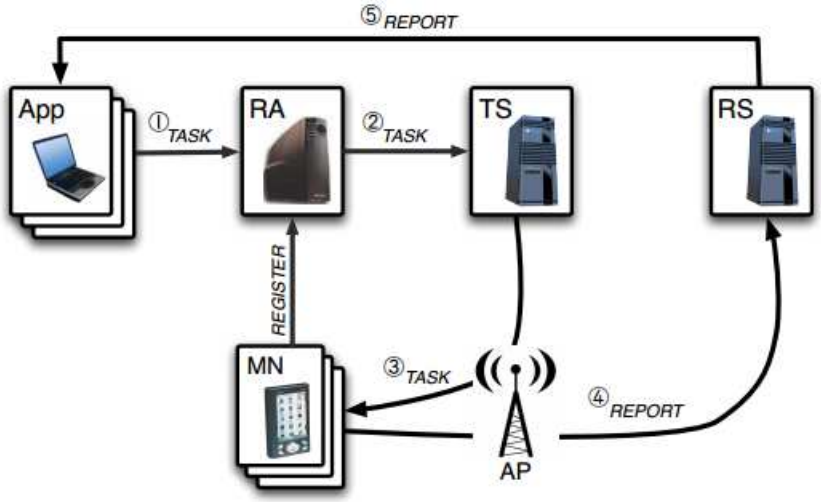

Figure 6 – Architecture de Anonysense [62]

```
1  ( Expires 1196728453 )
2  ( Accept (= @WiFi a/b/g ) )
3  ( Report (LOCATION APLIST)
4  ( Every 60 seconds )
5  ( In ( (1 1) (2 2) (3 0))))
```

Listing 2.5 – Exemple d'application développée en AnonyTL

Pour la description des tâches de collecte, Anonysense propose sur un langage dédié appelé AnonyTL, inspiré de la syntaxe Lisp. AnonyTL permet de spécifier le comportement d'une tâche en fournissant un ensemble de conditions d'acceptation, des reports de données, et une date d'expiration. Le listing 2.5 décrit un exemple de tâche exprimée en AnonyTL. Cette tâche peut être acceptée uniquement par des dispositifs connectés à une borne WiFi, et collecte l'ensemble des bornes WiFi voisines ainsi que la position toutes les 60 secondes lorsque l'utilisateur se trouve dans une zone précise. Les auteurs proposent, un nouveau argument, le choix d'un nouveau langage pour fournir une notation plus concise et plus compréhensible. Cependant, cette concision diminue également l'expressivité du langage, et empêche également la réutilisation de code pré-existant.

Le modèle proposé par Anonysense comporte les éléments suivants : un ensemble de dispositifs mobiles (MN), d'applications tierces (App), une autorité d'enregistrement (RA), un service de distribution de tâches (TS), et un service de rapport (RS) et un service d'anonymisation (AS). Dans ce modèle, un App soumet une tâche à l'autorité d'enregistrement (RA). Le RA vérifie ensuite si la tâche respecte la confidentialité des utilisateurs avant de la transmettre au service de distribution (TS). A intervalles réguliers, les MNs se connectent au TS et téléchargent l'ensemble des tâches disponibles. Les MNs choisissent ensuite les tâches qu'ils veulent accepter. Lorsqu'un rapport est prêt, les MNs l'envoient directement au RS ou au AS selon la sensibilité des données. Si le rapport contient des données sensibles, le rapport est envoyé au AS, dans le cas contraire, il est envoyé directement au RS.

2.4 Synthèse et conclusion

Dans cette section, nous faisons un bilan des plate-formes décrites dans la section précédente. Le tableau 7 fournit un récapitulatif de ces approches et les positionne par rapport aux défis décris en section 2.2.

En ce qui concerne les abstractions proposées pour le développement des tâches de collecte, seulement PRISM [16] supporte à la fois le développement de tâche opportuniste et participative, en permettant le déploiement de code natif directement vers les dispositifs mobiles. Cependant, il ne fournit par d'abstraction particulière permettant de simplifier le développement des tâches de collecte. De plus, le déploiement de code natif reste exclusivement réservé à un système d'exploitation mobile spécifique, dans ce cas Windows Mobile.

Pour le déploiement et le recrutement des tâches de collecte, principalement deux approches coexistent.

La première est l'approche *pull-based*, adoptée par Medusa [55] et Anonysense, où les dispositifs sont responsables de télécharger les tâches de collecte auprès du serveur. Le principal avantage de cette approche est qu'elle permet d'assurer un certain degré de transparence, dans le sens où les participants peuvent sélectionner les tâches qu'ils souhaitent exécuter. Cependant, le temps de propagation des tâches peut être relativement long, et demande aux participants de vérifier constamment si une nouvelle tâche est disponible. Pour le recrutement des participants, Anonysense permet

de sélectionner un sous ensemble de participants basé sur leur position. Pour assurer un temps de propagation rapide malgré l'approche *pull-based* utilisée, l'application mobile télécharge régulièrement toutes les tâches de collecte disponibles sur le serveur et exécute celles qui correspondent à son contexte. Cependant, cela peut impliquer un coût énergétique important des applications mobiles si de nombreuses tâches sont disponibles sur le serveur et ne correspondent pas au contexte du dispositif.

La deuxième est l'approche *push-based*, adoptée par PRISM [16] et Pogo [6], où au contraire c'est le serveur qui déploie les tâches directement auprès des dispositifs mobiles. Son principal avantage est qu'elle permet un déploiement rapide des tâches de collecte, qui peut être très bénéfique dans le cadre où une rapide mise à jour de l'application doit être effectuée. Cependant, cette approche manque de transparence, dans le sens où les participants n'ont pas conscience des données qui sont collectées sur leur dispositif mobile. Pour le recrutement des utilisateurs, PRISM propose également un modèle basé sur la position des participants. Afin de permettre au système d'identifier les participants se situant dans une région spécifique, les applications mobiles ont besoin de reporter continuellement leur position sur le serveur, ce qui peut causer par conséquent un trafic réseau important si de nombreux dispositifs sont connectés au serveur.

Finalement, la majorité de ces plate-formes propose une architecture centralisée, composée d'une application serveur pour le déploiement des tâches de collecte et une application mobile responsable de leurs exécutions. Cependant, ce style architectural impose à tous les utilisateurs du système de partager les ressources de l'infrastructure hébergeant la plate-forme, des mécanismes de stockage de données, le modèle financier imposé par le fournisseur de l'infrastructure et également la législation du pays où l'infrastructure est hébergée. Cela impose également les modèles mis en place pour déployer une application vers les terminaux mobiles, des mécanismes pour protéger la vie privée des utilisateurs, de récompense ainsi que la structure des données collectées. Dans ce contexte, nous pensons que ce type d'architecture limite fortement la réutilisation du système dans un contexte d'application non initialement prévu, ainsi que son passage à l'échelle. De plus, l'entité centrale représente un point unique de défaillance, qui, en cas d'attaque, peut compromettre l'ensemble des données collectées.

Dans ce chapitre, nous avons présenté une vue d'ensemble du *Mobile crowdsensing*, ses caractéristiques, ses avantages ainsi que les défis qui doivent être adressés pour le déploiement de campagnes de collecte de données à grande échelle. À cause de ces

nombreux défis, la mise en place de campagnes de collecte reste encore difficile, et reste réservée à des développeurs experts. Bien que de nombreuses plate-formes ont été proposées dans l'état de l'art pour simplifier le développement et le déploiement des ces campagnes, les solutions proposées restent encore restreintes à un contexte d'application particulier. De plus, la centralisation de leur architecture limite fortement la réutilisation de ces systèmes dans un autre contexte applicatif ainsi que leurs passages à l'échelle. À partir de ce constat, cela nous a mené à proposer APISENSE®, une plate-forme répartie pour la conception, le déploiement et l'exécution de campagnes de collecte de données. Plus particulièrement, APISENSE® c'est :

1. Un modèle de programmation facilitant le développement de tâches de collecte participatives et opportunistes
2. Un environnement mobile sécurisé assurant l'exécution des tâches de collecte
3. Une architecture décentralisée permettant d'assurer le passage à l'échelle de la plate-forme
4. Un modèle permettant de configurer une application serveur responsable du déploiement des tâches de collecte, de la persistance ainsi que de l'exploitation des données collectées.

Dans le chapitre suivant, nous présentons notre première contribution concernant le modèle de programmation des tâches de collecte ainsi que l'environnement mobile dédié à leur exécution.

		Funf	MyExperience	Medusa	Anonysense	Prism	Pogo	Bubble sensing
Development/ heterogeneity	Language	-	XML	XML	AnonyTL	C#	JavaScript	-
	Abstraction	User interface	Sensors triggers actions	Stages connectors	AnonyTL	No	Publish-subscribe	Action-region duration
	Cross-platform	No	Potentially	No	No	No	Potentially	No
	Extensibility	Code-level	Code-level	Code-level	No	**Abstraction-level**	**Abstraction-level**	No
Generality	Participatory	No	**Yes**	**Yes**	No	**Yes**	No	**Yes**
	Opportunistic	**Yes**	No	No	**Yes**	**Yes**	**Yes**	No
	Personal	**Yes**	**Yes**	No	No	No	Yes	No
	Community	No	No	**Yes**	**Yes**	**Yes**	No	Yes
Acceptation	Transpancy	No	No	**Yes**	**Yes**	No	No	No
	Feedback	No	No	**Yes**	**Yes**	No	No	No
	Incentive	No	No	**Monetary**	**Reward**	No	No	No
	Privacy	-		-	Transparency	k-anonimity mix-network	Control access	-
	Energy	-	-	Network and CPU metering	-	Network and sensors metering	Tails detection	-
Scalability	Deployment/update	Manual	Manual	Pull	Pull	Push	Push	Broadcast
	Recruitment	Manual	Manual	-	Location sensor-type	Location sensor-type	Manual	Location
	Coordination	No	No	No	No	No	No	No
Architecture	Database	-	MySQL	MySQL	-	-	-	-
	Flexible	-	-	No	No	No	No	No
	Extensibility	-	-	No	No	No	No	No
	Style	-	Centralized	Centralized	Centralized	Centralized	Centralized	hybrid

FIGURE 7 – Tableau comparatif des plate-formes MCS

Deuxième partie

Contributions

3

COLLECTE MOBILE DE DONNÉES

Sommaire

3.1 INTRODUCTION

Le Mobile Crowd Sensing (MCS) est un mode de collecte de données qui a récemment été défini comme :" *la participation d'un groupe d'individus, disposant de terminaux mobiles intelligents, qui collectivement, partagent des informations pour la mesure ou la cartographie de phénomènes d'un intérêt commun.*" [27]. Ces dernières années, le MCS a suscité l'intérêt d'un grand nombre d'acteurs académiques, dans des domaines tels que l'étude de la mobilité urbaine, la surveillance de l'environnement, la santé ou l'étude des comportements sociaux [9].

Cependant, le développement de ces applications reste une tâche complexe. En effet, ces applications ont besoin de faire face à de nombreuses propriétés non fonctionnelles telles que les ressources énergétiques limitées des dispositifs mobiles, le respect de

la confidentialité des utilisateurs mobiles, ou encore le coût du recrutement d'un nombre important d'utilisateurs. À cause de cette complexité, il peut être très difficile pour de nombreuses acteurs de développer leurs applications capables de récolter la masse de données nécessaires pour leurs études. Ces dernières années, de nombreuses communautés scientifiques, utilisant le MCS dans leur processus expérimental, ont beaucoup discuté des bénéfices que pourrait apporter un système facilitant ce mode d'acquisition de données [27, 60, 66, 68].

Dans ce contexte, nous proposons dans cette thèse APISENSE®, une plate-forme visant de nombreux acteurs privés, publics ou académiques, leurs permettant de facilement développer et déployer des applications de collecte de données à travers des utilisateurs mobiles. Dans le chapitre section 1.4, page 13 , nous avons présenté une vue d'ensemble d'APISENSE®. Nous rappelons brièvement que le système est composé de trois entités logicielles i) un serveur central responsable du déploiement de tâches de collecte, ii) des nœuds de collecte dédiés aux utilisateurs, responsable du déploiement des tâches de collecte vers le serveur central et la persistance des données collectées, et finalement iii) un agent mobile, dédié aux participants (*c.-à-d.* utilisateur mobile) responsables de l'exécution des tâches de collecte.

Dans ce chapitre, nous présentons notre première contribution qui consiste à proposer un modèle de programmation visant à minimiser l'expertise nécessaire liée aux développements d'applications de collecte, et la conception d'un environnement d'exécution mobile dédié à aux applications développées par notre interface de programmation.

STRUCTURE DU CHAPITRE La suite du chapitre est organisée comme suit : dans la section 3.2, nous faisons un bref rappel des problématiques et des solutions proposées dans l'état de l'art, et identifions les considérations prises en comptes pour la conception de notre solution. Nous présentons par la suite en section 3.3 une vue d'ensemble de l'interface de programmation proposée pour le développement des tâches de collecte. En section 3.4, nous présentons l'architecture du substrat d'exécution dédié à l'exécution des tâches de collecte. Nous présentons en section 3.5 l'évaluation d'un prototype réalisé sur Android, avant de conclure en section 3.6.

Afin d'avoir une large applicabilité, permettant de définir une grande variété de tâches de collecte, nous devons proposer un modèle de programmation qui demande de prendre en considération principalement les trois points suivants :

Diversité des tâches de collecte : Une tâche de collecte consiste à définir *quel* est le type de données (*par ex.* position, image), *quand* le processus de collecte doit être effectué (*par ex.* toutes les 5 minutes, lorsqu'un événement spécifique apparait) et *comment* une donnée doit être capturée. En ce qui concerne le comment, typiquement deux méthodes peuvent être utilisées. La première demande une interaction avec l'utilisateur (appelée collecte participative [7]), en lui demandant de répondre à un questionnaire ou prendre une photo par exemple. La deuxième s'effectue de manière autonome (appelée collecte opportuniste [39]), en interagissant directement avec les capteurs mobiles embarqués dans les terminaux mobiles.

Diversité des traitements locaux : Pour la définition d'une tâche de collecte, quatre préoccupations doivent être prises en considérations : i) la limitation des ressources énergétiques, ii) la quantité des données générées et propagées vers le serveur, iii) la qualité des données et finalement iv) la confidentialité des participants. Prendre en compte ces considérations demande généralement de faire des compromis entre elles, en effectuant un traitement local avant de propager les données vers le serveur. L'objectif de ces traitements est de diminuer les données à propager sur le serveur, en remontant uniquement les données essentielles pour l'expérience, économisant ainsi la batterie de l'utilisateur par la réduction de communications réseau. Ces traitements peuvent consister à filtrer les données aberrantes, inférer une information contextuelle de haut niveau à partir de données brutes (*par ex.* mode de transport), ou encore alterner l'utilisation des capteurs en fonction de leurs consommations énergétiques (*par ex.* GPS vs géolocalisation GSM).

Diversité des plate-formes mobiles : D'autre part, permettre le déploiement des tâches de collecte vers un grand nombre de participants demande également de prendre en compte la diversité des plate-formes mobiles disponibles. Ceci amène généralement à étudier et réaliser une application mobile spécifique pour chacune d'entre elles. Cette diversité a été soulignée par l'expérience menée par Balan et coll. [4], qui ont mis plus

de six mois pour le développement et le déploiement d'une expérience de collecte à travers 15,000 taxis à Singapour.

En tant que plate-forme de collecte de données, APISENSE® a deux publics distincts : i) les utilisateurs qui veulent définir de nouvelles campagnes de collecte de données, ii) les participants qui exécuteront ces tâches de collecte sur leur dispositif mobile. Dans ce contexte, à partir des considérations décrites ci-dessus et de ces deux publics distincts évoluant dans APISENSE®, nous identifions les objectifs de conception suivants :

Objectif de conception pour les utilisateurs : Afin de minimiser le niveau d'expertise nécessaire liée aux développements des applications de collecte, il est nécessaire de fournir une abstraction facilitant leurs développements. Dans ce contexte, l'objectif est de proposer un langage de programmation reposant sur trois propriétés :

1. *la généralité :* pour supporter une grande variété d'activités de collecte (*c.-à-d.* participative et/ou opportuniste), qui peuvent également impliquer une grande variété de capteurs.
2. *l'accessibilité :* permettant de minimiser l'expertise nécessaire dans les technologies mobiles afin d'accéder aux fonctionnalités offertes par les capteurs des terminaux mobiles, ainsi que leurs collectes et leurs propagations vers l'infrastructure serveur.
3. *la portabilité :* afin d'éviter de devoir définir plusieurs tâches de collecte pour différentes plate-formes mobiles, l'abstraction proposée doit être en mesure de s'exécuter sur ces différentes plate-formes.

Objectif de conception pour les participants : Inévitablement, les participants mobiles jouent un rôle central dans le processus de collecte. Afin de favoriser leurs participations, nous pensons que la confiance et la transparence doivent être un aspect fondamental du système. Cela nécessite que les utilisateurs mobiles aient connaissance des données qui sont collectées sur leurs terminaux, avec qui ils les partagent et dans quel but. D'autre part, favoriser l'acceptabilité des utilisateurs demande de fournir un substrat d'exécution permettant de :

1. *Contrôler la confidentialité :* en laissant la possibilité aux utilisateurs de choisir les données qui sont partagées, les capteurs qui peuvent être utilisés par les

applications de collecte, et dans quelle circonstance les données peuvent être partagées.

2. *Minimiser l'impact énergétique :* lié à l'exécution d'une application de collecte, pour ne pas empêcher une utilisation normale du dispositif. Notamment si de multiples applications doivent être exécutées sur un seul dispositif.
3. *Récompense adaptée :* pour favoriser la participation des utilisateurs mobiles, en rendant ludique le but de la collecte de données, ou en récompensant les utilisateurs financièrement selon la nature des données collectées.

3.3 LANGAGE DE PROGRAMMATION DES COLLECTES

Pour le développement des tâches de collecte, nous avons opté pour le langage de script JavaScript. Le choix du JavaScript est motivé principalement pour trois raisons. Premièrement, JavaScript est un langage accessible et populaire, bénéficiant de l'expressivité d'un langage générale. Cela donne aux utilisateurs une grande liberté pour définir des traitements complexes, intégrer des libraires déjà existantes de traitement de données (*par ex.* clustering) pour inférer des données de haut niveau directement dans le dispositif mobile. Ensuite, JavaScript est facilement transportable et exécutable sur de nombreuses plateformes mobiles, permettant un rapide déploiement et une rapide propagation des mis à jour de la tâche de collecte. Et finalement, d'un point de vue plus technique, JavaScript est nativement exécuté dans un bac à sable (*sandbox* en anglais), facilitant le contrôle des fonctionnalités des dispositifs mobiles accessibles par les scripts.

Pour faciliter le développement des tâches de collecte, nous proposons une interface de programmation (API) dédiée à la collecte de données, au-dessus de JavaScript. Le principal objectif de l'API est de fournir une abstraction complète des technologies mobiles, permettant le développement d'application de collecte de données, sans avoir une expertise particulière dans ces technologies. Dans APISENSE, une tâche de collecte est spécifiée par un ou plusieurs scripts JavaScript, qui peuvent être déployés et exécutés sur les dispositifs mobiles des participants. Dans la suite de cette section, nous présentons une vue d'ensemble de l'API proposée.

3.3.1 *Concept de l'interface de programmation*

Comme le montre la figure 8, notre API consiste en un ensemble de variables prédéfinies, appelée façades, définissant l'ensemble des fonctionnalités accessibles par les scripts. Plus précisément, une façade permet d'interagir avec les ressources d'un dispositif, et est implémentée dans le langage natif de la plate-forme d'exécution. Ces ressources peuvent être par exemple des capteurs physiques (*par ex.* température, GPS), des capteurs logiciels (*par ex.* appel téléphonique, installation d'une application) ou encore l'interface graphique du dispositif mobile. Dans notre API, chaque façade proposée fait référence à une ressource spécifique, et délimite un sous-ensemble de fonctionnalités fourni par celle-ci. Ce procédé permet ainsi d'empêcher l'utilisation complète des fonctionnalités des dispositifs mobiles, pour des raisons de sécurité et de confidentialité vis-à-vis du participant. Par exemple, cela permet de crypter toutes les informations sensibles (*par ex.* contacts, numéro de téléphone, etc.), d'empêcher l'installation de nouvelles applications ou encore de faire des appels téléphoniques. Également pour des raisons de confidentialité, nous laissons la possibilité aux participants de contrôler les façades accessibles par les scripts de collecte. Par exemple, un participant peut à tout moment enlever la possibilité à un script de déterminer sa position, ou d'être notifié lorsqu'il reçoit des SMS.

La communication entre les scripts et les façades peut s'effectuer de deux manières différentes. La première, de manière asynchrone, permet aux scripts d'être notifiés du changement des états des capteurs des dispositifs mobiles. La deuxième, de manière synchrone, permet aux scripts de collecte d'effectuer une action spécifique, ou alors de lire directement le dernier état d'un capteur.

3.3.2 *Collecte de données dirigée par les évènements*

L'écoute des évènements permet de déclencher une action ou un traitement en fonction du contexte de l'utilisateur mobile. Par exemple, la collecte de données peut être déclenchée uniquement lorsque l'utilisateur est en situation de mobilité. L'observation du contexte consiste principalement à observer le changement d'état d'un capteur interne du dispositif. Pour interagir avec les capteurs, les façades proposent un ensemble de fonctions permettant aux scripts d'être notifiés de ce changement d'état.

Le tableau 1 montre quelques exemples de façade et des fonctions associées dédiées à l'observation d'un événement particulier. L'observation consiste donc à souscrire une

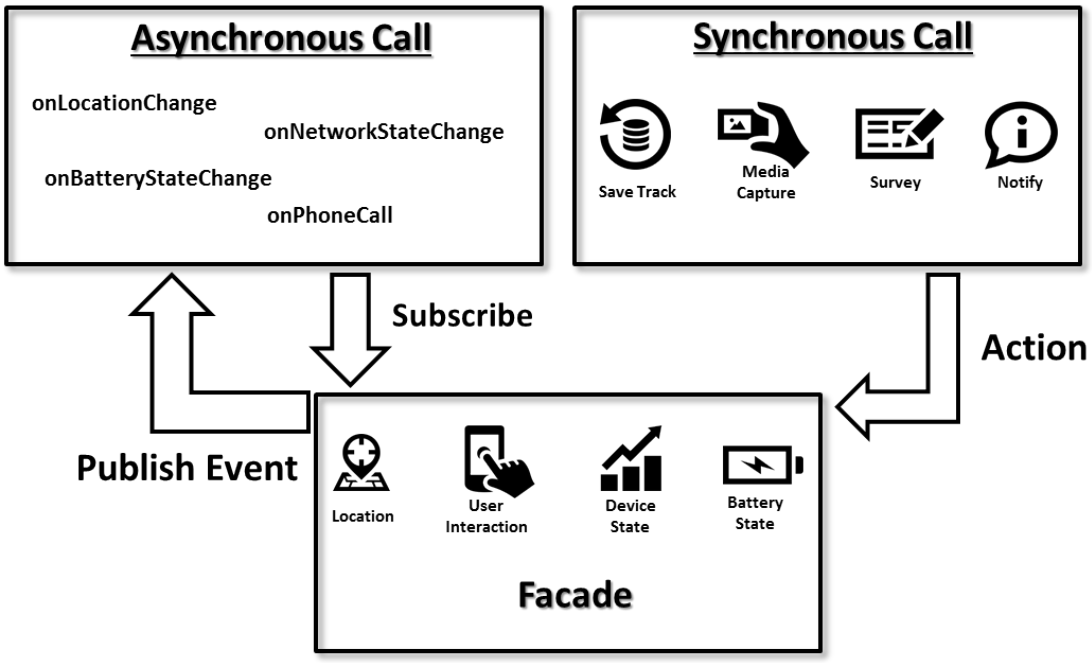

FIGURE 8 – Interface de Programmation APISENSE

Façade	Fonction	Description
`$location`	`onLocationChange`	Observation du changement de position
`$battery`	`onBatteryStateChange`	Observation de l'état de la batterie
`$acc`	`onAccelerometerChange`	Observation de l'accélération
`$phone`	`onPhoneCall`	Observation des appels téléphonique

TABLE 1 – Liste des fonctions d'observation d'événements prédéfinis.

Paramètre	Type	Description
`configuration`	`JSONObject`	Configuration de l'observation de l'état du capteur
`callback`	`function(event,subscription)`	Fonction de rappel
`condition (optionnel)`	`String`	Condition logique

TABLE 2 – Liste des paramètres d'une fonction de rappel

fonction de rappel, à une façade responsable du capteur capable de générer l'événement voulu. La souscription comprend trois paramètres, permettant la configuration du capteur, la fonction de rappel et une expression logique permettant de raffiner l'observation des événements, et retourne un objet de type *Subscriber*.

CONFIGURATION Le premier paramètre *configuration* laisse la possibilité de configurer l'observation du capteur en question. La configuration peut être utilisée afin de faire un compromis en qualité des données transmises par les capteurs (*c.-à-d.* fréquence d'échantillonnage, précision) et l'énergie consommée liée à l'observation du capteur. Un exemple classique est l'observation du changement de la position d'un utilisateur. Généralement, ce compromis peut être effectué en alternant le type de capteur utilisé — entre le GPS fournissant une grande précision, mais consommant beaucoup d'énergie, et le réseau qui au contraire consomme moins d'énergie, mais fournit une localisation moins précise —, ou la fréquence d'échantillonnage. Le tableau 2 illustre les configurations du capteur de position possibles.

Fonction	**Description**
`period`	Période d'échantillonnage de la position
`provider`	Capteur de position utilisé : `gps` ou `network`

TABLE 3 – Configuration de l'observation de la position d'un utilisateur

FONCTION DE RAPPEL Le deuxième paramètre permet d'enregistrer une fonction qui sera exécutée lors du changement d'état du capteur observé. Par exemple, le listing suivant montre la syntaxe permettant à un script de collecte d'être notifié lorsque l'utilisateur est en situation de mobilité :

```
$location.onLocationChanged({provider : ["gps"]},function(event){
 // code exécuté lorsque une nouvelle position est détectée
})
```

Chaque événement est défini par un ensemble d'attributs, sérialisés par un ensemble de clés/valeurs utilisant le formalisme JSON. Chaque événement est au minimum défini par un nom, et la date de sa production. Le listing suivant illustre un exemple d'événement produit lorsqu'une nouvelle position de l'utilisateur est détectée :

```
{
 event : "location",
 time : 18005010510,
 latitude : 50.65,
 longitude : 2.11,
 accuracy : 100,
 speed : 3
}
```

Dans cet exemple, l'événement est défini par cinq attributs indiquant la position de l'utilisateur (*c.-à-d.*`latitude`, `longitude`), la précision de la position (*c.-à-d.*`accuracy`) et la vitesse de l'utilisateur (*c.-à-d.*`speed`).

CONDITION LOGIQUE Optionnellement, l'écoute d'événement peut être raffinée en insérant une condition lors de la souscription. La condition se définit par un opérateur appliqué à un attribut appartenant à l'événement et à une valeur. Plusieurs conditions peuvent être exprimées en ajoutant des opérateurs logiques de type *et*, *ou* ou *non*. (*cf.* listing 3.1). Le listing suivant illustre un exemple utilisant une expression logique, implémentant une solution naïve pour déterminer si l'utilisateur est en train de marcher :

```
$location.onLocationChanged({provider : ["gps"]},function(event){
 // code exécuté lorsque l'utilisateur marche
},"&((speed > 3),(speed < 5))")
```

Dans cet exemple, nous observons donc le changement de position, et nous définissons que la fonction de rappel doit être exécutée uniquement si l'utilisateur se déplace à une vitesse comprise entre trois et cinq Kilomètres/h.

```
1 expression ::= key'('[condition+]')' | condition
2 condition ::= eventAttribute operator value
3 key ::= ( "!" | "&" | "|" )
4 operator ::= ( "<" | "==" | ">" | "<=" | ">=" )
5 eventName ::= alpha*
6 eventAttribute ::= alpha*
7 value ::= alphanumeric*
```

Listing 3.1 – Grammaire BNF d'une expression logique

OBJET DE SOUSCRIPTION Pour maintenir un lien entre la souscription et le script, un objet de type `Subscriber` (*cf.* table 4) est retourné lors de l'enregistrement. L'objet de souscription permet d'annuler la souscription à un événement particulier

(fonction `suspend`), permettant d'éteindre un capteur actif afin de conserver l'énergie du dispositif mobile. La méthode `resume` permet de reprendre une souscription qui a été annulée.

Fonction	Description
`suspend`	Suppression de l'observation
`resume`	Redémarrage de la souscription

TABLE 4 – Liste des fonctions d'un objet de souscription

3.3.3 *Les actions*

En plus de l'observation des événements, les façades fournissent également des fonctions permettant d'exécuter des actions spécifiques. Dans la suite de cette sous-section, nous présentons les principales actions prédéfinies par notre API qui sont la collecte et la propagation de données, l'interaction avec l'utilisateur mobile ou encore définir des informations pouvant être fournies en retour aux utilisateurs sur les données collectées.

Collecte et propagation des données

La collecte de données est effectuée à l'aide de la façade dédiée `$trace`. Cette façade possède trois fonctions comme le montre le tableau 5. La première permet de sauvegarder une donnée dans la base de données locale du terminal mobile (méthode `add`). Les données sont sous la forme d'un objet JSON, n'imposant aucune structure de données particulière pour représenter les données collectées. Cela laisse ainsi une grande flexibilité aux utilisateurs de définir leurs propres structures. Additionellement, les utilisateurs peuvent ajouter des méta-informations (méthode `addHeader`), permettant de diminuer la collecte de données redondantes, et leurs agrégations une fois propagées vers le serveur. Ces méta-informations représentent généralement des données statiques, c'est à dire qui n'évoluent pas dans le temps. L'API permet également de définir la stratégie pour propager les données vers les nœuds de collecte grâce à la méthode `send`. Lors de l'appel à cette fonction, la façade propagera les données sauvegardées dans la base de données locale si une connexion est disponible, sinon les données seront propagées une fois la connexion retrouvée. Dans le cas où aucune stratégie n'est définie, les données seront propagées automatiquement une fois que le

dispositif est branché sur secteur et possède une connexion réseau. Cela permet de diminuer la consommation énergétique des dispositifs des participants.

Fonction	Description
`add(data)`	Collecter une donnée
`addHeader(key,value)`	Collecter une donnée statique
`send()`	Propagation des données collectées

TABLE 5 – Liste des fonctions de la façade de collecte

Interaction avec l'utilisateur

Collecter des données de façon transparente peut ne pas être suffisant dans certains cas. En effet, certaines informations comme la satisfaction d'un utilisateur ou ses intentions sont difficilement perceptibles à partir de donnée brute. Prenons par exemple une collecte de donnée visant à observer la qualité du signal réseau d'un utilisateur mobile. Il peut être intéressant de voir comment la qualité du signal réseau influe sur la qualité de la communication vocale. Cependant, cette information peut être très difficilement inférée à partir des données brutes issues des capteurs. Dans ce contexte, une solution possible est de proposer des questionnaires d'évaluation aux utilisateurs. Ce type de collecte est également appelé collecte participative. Pour supporter la collecte participative, notre API dispose d'une façade permettant la création et la publication de questionnaire aux utilisateurs mobiles. Le tableau 6 illustre les principales fonctions fournies par cette façade.

La création d'un nouveau questionnaire s'effectue à partir de la fonction `create` de la façade `$survey`. La fonction retourne un objet de type `Survey`, disposant d'un ensemble de fonctions permettant d'insérer des questions ouvertes (*par ex.* zone de texte), de question fermée (*par ex.* case à cocher), ainsi que des captures multimédias (*c.-à-d.* photo, vidéo, audio). La fonction `then` permet de modifier la séquence des questions proposée à l'utilisateur, qui est par défaut dans l'ordre de leurs créations. Cela permet par exemple de proposer des questions différentes selon les premières réponses données par l'utilisateur. Nous illustrerons un exemple de création de questionnaire dans la sous-section 3.5.1 .

Facade $survey		
Return	**Fonction**	**Description**
`Survey`	`create`	Création d'un nouveau questionnaire
`void`	`publish(survey,callback)`	Publication du questionnaire
Type Survey		
`void`	`close(label,question,responses)`	Création d'une question fermée
`void`	`open(label,question)`	Création d'une question ouverte
`void`	`image(label,format)`	Capture photo
`void`	`audio(label,format)`	Capture audio
`void`	`video(label,format)`	Capture vidéo
`void`	`then(label,callback)`	Spécification de la prochaine fonction

TABLE 6 – Liste des fonctions de la façade dédiée à la création des questionnaires

Retour utilisateur

Afin de favoriser leurs participations des utilisateurs, et surtout maintenir les utilisateurs dans la durée, nous avons intégré à notre API la possibilité d'effectuer facilement des retours visuels sur les données collectées par l'utilisateur mobile. Ces retours visuels peuvent être directement développés utilisant le couple JavaScript/HTML, fournissant une grande flexibilité pour concevoir des retours selon la nature de collecte de données. Pour simplifier leurs développements, nous proposons des modules complémentaires qui peuvent être inclus dans la tâche de collecte lors de son déploiement, ou téléchargés dynamiquement lors de l'exécution du script. Actuellement nous proposons deux modules génériques permettant de créer automatiquement des graphes ou des cartes géographiques à partir des données collectées. Par exemple, le listing 3.2 décrit un exemple l'utilisation d'un module facilitant la génération de graphe. Dans cet exemple, les trois premières permettent de charger le module (ligne 1), créer un nouveau graphique (ligne 2) et rendre disponible le graphique pour les utilisateurs (ligne 3). Les lignes suivantes permettent d'observer les évènements relatifs aux changements de la force du signal GSM (ligne 7), et d'ajouter une nouvelle valeur au graphique lorsqu'un nouvel évènement est détecté (ligne 9). La figure 9 montre une capture d'écran du rendu graphique de cet exemple.

```
// signal strenght chart configuration
var chartLib = require("lib-chart.js")
   chart  = chartLib.area({name : "GSM Signal Strength Chart"});
   chart.publish();

// telephony sensor monitoring initialization
$telephony.onSignalStrengthChanged(function(event){

      chart.add({ x : event.time, event.level });
});
```

Listing 3.2 – Exemple de retour utilisateur

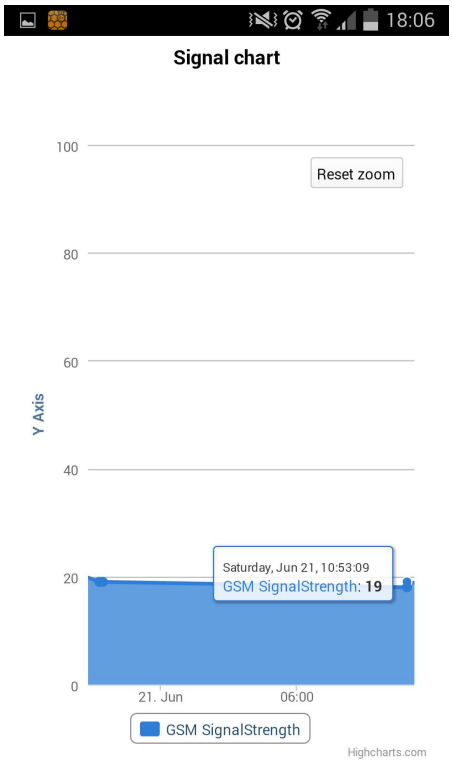

FIGURE 9 – Exemple de capture d'écran d'un retour utilisateur

Dans cette section, nous avons présenté une vue d'ensemble de notre API, dédié aux développements des tâches de collectes de notre système. Une documentation plus détaillée est également disponible le site internet dédié à APISENSE [1]. Dans la

1. http://apisense.io

section suivante, nous présentons les concepts et l'architecture de l'intergiciel dédié à l'exécution de l'interface de programmation présentée dans cette section.

3.4 INTERGICIEL DE COLLECTE

Dans la première section de ce chapitre, nous avons présenté une interface de programmation dédiée aux développements de tâche de collecte en environnement mobile. Dans cette section, nous présentons en détail l'architecture de l'intergiciel responsable de l'exécution des tâches de collecte. D'un point de vue utilisateur, l'intergiciel se décline sous la forme d'une application mobile, qui peut être téléchargée et installée afin de participer aux expériences de collecte de données. Dans ce cadre, l'application peut être perçue comme un conteneur générique de tâches de collecte, capable d'interagir avec les différents serveurs pour la propagation des données collectées (vers les nœuds de collecte), le téléchargement des tâches de collecte (auprès du serveur central) ainsi que leurs exécutions.

La figure 10 illustre une vue d'ensemble de l'architecture de l'intergiciel se déclinant en trois couches. Dans la suite de cette section, nous présentons en détails chacune de ces couches dédiées à : i) l'exécution des tâches de collecte (*cf.* sous-section 3.4.1), au contrôle des ressources du dispositif mobile (*cf.* sous-section 3.4.2), et iii) aux interactions avec les différentes entités logicielles d'APISENSE® (*cf.* sous-section 3.4.3).

3.4.1 *Couche d'exécution*

Nous présentons ici la couche responsable de l'exécution des tâches de collecte développées avec l'interface de programmation que nous avons présentée dans la section précédente. La première couche de l'architecture présentée est responsable de l'exécution des tâches de collecte. La Figure 11 illustre plus en détail les différents modules et les interactions nécessaires à l'exécution des tâches de collecte.

Le point d'entrée de cette couche est le Script Manager, qui est responsable de créer ou détruire une instance d'exécution d'une tâche de collecte, selon l'ordre envoyé par le Privacy Manager. Dans ce contexte, l'objectif clé est de fournir un environnement d'exécution sécurisé, permettant également aux participants de contrôler les données qu'ils partagent pour des raisons de confidentialité. Cependant, généralement tous les accès aux données brutes des capteurs peuvent constituer une menace pour la

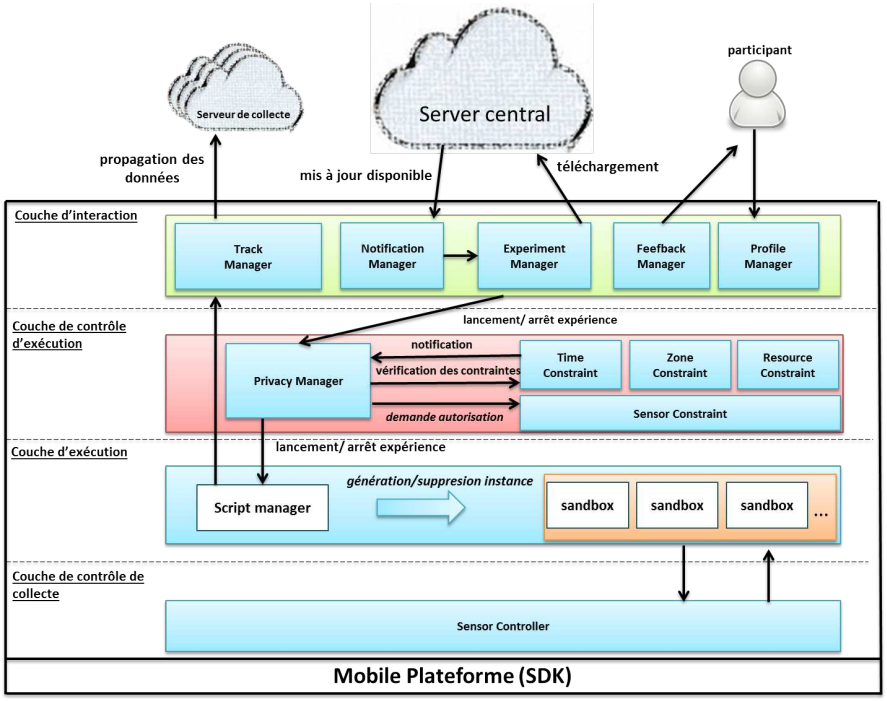

FIGURE 10 – Architecture de l'intergiciel mobile

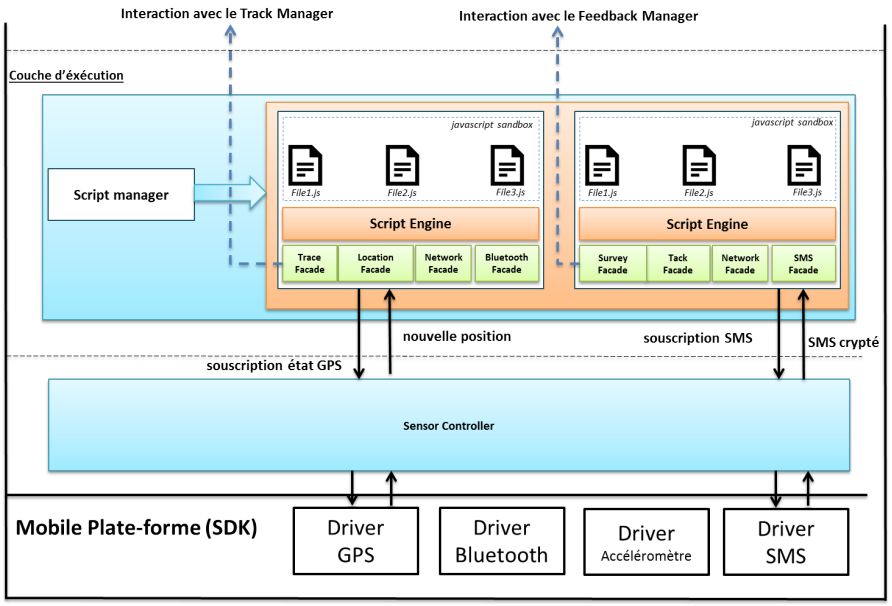

FIGURE 11 – Interaction de la couche d'exécution

vie privée des participants. Par exemple, l'accès au microphone peut être utilisé pour enregistrer une conversation téléphonique souhaitant rester privée. L'accès continu au GPS peut permettre à un utilisateur anonyme d'être identifié juste en couplant des informations géographiques et temporelles [26]. Néanmoins, ce risque reste inhérent dans le domaine de la collecte de données. En effet, si nous voulons une protection parfaite, tous les accès aux capteurs devraient être bloqués.

Pour mitiger ces risques, nous laissons la possibilité aux participants de contrôler les capteurs accessibles par les tâches de collecte (*cf.* 3.4.2). Pour la gestion de ce contrôle d'accès, chaque instance d'exécution d'une tâche de collecte est exécutée dans un bac à sable dédié (*sandbox* en anglais), permettant d'intercepter tous les appels entre les tâches de collecte et les capteurs du dispositif. Ainsi, toutes les fonctionnalités de l'interface de programmation sont fournies par un ensemble de façades (*cf.* sous-section 3.3.1), représentant également le point d'extension de notre l'architecture. Une façade est un objet, écrit en code natif de la plate-forme mobile, permettant l'interaction entre les tâches de collecte et l'environnement mobile. Le listing 3.3 montre l'exemple d'un squelette d'implémentation sous Android, de la façade permettant d'accéder aux capteurs de localisation. Dans cette façade, la fonction `getReferenceName` retourne le nom de la variable qui est utilisée pour accéder aux fonctions fournies par la façade. Chaque fonction annotée par une annotation `@ScriptMethod`, indique au moteur de script que la fonction est accessible par les tâches de collecte. Si nous reprenons l'exemple illustré dans la section précédente (*cf.* sous-section 3.3), la fonctionnalité `$location.onLocationChanged` accessible par la tâche de collecte, est implémentée par la façade `LocationFacade`, injectée en tant que variable `$location` et implémentant la fonction `onLocationChanged`.

```
1  public class LocationFacade extends AbstractAndroidFacade{
2
3    public String getReferenceName() {return "location";}
4
5    public String getSensorDescription(){ ...
6
7    @ScriptMethod
8    public ISubscriber onLocationChanged(
9          JSONObject params,IScriptFunction callback,String... filters){
10         // subscribe to location sensor update
11         return ...
12   }
13
14   public void shutdown() { ... // clean sensor subscription }
15 }
```

Listing 3.3 – Squelette de l'implémentation d'une façade sur Android

Lors de la création d'une nouvelle instance d'exécution, le Script Manager effectue une requête au Privacy Manager pour lister l'ensemble des capteurs admis par le participant. Le Script Manager injectera alors seulement les façades correspondantes aux capteurs qui auront été admis par le participant. Dans ce cadre, les façades correspondant aux capteurs non autorisés ne seront pas injectées, empêchant les scripts de bénéficier des fonctionnalités du capteur en question.

Réguler l'accès aux capteurs

Pour interagir avec les capteurs du dispositif mobile, les façades doivent interagir avec le Sensor Controller. Son objectif est principalement de réguler l'accès aux capteurs du dispositif et de filtrer les données sensibles. Par exemple, lorsqu'une façade demande l'accès à des informations de type SMS ou numéro de téléphone, les données retournées par la couche de contrôle seront automatiquement cryptées pour protéger la vie privée des participants.

Deux types de communications peuvent être effectuées avec le Sensor Controller, des communications de type asynchrone permettant la lecture continuelle de l'état d'un capteur et de type synchrone afin de lire directement l'état d'un capteur. Pour économiser la batterie du dispositif mobile, le Sensor Controller dispose de deux mécanismes. Le premier est un mécanisme de cache, dédié aux communications synchrones. Ce mécanisme permet principalement de limiter une lecture excessive de l'état d'un capteur. Ce mécanisme consiste principalement à définir une fenêtre temporelle dans laquelle seront insérées toutes les données récupérées par le Sensor Controller via les APIs des systèmes mobiles, donnant accès aux données fournis par les capteurs physiques. Avant de procéder à un nouvel accès aux capteurs, la couche de contrôle vérifiera dans un premier temps si la donnée demandée est disponible dans la fenêtre temporelle. Si elle est disponible, la valeur du cache sera alors renvoyée.

Le deuxième mécanisme mis en place tente de minimiser le nombre de souscriptions effectuées aux capteurs des dispositifs. Typiquement, un script peut être notifié du changement d'état d'un capteur en enregistrant à une façade une fonction de rappel, une expression logique permettant de filtrer les états ainsi qu'une période indiquant la fréquence des mises à jour des données. Dans le cas où plusieurs scripts procèdent

à la souscription d'un même capteur avec une période différente, le Sensor Controller effectuera seulement une souscription aux capteurs correspondants, avec une période correspondante à la plus petite période des deux souscriptions. Prenons le cas par exemple où deux scripts effectuent une souscription au capteur GPS avec les périodes respectives de 10 et 20 secondes. Dans ce cas, le Sensor Controller effectuera une seule souscription au capteur GPS avec une fréquence d'échantillonnage égale à 10 secondes, et notifiera les deux scripts toutes les 10 secondes de la nouvelle position.

3.4.2 *Couche de contrôle*

Cette couche est responsable du contrôle de l'exécution des scripts de collecte. Le contrôle consiste à automatiquement démarrer ou arrêter un script de collecte, à partir d'un ensemble de contraintes contextuelles définies par les participants et les expériences. Ces contraintes peuvent par exemple permettre aux participants d'empêcher l'exécution d'un script en dehors de la période allant de 9 heures et 18 heures ou lorsque le participant est en dehors d'une zone géographique. Un autre scénario envisagé est d'empêcher l'exécution d'un script si le niveau de la batterie est inférieur à 30 %. Ces contraintes peuvent être classifiées en quatre catégories : *contrainte temporelle*, *contrainte géographique*, *contrainte de ressource* et *contrainte de capteur*. Ces contraintes sont respectivement interprétées par les modules TimeConstraint, ZoneContraint, ResourceConstraint et SensorConstraint.

Contrainte temporelle

Ces contraintes, interprétées par le module Time Constraint, permettent de spécifier différentes périodes de la journée durant lesquelles les expériences ont l'autorisation d'être exécutées. Ces contraintes sont modélisées comme un ensemble de périodes où chaque période est définie par un temps de départ et un temps de fin. Pour une expérience donnée, la contrainte temporelle est vérifiée si, à instant t, t appartient au moins à une période définie par le participant et à une période définie par le contexte d'exécution de l'expérience.

La surveillance contextuelle des contraintes temporelle consiste à surveiller l'horloge interne du dispositif mobile. Pour chaque contrainte définie, ce module programme une alarme, demandant au système d'être notifié au début et à la fin de la période définissant la contrainte.

Contrainte géographique

Interprétées par le module Zone Constraint, les contraintes géographiques permettent de spécifier un ensemble de lieux ou de régions, dans lesquelles les scripts peuvent être exécutés selon la position du participant. Une contrainte géographique est modélisée par un ensemble de régions circulaires en terme de latitude, longitude et un rayon en kilomètre. Pour une expérience donnée, la contrainte géographique est vérifiée si à une position p donnée, p est inclus au moins dans une zone définie par le participant ou par le contexte d'exécution de l'expérience.

En ce qui concerne la surveillance contextuelle, surveiller activement la position de l'utilisateur peut engendrer une grande consommation énergétique du dispositif. Pour diminuer cette consommation énergétique, nous utilisons un algorithme fréquemment utilisé [3, 8] dans la littérature. Typiquement, l'idée principale est i) d'utiliser alternativement les différents mécanismes permettant de déterminer la position d'un utilisateur — entre le GPS fournissant une grande précision, mais consommant beaucoup d'énergie, et le réseau qui au contraire consomme moins d'énergie, mais est beaucoup moins précis —, ii) et la fréquence d'échantillonnage en fonction de la distance de l'utilisateur et la zone la plus proche de sa position actuelle.

Contrainte de ressource

Ces contraintes permettent de limiter les ressources consommées par l'application. Le participant peut par exemple définir un seuil de batterie pour stopper l'exécution de toutes les expériences de collecte pour éviter un épuisement complet de sa batterie. Une autre ressource qui peut être contrôlée est la quantité de données échangées via une connexion de données mobiles qui peut induire un coût monétaire pour le participant. Dans ce cadre, le participant peut définir un seuil maximum de données transmises lorsqu'une connexion est utilisée.

Contrainte de capteur

Ces contraintes permettent de spécifier les fonctionnalités permises par les scripts de collecte. Dans l'interface de programmation proposée pour définir une expérience de collecte, nous avons vu que les fonctionnalités accessibles par les scripts se font par l'intermédiaire de façades. Pour chaque façade, le participant a la possibilité de définir s'il autorise l'accès à l'ensemble des fonctionnalités proposées par la façade. Par exemple, le participant peut enlever l'autorisation de la façade `$sms`, qui supprimera la possibilité de recueillir des informations relatives aux SMS envoyés et lus.

Dans l'architecture présentée précédemment en figure 10, les modules TimeConstraint, ZoneConstraint et RessourceConstraint ont pour rôle de surveiller le contexte du dispositif (e.g. horloge interne, position, niveau de batterie) et d'envoyer un message au Privacy Manager afin de notifier un changement de contexte du dispositif mobile. Lors de la notification, le Privacy Manager vérifie pour chaque expérience si les contraintes spécifiées par le participant et par l'expérience sont respectées. Dans ce cas, le Privacy Manager interprètera les contraintes de capteurs définies par le module Sensor Constraint pour générer une nouvelle instance d'exécution ou stopper l'exécution d'une expérience de collecte.

3.4.3 *Couche d'interaction*

La dernière couche présenté dans cette section représente la couche de plus haut niveau de l'intergiciel. Cette couche est principalement responsable des interactions entre le dispositif mobile avec les différentes entités logicielles de notre architecture et le participant. Typiquement, cinq types d'interactions sont nécessaires au déroulement des expériences de collecte : i) l'enregistrement auprès du serveur central, ii) la souscription à une expérience de collecte, iii) la mise à jour des expériences, iv) l'interaction avec les participants et finalement v) la propagation des données collectées. Dans la suite de cette sous-section, nous présentons ces interactions et les modules responsables de leurs fonctionnements.

Souscription du dispositif

Le processus d'enregistrement, géré par le Profile Manager, permet d'informer le serveur central de la disponibilité d'un nouveau périphérique mobile pour l'exécution de nouvelles tâches de collecte. Typiquement, cette phase d'enregistrement consiste à fournir des informations générales du dispositif et du participant, permettant de proposer des expériences de collecte adéquates. Dans ce contexte, deux sortes d'informations sont principalement échangées. Les premières sont des informations relatives au dispositif mobile incluant le type du dispositif (*par ex.* smartphone, tablette), son système d'exploitation (*par ex.* Android, iOS), ses performances ainsi que les capteurs embarqués (*par ex.* température, GPS). Pour raffiner la proposition des expériences, les participants ont la possibilité de compléter ces informations volontairement en indiquant les différents lieux de vie dans lequel ils évoluent (*par ex.* maison, travail) son statut (*par ex.* professionnel) ainsi que son âge.Toutes les informations transmises

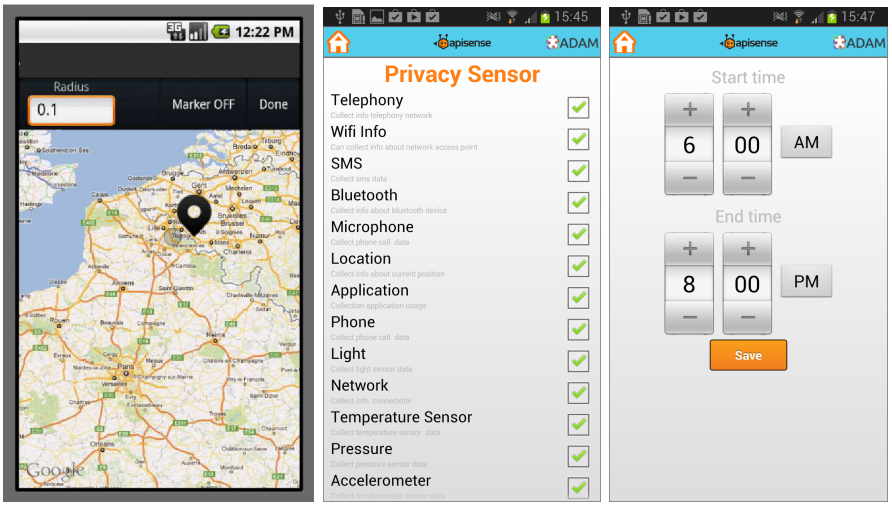

FIGURE 12 – Règles de confidentialité

resteront confidentielles, c'est-à-dire qu'elles ne seront pas directement accessibles par les nœuds de collecte du système. Ces informations seront utilisées uniquement par le serveur central lors de la phase du déploiement des expériences de collecte.

D'autre part, afin de protéger la vie privée des participants, le Profile Manager permet aux participants de définir certaines règles de confidentialités (*cf.* figure 12). Principalement trois catégories de règles peuvent être définies. Les deux premières permettent de spécifier des zones géographiques et temporelles, indiquant à l'agent mobile une interdiction d'exécuter des tâches de collecte. La dernière catégorie permet de spécifier des règles d'autorisations, permettant d'empêcher l'activation des capteurs ou l'accès aux données brutes du capteur si l'utilisateur ne veut pas partager une information. Nous aborderons plus en détails ce point dans la sous-section suivante (*cf.* sous-section 3.4.2).

Souscription à une expérience

Une fois le dispositif enregistré, les participants ont la possibilité de lister (via le ExperimentManager), un sous ensemble d'expériences de collecte proposées par les utilisateurs. Les expériences sont retournées en fonction des informations fournies lors de la phase d'enregistrement. Chaque expérience est définie tout d'abord par un nom ainsi qu'une brève description indiquant l'objectif de la collecte de données. La description d'une expérience est également caractérisée par un contexte d'exécution définissant quand l'expérience doit être exécutée. Le contexte d'exécution est défini

selon une dimension géographique (*par ex.* à l'intérieur d'une zone) et une dimension temporelle (*par ex.* période dans la journée). D'autre part, des informations relatives aux capteurs utilisés sont fournies pour indiquer aux participants quels sont les types de données qui peuvent être collectées sur leurs dispositifs.

À partir de l'ensemble des descriptions disponibles, le participant a la possibilité de soumettre une souscription à une expérience de collecte. Dans ce cadre, la souscription déclenchera le téléchargement des scripts associés dans le système de fichier du dispositif mobile ainsi que des informations complémentaires propres aux scripts de collecte. Ces informations comprennent la version actuelle des scripts de collecte, l'adresse du serveur de collecte ayant initié l'expérience ainsi qu'un identifiant anonyme généré lors de la souscription. L'identifiant anonyme généré par le serveur central est unique pour chaque expérience. Cette identifiant servira alors de clé d'authentification lors de la propagation des données vers le serveur de collecte. Lorsque les scripts de l'expérience sont téléchargés, le module déclenche automatiquement son exécution. Néanmoins le participant garde à tout moment la possibilité d'arrêter son exécution.

Mise à jour des expériences

La mise à jour des expériences de collecte est gérée par le module NotificationManager. Les mise à jour sont effectuées par des communications de type *push*, c'est-à-dire des communications serveur vers mobile. Dans notre contexte applicatif, cela permet d'éviter de faire des requêtes périodiques vers le serveur, induisant un surcoût énergétique pour assurer une rapide prolifération. Pour assurer ce type de communication, nous utilisons des services intermédiaires proposés par les plate-formes mobiles actuelles, telles que APNs (*Apple Push Notification service*) pour plateformes tournant sur iOS et GCM (*Google Cloud Messaging*) pour les plateformes Android. Généralement, pour utiliser ce type de service, les dispositifs mobiles doivent tout d'abord procéder à une phase d'enregistrement auprès du service en question. Cette phase permet l'obtention d'un identifiant unique, qui devra ensuite être partagé avec le serveur — dans notre contexte, le serveur central— voulant envoyer des messages vers le dispositif en question. Le serveur peut ensuite faire appel au service dédié en utilisant les identifiants partagés pour propager directement des informations auprès des dispositifs mobiles. Dans ce contexte, lorsqu'une nouvelle version d'une expérience est déployée, le serveur central envoie un message à tous les participants ayant souscrits à l'expérience en question. Ce message est ensuite intercepté par le NotificationManager qui déclenchera automati-

quement le téléchargement de la nouvelle version, sans imposer aux participants de télécharger à nouveau la tâche de collecte manuellement.

Retour utilisateur

Le Feedback Manager est responsable de l'interaction entre les tâches de collecte et les participants. Comme nous l'avons mentionné précédemment (*cf.* sous-section 3.3.3, page 66), l'interface de programmation proposée permet de diffuser des visualisations aux participants. Ces visualisations peuvent consister à fournir des statistiques sur les données collectées sous forme de graphique ou de carte, leur demander de prendre une photo d'un évènement particulier ou répondre à un questionnaire. Ces visualisations peuvent être directement développées en HTML/JavaScript, fournissant une grande flexibilité pour définir leurs propres visualisations en fonction de la nature de leurs expériences. D'autre part, cela permet également d'être indépendant de la plate-forme d'exécution, évitant ainsi le développement de plusieurs visualisations selon les plate-formes visées. Dans ce cadre, nous tirons bénéfice des modules fournis par les plate-formes mobiles (*par ex.* WebView [2] pour Android, UIWebView [3] pour iOS) pour faciliter l'intégration de contenus web dans une application mobile.

Dans ce contexte, les tâches de collecte peuvent interagir avec le FeedBackManager, via une façade dédiée, permettant de publier un nouveau contenu accessible par le participant. Chaque contenu web est ensuite sérialisé et sauvegardé dans une base de données, et pourra être visualisé par le participant à tout moment à partir d'une interface graphique fournie par le FeedBackManager, (*cf.* figure 13).

Propagation des données

La propagation des données vers les nœuds de collecte est gérée par le TrackManager. Lors de l'exécution d'une tâche de collecte, les données collectées sont transmises au TrackManager qui les sauvegarde dans la base de donnée locale du dispositif. Pour chaque expérience, le processus de propagation (*cf.* sous-section 3.3.3) consiste à compresser les données collectées, ajouter des méta-informations sur la configuration du dispositif du participant (*c.-à-d.* filtre de vie privée) et à faire appel au service dédié du serveur de collecte ayant initié l'expérience pour l'envoi des données. Pour des raisons de sécurité, l'envoi des données s'effectue en utilisant le protocole HTTPS pour le cryptage des données.

2. Android WebView : http://tinyurl.com/android-webview
3. iOS UIWebView : http://tinyurl.com/ios-webview

FIGURE 13 – Exemple de contenu web disponible via le FeedbackManager

Nous venons de présenter dans cette section l'intergiciel mobile responsable de l'exécution de tâche de collecte, développée avec l'interface de programmation présentée dans la première section de chapitre. Dans la section suivante, nous présentons l'évaluation d'un prototype implémentant les concepts proposés dans ce chapitre.

3.5 ÉVALUATION DU PROTOTYPE

Afin d'implémenter les concepts présentés dans ce chapitre sur une plate-forme mobile spécifique, celle-ci doit permettre principalement *i)* d'effectuer des processus en tâche de fond, *ii)* permettre la lecture en continue des capteurs, *iii)* et finalement être capable d'intégrer un moteur de script dans une application. Lorsque nous avons commencé le développement de notre prototype, la plate-forme Android correspondait parfaitement à ces caractéristiques, contrairement aux systèmes d'exploitation iOS qui ne permettaient pas d'exécuter des processus en tâche de fond limitant ainsi le développement d'application de collecte opportuniste. Cependant, les récentes mise à jour de ce système d'exploitation fournissent désormais ce support, rendant portable notre système pour iOS. Par manque de temps, nous n'avons pas pu fournir un prototype complet sur ce système d'exploitation, mais nous avons fourni une preuve de faisabilité disponible en open source [4].

Le prototype Android proposé a été conçu pour être utilisé comme application autonome ou comme librairie. En tant qu'application autonome, l'application peut être

4. https://github.com/APISENSE/APISENSE-SDK-iOS

téléchargée par des participants voulant évoluer dans le système en flashant un *QR code* publié sur le site internet APISENSE® [5].

Plus particulièrement, le prototype réalisé à été développé en utilisant la version 2.3 du SDK proposé par Android. L'ensemble des modules proposés par l'architecture est inclus dans un *Service* de haute priorité pour prévenir que le système ne supprime notre application lorsqu'il a besoin de mémoire. Pour l'exécution des scripts, nous utilisons Rhino [6], un moteur d'exécution JavaScript pour Java, qui permet l'interopérabilité des deux langages. L'ensemble du code réalisé représente approximativement 30000 lignes de codes, incluant l'interface graphique et l'architecture de l'application et les façades développées.

Dans la suite cette section, nous présentons plusieurs cas réels d'expériences, montrant la diversité des tâches de collecte qui peuvent être réalisées à l'aide notre interface de programmation. Nous comparons par la suite son expressivité avec les différentes approches proposées dans la littérature, et discutons du coût énergétique lié à l'exécution des tâches de collecte sur les dispositifs mobiles.

3.5.1 *Quelques exemples de collectes*

Pour démontrer l'expressivité de l'interface de programmation proposée dans ce chapitre, nous avons implémenté quatre tâches de collecte différentes (*cf.* table 7). Chaque tâche de collecte est inspirée de l'état de l'art, et démontre différents aspects de APISENSE.

Wifi/Blutooth Scanner

Wifi/Blutooth Scanner est une application opportuniste, montrant une interaction avec les capteurs Wi-Fi et Bluetooth d'un dispositif. Le listing 3.4 illustre le script implémentant cette application. L'objectif du script est de collecter, toutes les minutes, les points d'accès WIFI et les périphériques Bluetooth d'un utilisateur mobile. Dans le script, nous enregistrons une fonction de rappel aux façades `$wifi` et `$bluetooth` responsables des interactions avec ces capteurs, et nous configurons les capteurs pour qu'ils effectuent une nouvelle recherche toutes les minutes (`period:"1 min"`). La

5. http://www.apisense.fr
6. http://www.mozilla.org/rhino

Application	Capteurs	Description
Wifi-Blutooth Scanner	Réseaux	Application opportuniste. Collecte périodiquement les périphériques Bluetooth et Wi-Fi.
Citoyen Journaliste	GPS + Questionnaire	Application participative. Demande à l'utilisateur de prendre une photo lorsqu'il se situe dans une zone géographique.
Qualité Réseau	Réseau + Questionnaire + Téléphone	Combinaison d'une application opportuniste et participatif.pour corréler la qualité d'une communication en fonction de la qualité du signal GSM.
Exploiter des capteurs externes	Bluetooth	Communication avec un robot Lego Mindstorms NXT
Inférence Conxtuelle	Accéléromètre + Questionnaire	Intégration d'une machine d'apprentissage pour inférer l'activité d'un utilisateur.

TABLE 7 – Exemples de tâches de collecte implémentées avec APISENSE. (*Lignes de code)

fonction de rappel enregistrée collecte alors les résultats donnés par les capteurs pour les propager vers le serveur (`$trace.add(event)`).

```
1 $wifi.onScan({period : "1 min"},function(event){
2       $trace.add(event)
3 })
4 $bluetooth.onScan({period : "1 min"}, function(event){
5       $trace.add(event)
6 })
```

Listing 3.4 – Tâche de collecte : Wifi/Bluetooth Scanner

Nous décrivons dans l'exemple suivant un exemple de collecte participative.

Le Citoyen Journaliste

L'objectif de cette application est de montrer comment une tâche de collecte peut interagir avec un utilisateur. Le Citoyen Journaliste, présenté initialement en [29], demande aux utilisateurs mobiles de prendre une image dans une région géographique spécifique, et de la commenter. Le script présenté dans le listing 3.5, crée d'abord un questionnaire (ligne 1-3). En ligne 5 nous observons la position de l'utilisateur et définissons que la fonction enregistrée (ligne 6-8) doit être exécutée uniquement lorsque la position de l'utilisateur se trouve dans le polygone précis (ligne 8). La fonction enregistrée publie le questionnaire à l'utilisateur (ligne 6), et collecte les données pour les propager vers le serveur lorsque l'utilisateur aura répondu au questionnaire.

```
1  var survey = $survey.create();
2  survey.image("Take a picture");
3  survey.open("Add a comment");
4
5  $location.onLocationChanged(function(event){
6    $survey.publish(survey,function(content){
7      $trace.add(content);
8      $task.finish();
9    });
10 },"&((lat >= 0)(lat <= 1)(lon >= 0)(lon <= 1))")
```

Listing 3.5 – Tâche de collecte : Le Citoyen Journalist

Qualité Réseau

Dans cette application, inspirée de MyExperience [25], nous voulons étudier la corrélation entre la qualité du signal réseau et la qualité de la communication vocale. Dans cette application (*cf.* listing 3.6), nous voulons montrer deux nouvelles propriétés de notre interface de programmation qui sont i) sa capacité à pourvoir adapter la séquence des questions d'un questionnaire en fonction des premières réponses d'un utilisateur et ii) combiner une collecte opportuniste et participative

Dans la première partie du script (ligne 1-6), nous collectons la force du signal réseau GSM lorsque l'utilisateur est en situation de mobilité. Afin d'effectuer un retour sur les données collectées, nous insérons un nouveau point dans la visualisation proposée (ligne 6).

Dans la deuxième partie du script (ligne 8-14), nous définissons un questionnaire et ajoutons deux questions relatives à la qualité d'une communication vocale. La première question consiste à demander la qualité de sa communication vocale. La deuxième consiste à lui demander pourquoi le participant a considéré que la communication a été mauvaise. Nous définissons ensuite l'enchaînement des questions (ligne 15-18) qui consiste à proposer uniquement la deuxième question uniquement si l'utilisateur a répondu une réponse spécifique à la première question. Ensuite, nous proposons le questionnaire lorsque l'utilisateur a terminé une communication vocale.

```
1  $location.onLocationChanged(function(event){
2        $trace.add({
3              latlng : [event.latitude,event.longitute],
4              signal : $telephony.signalStrength()
5        })
6  })
7
```

```
8  var qualitySurvey = $survey.create()
9  qualitySurvey.close("q1",
10       "Quelle a été la qualité de votre communication ?",
11       ["Trés Mauvaise", "Mauvaise", "Moyenne", "Bonne", "Excellente"])
12 qualitySurvey.close("q2",
13       "Quel a été le probléme ?",
14       ["écho", "grésillement", "voix coupé"])
15 qualitySurvey.then("q1",function(response){
16       if (response == "Trés Mauvaise")
17            return "q2"
18 })
19
20 $phone.onCallFinish(function(event){
21 $survey.publish(qualitySurvey, function(content){
22       $trace.add({
23            latlng : $location.lastLocation(),
24            signal : $telephony.signalStrength(),
25            content : content
26       })
27 })}
```

Listing 3.6 – Exemple d'application participative

Exploiter des capteurs externes

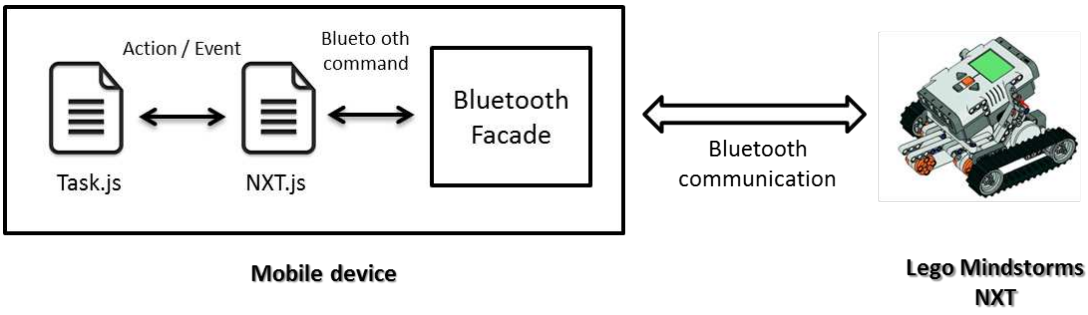

FIGURE 14 – Flux de données de l'expérience Manipulation de Lego NXT

Dans cette expérience, nous montrons que notre interface de programmation peut également être utilisée pour interagir avec des capteurs externes, afin de collecter des données qui ne peuvent pas être initialement capturées avec les capteurs initialement inclus dans les dispositifs mobiles. Cette expérience exploite les capteurs intégrés d'un robot Lego Mindstorms NXT (*c.-à-d.* capteur de contact, d'ultrason, de température et de pression atmosphérique), et permet également d'interagir avec le moteur du robot afin d'en prendre le contrôle à partir de la tâche de collecte.

Comme le montre la figure 14, l'expérience est composée de deux scripts : `NXT.js` et `Task.js` décris par le listing 3.7. Le rôle du premier script est de faire une abstraction avec les différentes fonctionnalités des robots NXT. Ce script communique avec la façade responsable de l'interface Bluetooth, pour établir une connexion avec le robot et le manipuler à distance. Ce script peut être ensuite intégré dans le script `Task.js` en tant que librairie (ligne 1). Dans cet exemple, nous établissons une connexion avec un robot NXT à proximité (ligne 8), et déclenchons les moteurs du robot pour le faire avancer. Nous réagissons également lorsque le robot rencontre un obstacle pour le faire changer de direction (ligne 11-13). Les lignes (15-22) permettent ensuite de collecter des données capturées par les capteurs du dispositif mobile (*par ex.* GPS) et des capteurs du robot NXT (*c.-à-d.* pression atmosphérique et température) lorsqu'il se déplace.

```
1  var nxt = require("NXT.js");
2
3  var change_direction = function(){
4        nxt.move_back()
5        nxt.turn_left({ 'angle' : 45 * random(), 'duration' : '2s'})
6        nxt.move_forward()
7  }
8  nxt.onConnect({'timeout': '30s'},function(){
9        nxt.move_forward()
10
11       nxt.onTouch(function(){
12             change_direction()
13       })
14
15       $location.onLocationChanged({provider : ["network"]},function(event){
16             $trace.add({
17                   latlng : [event.latitude,event.longitude],
18                   pressure : nxt.pressure(),
19                   temperature : nxt.temperature()
20             })
21       })
22 })
```

Listing 3.7 – Tâche de collecte : Manipulation de Lego NXT

Inférence Contextuelle

Dans cette dernière expérience, l'objectif est de montrer que APISENSE® peut également être utilisé pour rapidement prototype et valider empiriquement des algorithmes contextuels. Dans ce scénario, nous voulons mettre en place un modèle d'apprentissage permettant de reconnaître l'activité d'un utilisateur (*c.-à-d.* assis, debout, marche, courir, monter ou descendre des escaliers). L'idée générale de l'approche est d'observer

les données transmises par le capteur d'accélération afin de déterminer l'activité de l'utilisateur. Le script décrit dans le listing 3.8 implémente l'algorithme initialement proposé en [37].

```
var classes = ["walk","jog","stand", "sit", "up", "down"];
var current = 0; var buffer = new Array();
var model = weka.newModel(["avrX","avrY",...], classes);
var filter = "|(dx>"+delta+")(dy>"+delta+")(dz>"+delta+")";

var input = $accelerometer.onChange(filter,
   function(acc) { buffer.push(acc) });

var learn = time.schedule({ period: '5s' }, function(t) {
  if (model.learn(classes[current]) >= threshold) {
    current++;
  }
  if (current < classes.length) { // Learning phase
    input.suspend();
    var output = $dialog.display(
      { message: "Select movement", spinner: classes });
    model.record(attributes(buffer), output);
    sleep('2s');
    buffer = new Array();
    input.resume();
  } else { // Exploitation phase
    dialog.display({message: "Learning phase completed"});
    learn.cancel();
    model.setClassifier("NAIVE_BAYES");
    time.schedule({ period: '5s' }, function(t) {
      trace.add({
        position: model.evaluate(attributes(buffer)),
          stats: model.statistics() });
      buffer = new Array();
} } });
```

Listing 3.8 – Tâche de collecte : Inférence Contextuelle.

Le script se décompose en deux phases : i) une phase d'apprentissage et ii) une phase d'exploitation. Pour réaliser la phase d'apprentissage, nous avons intégré un algorithme d'apprentissage [44] implémenté en Java dans l'intergiciel mobile, et nous avons ajouté une façade pour utiliser ses fonctionnalités à partir du script. Durant la phase d'apprentissage, le script génère une boite de dialogue demandant à l'utilisateur de répéter des mouvements spécifiques. Lorsque l'utilisateur exécute les mouvements demandés, nous enregistrons les données d'accélérations selon les trois axes, et calculons des métriques (*c.-à-d.* moyenne, l'écart type, etc.) sur les valeurs obtenues. Ces métriques sont ensuite insérées dans l'algorithme d'apprentissage et le mouvement qui à été effectué. Une fois que suffisamment de données ont été insérées dans le modèle d'apprentissage, le

	Predicted Class						Acc (%)
	Walk	Jog	Stand	Sit	Up	Down	
Walk	**66**	0	4	0	0	0	94,3
Jog	0	**21**	0	0	0	0	100
Stand	4	0	**40**	0	0	0	90,9
Sit	0	0	2	**83**	0	0	97,6
Up stair	0	0	0	0	**22**	0	100
Down stair	0	0	0	0	0	**11**	100

TABLE 8 – Tableau décrivant la qualité du système de classification

script passe en phase d'exploitation. Dans cette phase, nous évaluons, toutes les cinq secondes, les métriques des données d'accélération dans le modèle d'apprentissage. Le résultat correspondant au mouvement de l'utilisateur et ensuite sauvegardé avec les statistiques du modèle obtenu pour les reporté sur le serveur.

Le tableau 8 reporte la matrice de confusion décrivant la qualité du système de classification obtenue par un utilisateur durant cette expérience. La matrice illustre le nombre de mouvements effectués durant cette expérience, et la haute précision des prédictions obtenues pour chacun des mouvements.

Discussion

Dans cette sous-section, nous discutons de l'expressivité et la complexité des approches décrites dans l'état de l'art avec APISENSE®. Nous utilisons le nombre de lignes de code pour comparer la complexité avec deux applications de collecte couvrant les deux formes d'acquisition de données qui sont la collecte opportuniste (application Wifi-Bluetooth Scanner) et participative (application Citoyen Journaliste). Le tableau 9 présente le nombre de lignes de code nécessaires pour développer ces applications. Le symbole `X` signifie que l'application ne peut pas être exprimée et le symbole `?` signifie que l'application est réalisable, mais que nous n'avons pas d'information sur le nombre de lignes de code nécessaire avec l'approche concernée.

En terme de lignes de code, APISENSE® est beaucoup plus concis (4 lignes) que les approches utilisant le langage déclaratif XML pour concevoir une application. C'est la cas avec MyExperience (27 lignes) et Medusa (45 lignes) pour l'application Citoyen Journaliste. De plus ces deux dernières approches ne supportent la définition d'application de collecte purement opportuniste comme l'application Wifi-Blutooth

Scanner. Même si la concision de notre code est à peu près similaire à Anonysense (7 lignes de code) et Pogo (6 lignes de code), notre approche comporte de nombreux avantages comparés à ces approches. Par apport à Anonysense, qui utilise un DSL pour le développement des applications, le langage JavaScript nous permet d'avoir une plus grande flexibilité pour concevoir des traitements plus complexes, et surtout nous permet de facilement réutiliser du code déjà existant, comme nous l'avons montré dans l'application présentée en sous-section (*cf.* 3.5.1). Par rapport à Pogo, utilisant également une interface de programmation basée sur JavaScript, notre interface de programmation supporte les tâches de collecte de données participative qui n'est pas possible avec l'interface proposée par Pogo. La seule approche proposée supportant la collecte opportuniste et participative est PRISM, qui propose le déploiement d'applications de collecte écrites en code natif. Cependant PRISM ne propose pas d'abstraction pour faciliter le développement de ses applications, nécessitant une expertise approfondie en développement mobile. L'application Citoyen Journaliste nécessite par exemple 330 lignes de code. En outre, le développement natif des applications les rend exclusives pour un OS mobile, demandant par conséquent d'étudier et développer une nouvelle version de l'application pour supporter l'hétérogénéité des OS disponible.

Nous avons présenté dans cette section un ensemble d'applications réalisables avec APISENSE®. Nous pensons que APISENSE® propose un bon compromis entre toutes les approches proposée dans l'état de l'art, supportant la collecte opportuniste (application Wifi/Blutooth Scanner), participative (application Citoyen Journaliste), ou une combinaison des deux (application Qualité Réseau) tout en fournissant une abstraction complète des technologies mobiles. En proposant une interface de programmation au-dessus de JavaScript, APISENSE® permet également de définir des traitements complexes (application Inférence contextuelle), et faciliter la réutilisation de code déjà existant (application Exploiter des capteurs externes). Nous pensons que la réutilisation est un aspect fondamental pour le succès APISENSE®, favorisant son utilisation par de nombreux acteurs académiques ou industriels, avec des niveaux d'expertises différents dans la collecte de données. En effet, les experts du domaine peuvent proposer des modules ou algorithmes complexes, utilisant les capteurs des dispositifs pour inférer des informations de haut niveau (*par ex.* activité de l'utilisateur, mode de transport), ou encore utiliser intelligemment les différents capteurs pour réduire la consommation énergétique des applications de collecte dans un contexte spécifique [14]. Ces modules pourront être ensuite réutilisés par des utilisateurs non experts. Et finalement, l'utilisation de JavaScript permet également de supporter l'hétérogénéité des OS mobiles

	Wifi-Blutooth Scanner	Citoyen Journaliste
APISENSE	4	9
Medusa [55]	X	45
MyExperience [25]	X	27
Anonysense [62]	5	X
Pogo [6]	4	X
PRISM [16]	?	330

TABLE 9 – Comparaison de l'expressivité et du nombre de lignes de code

pour s'affranchir d'étudier et développer une nouvelle application pour chaque OS disponible.

3.5.2 *Coût énergétique d'une expérience de collecte*

Dans cette section, nous comparons la consommation énergétique de notre prototype réalisé sur Android avec Funf [1], une application présentée dans l'état de l'art APISENSE® (section 2.3.1, page 34). Nous rappelons que Funf est un outil permettant de générer des applications natives Android à la carte. Pour réaliser cette expérience, nous avons exécuté chaque application durant 24 heures, sur un Nexus-S réinitialisée à une configuration par défaut, pour limiter le bruit qui peut être induit par d'autres applications. Chaque application est configurée pour collecter toutes les 10 minutes des informations de la batterie du dispositif mobile. Le résultat de cette expérience est illustré par la figure 15

Dans cette courbe, l'abscisse représente le temps, et l'ordonnée la tension en millivolt représentant le niveau de la batterie du dispositif. La tension d'un dispositif varie généralement entre 4200 millivolts, représentant 100% de niveau de batterie, et 3200 millivolts. Comparé à une application native (*baseline*), nous pouvons observer que le surcoût énergétique de notre prototype est faible comparé à une application native, et moins important que celui imposé par Funf. Bien que notre solution soit plus consommatrice qu'une application native, notre solution ne nécessite aucune expertise particulière de la plate-forme Android, et rend également transparent le déploiement de l'application et la propagation des données pour les développeurs.

Comme la consommation énergétique dépend fortement de i) la nature de l'expérience, ii) du type de capteur utilisé et iii) du volume des données produit, nous avons réalisé une seconde expérience permettant d'estimer l'impact des différents capteurs

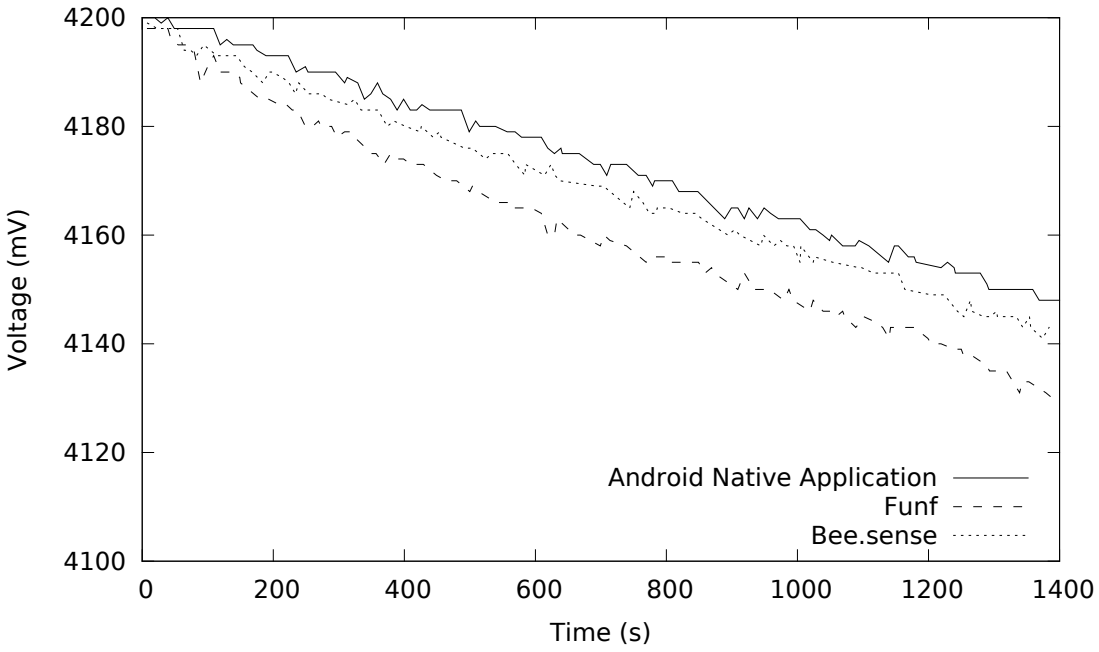

FIGURE 15 – Impact d'APISENSE sur la consommation énergétique

sur la consommation de la batterie (*cf.* figure 16). Pour cette expérience, nous avons développé trois scripts, qui ont été exécutés séparément sur trois dispositifs mobiles. Le premier script (APISENSE + Bluetooth), déclenche un scan toutes les minutes et collecte le niveau de la batterie aussi bien que le résultat du scan. Le second (APISENSE + GPS) collecte également toutes les minutes les données GPS tandis que le dernier script collecte des données relatives au point d'accès WIFI. Cette expérience démontre que même avec un stress important sur les capteurs, il est possible de collecter des données durant une journée normale de travail sans avoir à recharger le dispositif mobile (40% d'énergie consommée après 10 Heures d'activation du GPS).

3.6 CONCLUSION

Le développement d'applications mobiles dédiées à la collecte de données est un processus long et complexe. Généralement, afin de déployer l'application vers un grand nombre d'utilisateurs, plusieurs applications ont besoin d'être développées pour faire face à la diversité des plate-formes mobiles disponibles. D'autre part, ces applications doivent généralement prendre en compte des aspects non fonctionnels tels que la

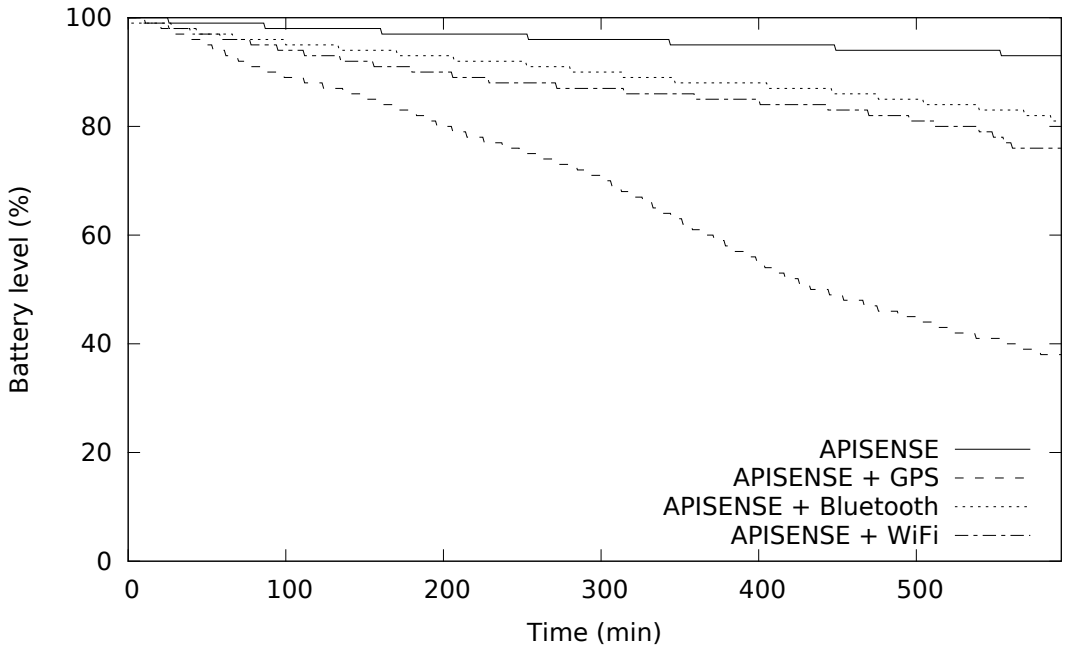

FIGURE 16 – Impact des capteurs sur la consommation énergétique

protection de la vie privée des utilisateurs ainsi que la consommation énergétique de ces applications.

Dans ce chapitre, nous avons présenté notre approche pour diminuer le coût lié aux développements d'applications mobiles dédiées à la collecte de données. Plus particulièrement, nous avons présenté dans un premier temps une interface de programmation de haut niveau pour la définition d'une application de collecte. L'interface proposée a été conçue pour *i)* faire face à la diversité des applications de collecte (i.e. participative, opportuniste), *ii)* supporter l'hétérogénéité des plate-formes mobiles *iii)* tout en permettant de s'abstraire de la complexité liée au développement mobile.

Nous avons ensuite présenté l'environnement mobile dédié aux utilisateurs en charge de l'exécution de l'interface proposée. Nous avons particulièrement présenté son architecture composée de quatre couches pour i) le déploiement et la propagation des données, ii) permettre aux utilisateurs de contrôler des les données collectées sur leurs dispositifs, iii) l'exécution des applications de collecte et iv) limiter la consommation énergétique liée à l'exécution d'une application de collecte.

Dans le chapitre suivant, nous allons présenter l'environnement serveur de notre architecture.

4

COLLECTE RÉPARTIE DE DONNÉES

Sommaire

4.1 INTRODUCTION

L'objectif des travaux de cette thèse est de proposer une solution générique, facilitant le développement et le déploiement de campagnes de collecte de données à travers des dispositifs mobiles. Dans le chapitre précédent, nous avons présenté l'environnement mobile de APISENSE®, la plate-forme résultante des travaux de cette thèse. Nous avons proposé un modèle de programmation facilitant le développement d'applications mobiles dédiées à la collecte de données et un intergiciel déployé au préalable sur les dispositifs mobiles, dédié à leurs exécutions.

Dans ce chapitre, nous présentons notre deuxième contribution concernant l'architecture et l'implémentation de l'environnement serveur de APISENSE®, responsable du déploiement des campagnes de collecte, de la persistance des données ainsi que leurs analyses. Ce chapitre adresse particulièrement deux défis introduit en début de manuscrit, qui sont la généralité de la plate-forme, concernant la capacité de la plate-forme à supporter une grande diversité de campagnes de collecte ainsi que son passage à l'échelle.

STRUCTURE DU CHAPITRE La suite du chapitre est organisée comme suit : en section 4.2, nous présentons une vue d'ensemble de l'architecture proposée, et discutons des bénéfices de cette architecture par rapport aux solutions de l'état de l'art. Dans les sections 4.3 et 4.4, nous présentons en détail l'architecture des différentes entités logicielles de APISENSE®, ainsi que les méthodes de conceptions et de modélisations utilisées. Dans le chapitre 4.5, nous proposons une optimisation améliorant le passage à l'échelle du système avant de conclure en section 4.6.

4.2 INFRASTRUCTURE RÉPARTIE DE TRAITEMENT DES DONNÉES

Avant de présenter en détail l'architecture proposée dans ce chapitre, nous présentons ici une vue globale de la plate-forme APISENSE®. La figure 17 donne un aperçu simplifié des différentes entités logicielles qui la composent, et leurs principales interactions. L'architecture est principalement composée de trois parties distinctes : un ensemble de *nœuds de collecte* (*DataGathering Node*), un *serveur central* (*Central Server*) et un ensemble de *nœuds mobiles* avec une application dédiée installée au préalable par les participants. Nous avons déjà présenté les *noeuds mobiles* dans le chapitre précédent (*cf.* 3.4, page 68).

APISENSE® adopte une architecture décentralisée, où le serveur central est assumé comme le tiers de confiance de la plate-forme. Plus particulièrement, le serveur central est responsable du déploiement des tâches de collecte soumises par les nœuds de collecte, vers les nœuds mobiles du système. Afin de fournir un déploiement adapté (*par ex.* zone géographique, type de dispositif mobile), le serveur central a besoin de maintenir à jour un ensemble d'informations, sur les dispositifs mobiles disponibles pour exécuter des tâches de collecte. Dans ce contexte, nous supposons que le serveur central est géré par un acteur de confiance, qui peut être une organisation publique, ou une structure de recherche.

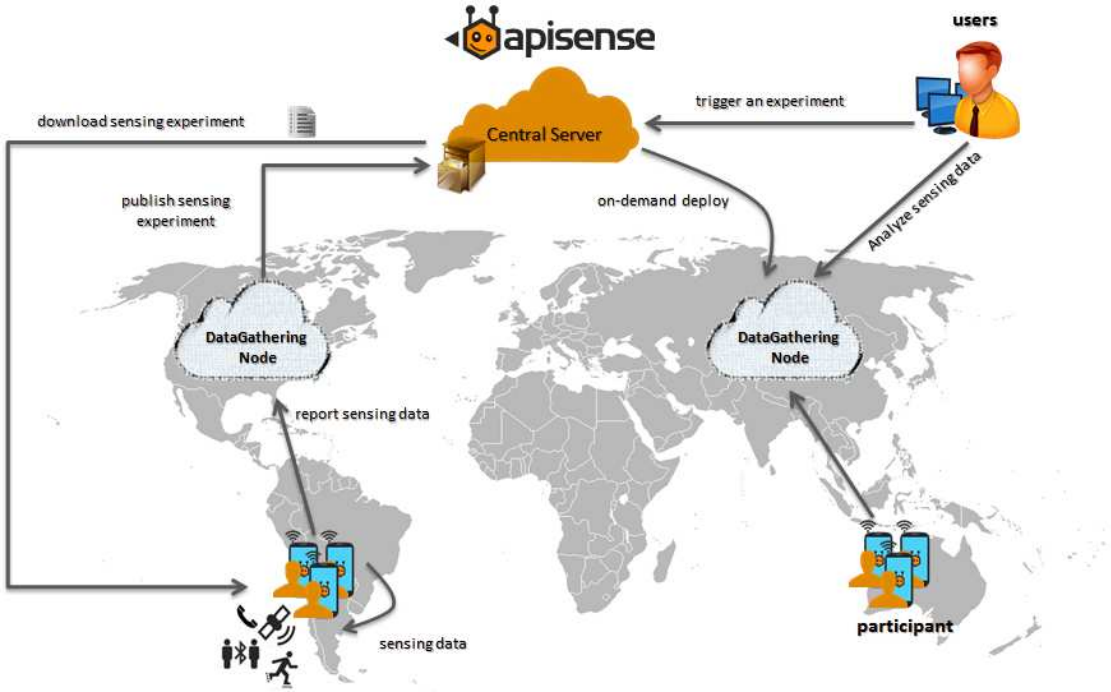

FIGURE 17 – Vue d'ensemble d'APISENSE®

La gestion des campagnes de collecte est assurée par les nœuds de collecte, qui peuvent être dédiés à un ou plusieurs utilisateurs. Typiquement, la gestion d'une campagne de collecte comporte quatre phases : i) définir une tâche de collecte, ii) recruter un sous-ensemble de dispositifs mobiles, iii) assurer la persistances des données collectées (*c.-à-d.* dans une base de données) lors de l'exécution de la tâche iv), exploiter les données. Pour réaliser ces différentes étapes, un nœud de collecte propose un environnement fournissant un ensemble de services qui peuvent être entièrement configurés par les utilisateurs. La configuration consiste en la sélection de modules implémentant la logique dédiée à la réalisation d'une étape d'une campagne de collecte. Dans ce cadre, les modules proposés peuvent implémenter différentes stratégies selon la nature de la campagne. Par exemple, la phase de collecte peut inclure un modèle particulier d'anonymisation des données pour assurer la confidentialité des participants.

Ce choix de conception, séparant les préoccupations du déploiement ainsi que la collecte et l'analyse des données, nous permet de rompre avec le syndrome de l'enfermement propriétaire. Premièrement, en offrant la possibilité aux utilisateurs de déployer leurs nœuds de collecte sur l'infrastructure de leur choix, leur permettant

ainsi de garder un contrôle total sur les données collectées durant leurs campagnes. Les nœuds de collecte peuvent ainsi être déployés vers une infrastructure publique ou privée, en fonction des préoccupations éthiques ou légales en rapport avec le stockage des données. De plus, les nœuds de collecte des utilisateurs sont complètement isolés les uns des autres, améliorant la sécurité et la disponibilité de leurs environnements, permettant d'effectuer des traitements lourds sur les données collectées sans affecter les autres instances. Et finalement, en gardant une entité centrale (*c.-à-d.* le serveur central), cela permet aux nouveaux utilisateurs de bénéficier d'un ensemble de participants déjà disponibles pour exécuter leurs tâches de collecte, s'affranchissant ainsi d'une longue période de recrutement.

Après avoir présenté une vue globale de la plate-forme APISENSE®, nous présentons dans la suite de ce chapitre l'architecture et les services fournis par le serveur central (*cf.* section 4.3), ainsi que les nœuds de collecte (*cf.* section 4.4).

4.3 Architecture du serveur central

Comme nous l'avons présenté dans la section précédente, APISENSE® est une plate-forme répartie dédiée aux déploiements et à l'exécution de campagnes de collectes de données. Afin de fournir une plate-forme modulaire, facilement extensible et configurable, APISENSE® est une plate-forme orientée service, utilisant un modèle à composants logiciels pour modéliser les différentes applications serveur qui la composent. Plus spécifiquement, APISENSE® est basée sur le modèle SCA [46], un standard initié par le consortium OASIS, dédié à la modélisation d'architectures tout en étant indépendant des langages de programmation, des protocoles de communication, des langages de description d'interface et des propriétés non fonctionnelles.

La figure 18 illustre le modèle SCA du serveur central. Dans SCA, les bases des constructions sont les *composants* logiciels, qui peuvent offrir des *services*, ou solliciter d'autres composants via des *références*. Les services ainsi que les références peuvent être connectés à l'aide de fils (*wire* en anglais). SCA spécifie un modèle à composants hiérarchiques, laissant la possibilité d'implémenter un composant par un langage de programmation, ou par un ensemble de sous composants. Dans le dernier cas, ces composants sont appelés *composite*. Afin de supporter l'interaction entre des services distants, SCA spécifie la notion de communication distante (*binding*), permettant de décrire

le protocole que le client devra utiliser pour invoquer un service. Dans l'architecture SCA proposée, les composants ParticipantFrontend, GatheringNodeFrontend ScientistFrontend et représentent les points d'entrés pour les différents acteurs qui interagissent avec le serveur central. Ces composants exposent leurs services en tant que ressource REST, accessibles via le protocole HTTP. Plus particulièrement, ces composants fournissent quatre types services nécessaires au fonctionnement du système, dédié à :

L'enregistrement des participants : ce service (*participant registration*) est accessible par les dispositifs mobiles, permettant l'enregistrement d'un nouveau participant volontaire pour exécuter de nouvelles tâches de collecte.

L'enregistrement des expériences de collecte : ce service (*deploy scripts*) est accessible par les nœuds de collecte, et permet aux utilisateurs d'enregistrer de nouvelles expériences de collecte.

Le déploiement des expériences de collecte : ces services (*list experiment*, *download script*, *recruit participants*) assurent la propagation des expériences vers les terminaux mobiles.

La gestion des nœuds de collecte : ces services (*user registration*, *deploy data gathering node*, *downloads modules*) représentent les points d'entré dans le système pour les utilisateurs voulant créer et configurer un nœud de collecte.

Nous présentons dans la suite de cette section les services fournis par ces composants.

4.3.1 *L'enregistrement des participants*

L'enregistrement des participants permet d'informer le serveur central de la présence et de la disponibilité des participants pour exécuter des tâches de collecte. Ce service, fourni par le composant DeploymentService, est invoqué par les nœuds mobiles lors de leurs installations sur les terminaux mobiles des participants.

L'inscription inclut des informations statiques sur la configuration matérielle des terminaux des participants, ainsi que des informations confidentielles. Les informations matérielles comprennent le type du terminal (*par ex.* smartphone, tablette), le système d'exploitation utilisé (*par ex.* Android, iOS) ainsi que l'ensemble des capteurs embarqués dans le terminal. Les informations confidentielles comprennent par défaut le nom, le prénom et une adresse électronique valide. Cependant, le participant a la possibilité de raffiner ces informations, volontairement, en indiquant des lieux géographiques — à gros grain (*par ex.* pays, région ou ville) — où il évolue, son statut ainsi que son âge. Ces informations sont ensuite stockées dans une base de données, et serviront

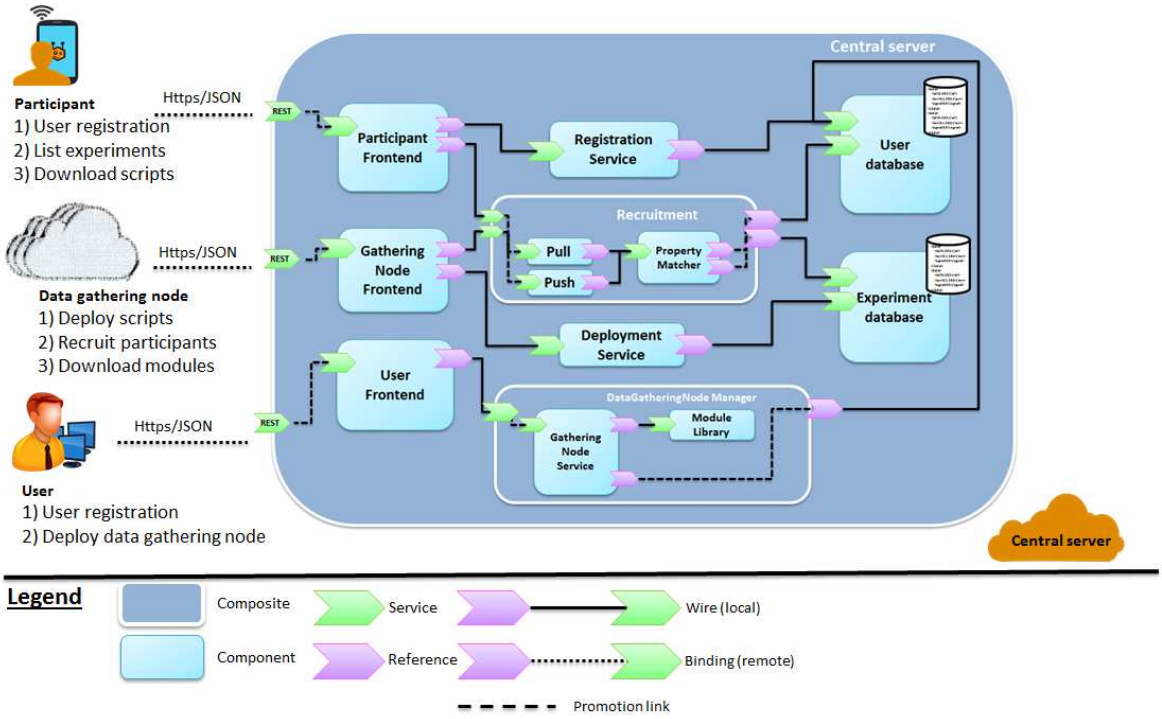

FIGURE 18 – Architecture SCA du serveur central

lors de la phase de déploiement. Pour des raisons de confidentialité, elles pourront être accessibles uniquement par le serveur central, c'est-à-dire qu'elles ne seront pas accessibles par les nœuds de collecte du système.

4.3.2 *L'enregistrement des expériences de collecte*

Accessible par les nœuds de collecte du système, ce service, fourni par le composant Recruitment, permet l'enregistrement d'une expérience de collecte auprès du serveur central. Cette enregistrement comprend deux catégories d'éléments. La première correspond a un ensemble de fichiers JavaScript définissant la tâche de collecte (*cf.* section 3.3, page 59). La deuxième catégorie définit les propriétés de l'expérience et également ses exigences en matière de déploiement. Ces propriétés sont définies par un ensemble de clés/valeurs utilisant le formalisme JSON. Comme le montre le tableau 10, trois catégories de propriétés sont actuellement supportées :

`global` : définit principalement des informations générales sur l'expérience de collecte. Ces propriétés ne sont pas interprétées par le serveur central, mais

propriété	nom	description
`global`	`name`	Nom de l'expérience
	`description`	Description du but de la collecte de données
	`version`	Version de l'expérience de collecte
	`privacy`	Description du modèle de protection de la vie privée
	`incentive`	Description du modèle de récompense
	`nodeurl`	URL du nœud de collecte de collecte
`recruitment`	`sensors`	Liste des capteurs utilisées par la tâche de collecte
	`max_numbers`	Nombre maximum de participants admis dans l'expérience de collecte
	`area`	Zone globale de collecte (*par ex.* pays, région, ville)
	`platform`	Liste de plate-formes mobiles
`context`	`area`	Liste de zone géographique.
	`period`	Liste de période journalière

TABLE 10 – Propriétés définissant une expérience de collecte

permettent d'informer les participants sur les objectifs de l'expérience (*description*), le modèle de confidentialité (*privacy*) et de récompense (*incentive*).

`recruit` : permet de définir les participants autorisés à participer à une expérience de collecte. Ces propriétés permettent de limiter le nombre de participants impliqué dans l'expérience (*max_numbers*), de sélectionner des participants évoluant dans un lieu géographique (*area*) — à gros grains (*par ex.* pays, région ou ville) —, disposant d'un terminal ayant les capteurs nécessaires pour l'exécution de la tâche de collecte (*sensors*) et disposant d'une plate-forme mobile spécifique (*platform*).

`context` : permet de spécifier le contexte d'exécution de la tâche de collecte une fois propagée vers les terminaux mobiles. Dans ce cadre, les propriétés permettent de spécifier une liste de zones géographiques (`area`) — région circulaire définie par les coordonnées du centre et un rayon exprimé en mètres —, et une liste de périodes journalières (`period`) dans lesquelles l'expérience doit s'exécuter. Ces propriétés seront ensuite interprétées par les nœuds mobiles, qui seront responsables de déclencher ou de stopper l'exécution de la tâche de collecte (*cf.* section 3.4.2, page 72).

4.3.3 *Déploiement des tâches de collecte*

Une fois l'expérience enregistrée, deux stratégies principales peuvent être considérées pour propager les tâches de collecte vers les nœuds mobiles des participants [16]. La première approche (*push-based*) consiste à déployer automatiquement l'expérience de collecte auprès des utilisateurs mobiles. Dans la deuxième (*pull-based*), le nœud mobile télécharge la tâche de collecte après une action volontaire du participant, ou en effectuant une requête périodiquement vers le serveur pour voir si une nouvelle expérience est disponible [62]. Dans ce cadre le serveur central supportent ces deux approches pour la propagation des tâches de collecte.

Approche pull-based

La première approche (*pull-based*) est supportée par les composants Pull, DeploymentService et PropertyMatcher. Dans un premier temps, les participants peuvent accéder au service proposé par le Pull, via leur dispositif mobile, pour récupérer une liste d'expériences mises à disposition par les utilisateurs. Dans ce cadre, le PropertyMatcher détermine les expériences accessibles pour le participant, en faisant la correspondance entre les propriétés de recrutement des expériences, et des informations partagées par le participant lors de son inscription. Des informations générales sont fournies pour chaque expérience accessible, permettant au participant de visualiser le but de l'expérience de collecte, du modèle de confidentialité et de récompense ainsi que les capteurs utilisés lors de son exécution. À partir de ces informations, le participant a alors la possibilité de s'enregistrer à une expérience. Lors de la souscription, un identifiant anonyme est alors généré. Cet identifiant est ensuite envoyé au nœud de collecte ayant initié l'expérience et à l'agent mobile, pour l'avertir d'un nouveau recrutement dans l'expérience. Dans ce cadre, l'identifiant anonyme servira pour toutes les communications entre les nœuds de collecte et les participants, comme lors de la propagation des données. Cela permet en effet aux participants de ne pas révéler leurs identités réelles lors des interactions avec les nœuds de collecte, assurant ainsi une première couche d'anonymisation. Une fois la souscription terminée, l'agent mobile peut alors télécharger la tâche de collecte associée et déclencher son exécution.

Approche push-based

La deuxième approche (push-based) est supportée par les composants Push, et également par le composant PropertyMatcher. Dans ce cadre, le Push fournit un service,

accessible par les nœuds de collecte, pour pousser directement une tâche de collecte vers les agents mobiles des participants. Ce processus comprend deux étapes. La première consiste à envoyer un message vers les agents mobiles acceptant ce mode de déploiement, et correspondant aux propriétés de l'expérience. La deuxième étape est ensuite similaire à l'approche présentée ci-dessus, c'est-à-dire composée de la phase de souscription et de téléchargement de la tâche de collecte. Pour établir ce type de propagation, il est nécessaire de supporter des communications dans le sens serveur vers terminaux mobiles. Typiquement, ce type de communication peut être effectué en utilisant des services dédiés, proposés par les fournisseurs des OS mobiles tels que GCM (*Google Cloud Messaging*) pour Android ou APNs (*Apple Push Notification service*) pour iOS. Dans ce contexte, le composant *PushService* est en charge de l'interaction avec ces services pour envoyer des messages vers les terminaux mobiles selon l'OS mobile visé.

Indépendamment de l'approche utilisée, le composant Push permet également aux nœuds de collecte de propager des messages aux utilisateurs mobiles s'étant inscrits à leurs expériences. Ces messages peuvent être propagés soit à un participant en particulier, ou alors à l'ensemble des participants ayant souscrits à l'expérience. Deux types de messages peuvent être transmis :

`update` : Message indiquant la disponibilité d'une nouvelle mise à jour de l'expérience. Dans ce cadre, ce message déclenchera le téléchargement de la nouvelle version de l'expérience par les nœuds mobiles. Cela permet de rapidement adapter le comportement de l'expérience une fois déployée.

`notification` : Ces messages permettent aux utilisateurs de notifier l'état d'avancement de l'expérience en cours, en partageant des statistiques globales des données collectées avec tous les participants.

4.3.4 *Gestion des nœuds de collecte*

Les derniers services présentés sont responsables de la création et de la configuration des nœuds de collecte. Un utilisateur voulant évoluer dans APISENSE®, peut se connecter au service géré par le composant DataGatheringRepository via une interface web, et déclencher le téléchargement de leur propre nœud de collecte. Celui-ci pourra ensuite être déployé vers une infrastructure publique ou privée, selon les exigences des utilisateurs.

Un nœud de collecte peut être téléchargé suivant deux formats :

1. En tant qu'image virtuelle pré-configurée, permettant son déploiement au-dessus d'un environnement virtualisé. Dans ce cadre, l'image virtuelle comprend toute la pile logicielle nécessaire au bon fonctionnement du nœud de collecte. Plus particulièrement, elle est composée d'une distribution Linux, d'une machine virtuelle Java (JVM), d'une base de données MongoDb [1] pour le stockage des données, et FraSCAti responsable de l'exécution des composants SCA du nœud de collecte.
2. En tant qu'archive web (.war), permettant son déploiement directement au-dessus d'un serveur d'application (*c.-à-d.* Apache Tomcat).

Le service fourni par le *ComponentRepository* propose un ensemble de composants modulaires, dédiés à la configuration d'un nœud de collecte. Les utilisateurs peuvent accéder à ce service via leur nœud de collecte nouvellement déployé, pour télécharger et assembler des composants dans leur nœud de collecte en fonction de la nature des expériences qu'ils veulent mener. Nous présentons ce point plus en détail dans la section suivante.

4.4 ARCHITECTURE DES NOEUDS DE COLLECTE

Dans la section précédente, nous avons présenté l'architecture et les services fournis par le serveur central de la plate-forme APISENSE®. Dans ce chapitre, nous présentons l'architecture des nœuds de collecte, et plus particulièrement comment nous tirons profit du modèle SCA pour fournir des nœuds de collecte facilement configurables. Les nœuds de collecte sont les entités logicielles de notre système responsables de la gestion des campagnes de collecte des utilisateurs. Nous rappelons qu'une campagne est essentiellement composée de quatre étapes qui consiste :

1. à définir une tâche de collecte via une interface de programmation dédiée,
2. à publier la tâche de collecte auprès du serveur central pour le recrutement des participants,
3. à assurer la persistance des données collectées par les dispositifs,
4. et finalement à analyser et exploiter les données collectées

Par ailleurs, une expérience peut également comporter un modèle de protection de la vie privée des participants, ainsi qu'un modèle de récompense.

1. https://www.mongodb.org

Pour faire face à la diversité des expériences qui peuvent être menées, les nœuds de collecte peuvent être entièrement configurés selon la nature d'une expérience. Cette configuration consiste essentiellement à assembler un ensemble de composants SCA, où chaque composant est responsable de la logique fonctionnelle d'une étape d'une expérience. Dans ce contexte, plusieurs composants peuvent être dédiés à une même étape, représentant la partie variable d'une configuration.

Pour guider les utilisateurs dans ce processus d'assemblage, nous avons adopté une approche basée sur l'ingénierie des Lignes de Produits Logiciels (LDPs) [12, 52] (en anglais SPL pour *Software Product Line*). Typiquement, l'idée est i) de définir un Modèle de Caractéristiques (MC) [32](en anglais FM pour *Feature Model*) qui capture les caractéristiques communes et les points de variabilité d'une configuration d'une expérience, ii) d'implémenter le MC comme un assemblage de composants SCA et de proposer le MC aux utilisateurs pour leurs permettre de sélectionner les composants selon la nature des expériences qu'ils veulent mener.

Dans la suite de cette section, nous présentons le MC dédié à la configuration des expériences de collecte en sous-section 4.4.1, un exemple d'architecture SCA généré et leurs interactions permettant la réalisation d'une campagne de collecte, et nous finissons par présenter le mécanisme d'extension permettant d'adresser les exigences non anticipées.

4.4.1 *Modèle de caractèristiques d'une campagne de collecte*

La notion de Modèle de Caractéristiques (MC), introduite par Kang et al. [32], est une technique de modélisation permettant la description d'un ensemble d'artéfacts réutilisables, qui peuvent être utilisés pour créer une famille de logiciels qui possède des caractéristiques communes, mais exhibant également des différences. Le MC représente ainsi l'ensemble des choix proposés à un utilisateur pour lui permettre de configurer un logiciel spécifique à ses besoins.

Typiquement, un MC est spécifié sous forme d'arbre, dont les nœuds représentent les caractéristiques d'un logiciel et les arcs spécifient des liens de composition entre les caractéristiques. Trois catégories de caractéristique peuvent être définies : les caractéristiques obligatoires, les caractéristiques optionnelles et les caractéristiques alternatives. Additionnellement, des règles de composition peuvent être définies, établissant une

relation entre les caractéristiques. Ces règles peuvent prendre deux formes i) une caractéristique qui nécessite la sélection d'une autre caractéristique (définissant une interdépendance entre les caractéristiques), et ii) la sélection d'une caractéristique peut exclure la sélection d'une autre.

Dans notre contexte d'utilisation, nous utilisons cette technique pour modéliser les différents choix proposés aux utilisateurs pour leur permettre de configurer les services d'une campagne de collecte selon les spécificités de celle-ci. La figure 19 illustre le MC dédié à la configuration d'une campagne de collecte. Typiquement, le MC identifie principalement cinq points de variabilités pour i) configurer l'environnement de développement des tâches de collecte (SensingTask Description), ii) choisir un modèle de recrutement des utilisateurs (Recruitment), iii) choisir une série de traitements sur les données collectées avant leurs insertions dans la base de données (OnlineProcessing), iv) spécifier l'indexation des données dans la base de données (Collector), v) exposer des services au-dessus des données collectées (Service).

Nous décrivons à présent brièvement ces différents points de variabilités et les caractéristiques actuellement supportées.

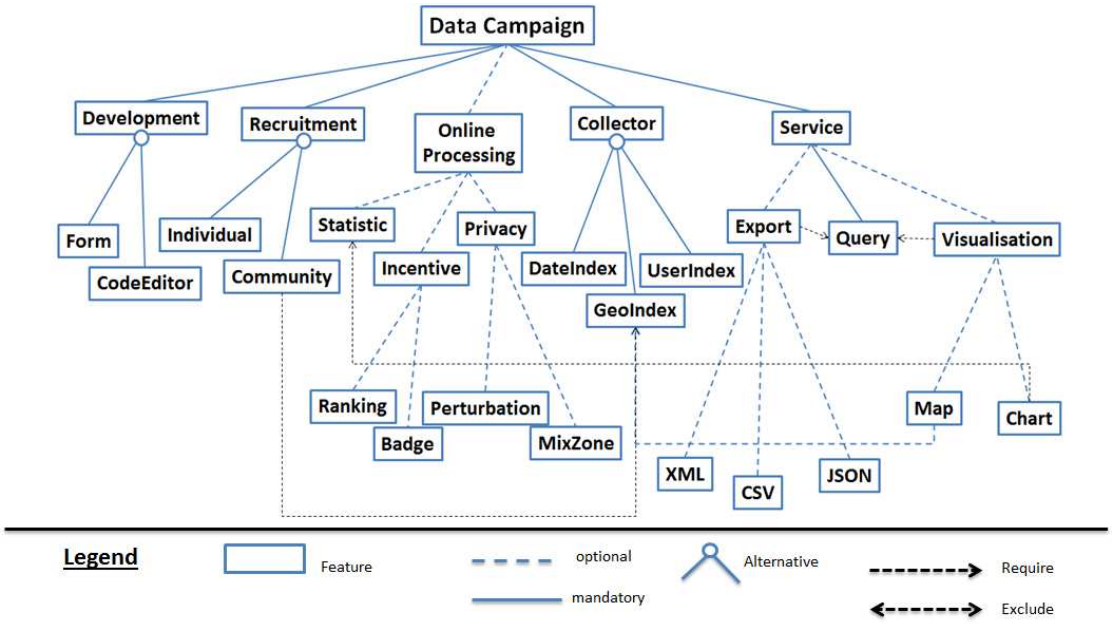

FIGURE 19 – Modèle de caractéristiques d'une campagne de collecte

SensingTask Description

Ce premier point de variabilité permet aux utilisateurs de sélectionner l'interface web dédiée aux développements des tâches de collecte vue dans le chapitre précédent (*cf.* section 3.3). La première interface, définie par la caractéristique CodeEditor, propose un éditeur de code en ligne — avec coloration syntaxique, complétion de code, etc. — permettant le développement des tâches de collecte directement en JavaScript. La deuxième interface, définie par la caractéristique Form, quant à elle, fournit une interface web permettant le développement des tâches à partir d'un formulaire. Le formulaire permet principalement de spécifier le type de données à collecter (*par ex.* position, accéléromètre, périphérique Bluetooth) avec la fréquence de collecte pour chaque donnée (*par ex.* toutes les cinq minutes). Après validation du formulaire, un fichier JavaScript est généré à partir des choix effectués par l'utilisateur. Le formulaire permet de rendre accessible le développement des tâches pour des utilisateurs n'ayant aucune expertise en programmation. Cependant, le formulaire limite l'expressivité des tâches qui peuvent être développées, en définissant uniquement des tâches de collecte périodique de données. Nous prévoyons dans des travaux futurs de proposer des métaphores visuelles pour développer des tâches de collecte plus complexes, tout en permettant leur développement par des utilisateurs n'ayant aucune expertise en développement. Certains travaux ont déjà été proposés dans ce sens tels que Scratch [48, 58] ou Alice [15] ou plus récemment App Inventor [2]. Le principe de ces approches consiste à fournir un environnement intuitif, dans lequel on associe des composants visuellement pour définir le comportement d'une application.

Recruitment

Ce deuxième point de variabilité permet de sélectionner le modèle utilisé pour recruter un sous-ensemble de participants exécutant la tâche de collecte. Dans ce contexte, le recrutement consiste essentiellement à interagir avec les services du serveur central pour déployer la tâche de collecte et assurer sa mise à jour, définir un sous-ensemble de participants autorisés à l'exécuter et son contexte d'exécution. Les deux caractéristiques associées à ce point de variabilité (Individuel et Collaborative) permettent de faire face aux différentes échelles de collecte (*cf.* section 2.1.2) que sont la collecte personnelle et la collecte communautaire.

La caractéristique Individuel propose un modèle simple de recrutement des participants dédié à la collecte personnelle de données, qui a généralement pour objectif de suivre l'activité d'un individu à travers le temps. Nous présentons plus en détail en

2. http://appinventor.mit.edu

sous-section 4.4.4 ce modèle et les différentes interactions entre les nœuds mobiles, le serveur central et les nœuds de collecte nécessaires. La caractéristique Collaborative propose un modèle principalement dédié à la collecte communautaire de données, qui a généralement pour objectif de surveiller des phénomènes liés à l'environnement des participants (*par ex.* surveiller la qualité du signal réseau dans la ville de Paris). Nous décrirons plus en détail ce modèle dans la section 4.5.

Online Processing

Ce point de variabilité permet de définir un ensemble de traitements à effectuer sur les données collectées avant leurs insertions dans une base de données. Les traitements peuvent consister à calculer des statistiques globales sur les données collectées (caractéristique Statistique), calculer un modèle d'incitation pour récompenser les participants (point de variabilité Incentive), ou encore effectuer des traitements modifiant les données collectées pour assurer leurs confidentialités (point de variabilité Privacy).

Incentive

Ce point de variabilité définit le modèle de récompense de l'expérience de collecte. Typiquement, l'incitation peut prendre une grande variété de formes. Par exemple, récompenser les participants financièrement [41] pour les données partagées, ou transférer des mécanismes de jeux dans le contexte de la collection de données [69]. Généralement, un modèle de récompense a pour objectif d'augmenter la participation des utilisateurs, et de les motiver à partager des données plus pertinentes pour l'expérience.

Actuellement, nous supportons un modèle de récompense [24] basé sur la quantité et la qualité des données partagée par des participants. Ce modèle est également utilisé dans des applications mobiles populaires telles que Waze [3] ou encore Foursquare [4]. Le modèle est basé sur le concept de récompense [69] qui peut prendre deux formes différentes. La première consiste à émuler une compétition, en effectuant un classement entre les participants. Dans ce cadre, le modèle peut être configuré par les utilisateurs, en attribuant des points selon le type de données partagées. Le deuxième permet d'attribuer des badges/trophées, en fonction du nombre de points remportés par les participants.

3. https://www.waze.com
4. https://foursquare.com

Dans l'application mobile présentée dans le chapitre précédant, plusieurs mécanismes permettent aux participants de contrôler les données qui sont collectées par leur dispositif (*cf.* chapitre 3 sous-section 3.4.2, page 72). Ce mécanisme peut être utilisé par exemple par les participants pour empêcher l'application d'activer le GPS qui consomme beaucoup d'énergie. Dans ce cas, les utilisateurs peuvent contrebalancer l'énergie utilisée par les capteurs, et la sensibilité des données — d'un point de vue vie privée —, en attribuant plus de points selon les données collectées par les participants. Par exemple, le capteur GPS est bien connu pour fournir une grande précision pour déterminer la position d'un utilisateur, mais consomme beaucoup plus d'énergie que la géolocalisation GSM, moins précise. Dans ce contexte, pour inciter les participants à partager leurs capteurs GPS, plus de points peuvent être attribués aux données collectées par le GPS.

Privacy

Additionnellement, un modèle de confidentialité peut être intégré dans l'expérience de collecte. Dans ce cadre, les composants implémentant ces modèles sont responsables de l'assainissement des données reportées par les participants, avant leurs insertions dans la base de données. Actuellement nous supportons un modèle classique de perturbation [2], consistant à modifier les coordonnées spatiales et temporelles en y ajoutant une perturbation aléatoire. La perturbation est introduite par un bruit gaussien à l'intérieur d'une sphère de rayon `r` centré sur les coordonnées d'origine. Nous envisageons d'intégrer plusieurs modèles issus de l'état de l'art [11] tels que des modèles de dissimulation spatiale [64, 30, 62].

Collector

Ce point de variabilité est responsable de la persistance ainsi que de l'indexation des données collectées dans une base de données. Actuellement, la persistance des données est effectuée avec la technologie MongoDB, une base de données non-relationnelle orientée documents. Dans notre cas d'utilisation, le principal bénéfice de cette technologie est de ne pas imposer un schéma spécifique pour sauvegarder les données. En effet, de nombreux systèmes [25, 22] utilisent des bases de données SQL qui nécessitent la définition de schéma statique pour structurer les données. Ce choix de conception limite alors la réutilisation du système nécessitant la sauvegarde de données ayant une structure non prévue dans le schéma défini au préalable. Dans ce cadre, MongoDB représente une technologie fournissant la flexibilité nécessaire pour faire face à l'hétérogénéité des données qui peuvent être collectées durant les expériences.

Les données sont sérialisées sous la forme d'un tableau JSON. Dans ce cadre, l'utilisation du formalisme JSON fournit une grande flexibilité aux utilisateurs pour représenter les données, selon le type des capteurs impliqués dans la tâche de collecte. La structure des données est définie par les utilisateurs lors du développement de la tâche de collecte (*cf.* sous-section 3.3.3). Les métadonnées sont générées par l'application mobile, comportant l'identifiant unique de l'utilisateur — crée par le serveur central lors de l'enregistrement du participant dans l'expérience —, la version de la tâche de collecte ainsi que la date où les données ont été générées. Les utilisateurs ont également la possibilité de compléter ces méta-informations. Dans ce cadre, ces méta-informations peuvent être utilisées pour structurer les données dans la base de données, par exemple indexer les données par jour (caractéristique DateIndex), par utilisateur (caractéristique UserIndex) ou utiliser un index géographique (caractéristique GeoIndex) pour faciliter le traitement des données à caractère géographique.

Service

Le dernier point de variabilité permet de sélectionner des services responsables d'exposer les données après leurs sauvegardes dans la base de données. Actuellement, nous supportons des composants génériques qui permettent de fournir une interface web donnant un accès à la base de données (caractéristique Query), de visualiser des statistiques sur données collectées sous forme de graphe (caractéristique Chart), visualiser sous forme de carte (caractéristique Map) si l'indexation sélectionnée dans la partie précédente utilise un index géographique, ou alors d'exporter les données dans divers formats (point de variabilité Export) (*c.-à-d.* CSV, JSON, XML).

Dans cette section, nous avons présenté notre approche pour permettre de faire à face à la diversité des campagnes de collecte qui peuvent être menées par les utilisateurs. Pour prendre en compte cette diversité, nous avons proposé de modéliser l'environnement responsable de la gestion d'une campagne comme une famille de produits à l'aide de Modèle de Caractéristique (MC). Le MC présenté permet de définir les points communs et également les différences relatives aux campagnes de collecte. Ces différences peuvent consister à utiliser différents modèles de confidentialité, de recrutement ou d'incitation selon la nature de l'expérience, choisir différentes façons pour développer les tâches de collecte selon le niveau de programmation des utilisateurs ou alors structurer et exposer les données collectées durant la campagne. Ainsi, le MC définit l'ensemble des choix présentés aux utilisateurs pour leur permettre de configurer leurs campagnes de collecte en fonction de leurs exigences. Dans la section

suivante, nous décrivons comment un nœud de collecte génère un environnement assurant la gestion d'une campagne de collecte à partir des choix effectués par les utilisateurs et comment les utilisateurs peuvent étendre les nœuds de collecte pour faire face à des exigences non anticipées.

4.4.2 *Création d'une nouvelle campagne*

Toutes les caractéristiques initialement introduites dans la section précédente sont associées à un composites SCA. La création d'une nouvelle campagne consiste donc à intégrer et à assembler dans l'architecture initiale du nœud de collecte (*cf.* figure 20) un ensemble de composite SCA. Dans ce cadre, le MC permet de spécifier l'ensemble des assemblages valides pour générer une nouvelle instance de campagne de collecte.

Une fois déployé, un nœud de collecte (*cf.* figure 20) est composé de quatre composants génériques, dédiés à la création d'une nouvelle campagne : ExperimentManager, WebManager, ModuleManager et le ReconfigurationManager.

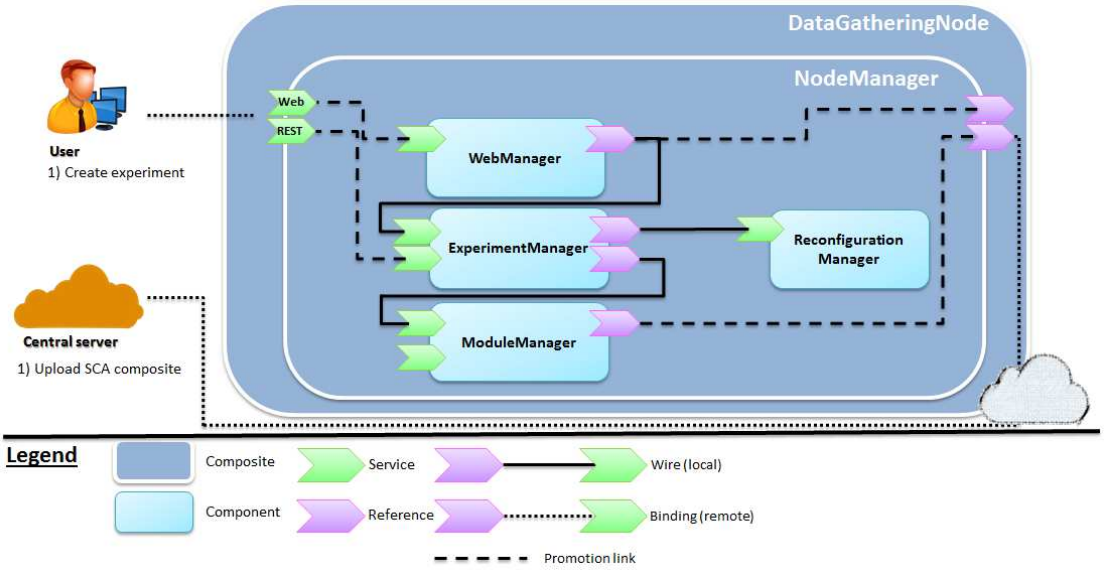

Figure 20 – Architecture initiale d'un nœud de collecte

Le point d'entrée de cette architecture est le composant ExperimentManager. Ce composant est responsable du déclenchement de la génération d'une nouvelle campagne. Pour lancer cette génération, ce dernier composant a besoin de trois informations : i) le nom de la campagne de collecte, ii) une description de l'objectif de la collecte, iii) et l'ensemble des caractéristiques sélectionnées dans le MC par les utilisateurs. La

création d'une nouvelle campagne peut également s'effectuer à partir d'une interface web fournie par le composant WebManager.

La figure 21 montre une vision simplifiée de la génération d'une nouvelle campagne.

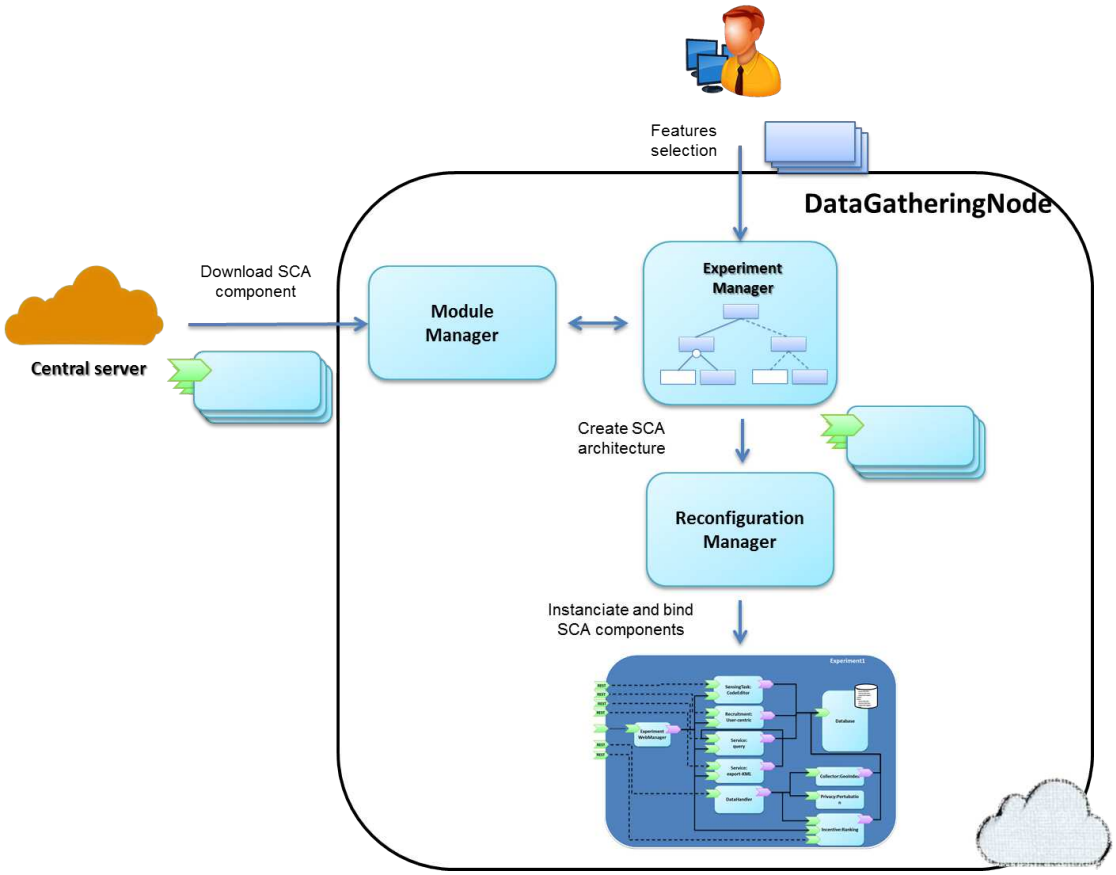

Figure 21 – Processus de génération d'une architecture SCA

Une fois la sélection validée, en fonction des différentes contraintes exprimées dans le MC, le composant ExperimentManager procède alors aux téléchargements des différents composites associés aux caractéristiques sélectionnées, auprès du serveur central via le composant ModuleLibrary. Une fois les composites téléchargés, le composant ExperimentManager exploite alors les capacités d'introspection du moteur d'exécution SCA via le composant ReconfigurationManager pour instancier les composites téléchargés dans l'architecture du nœud de collecte.

Par exemple, la figure 22 illustre une nouvelle instance générée à partir de la sélection des caractéristiques suivantes : CodeEditor, Individuel, Badge, Pertubation, userIndex, Query, Export-KML. Cette nouvelle instance expose ainsi de nouveaux services permettant

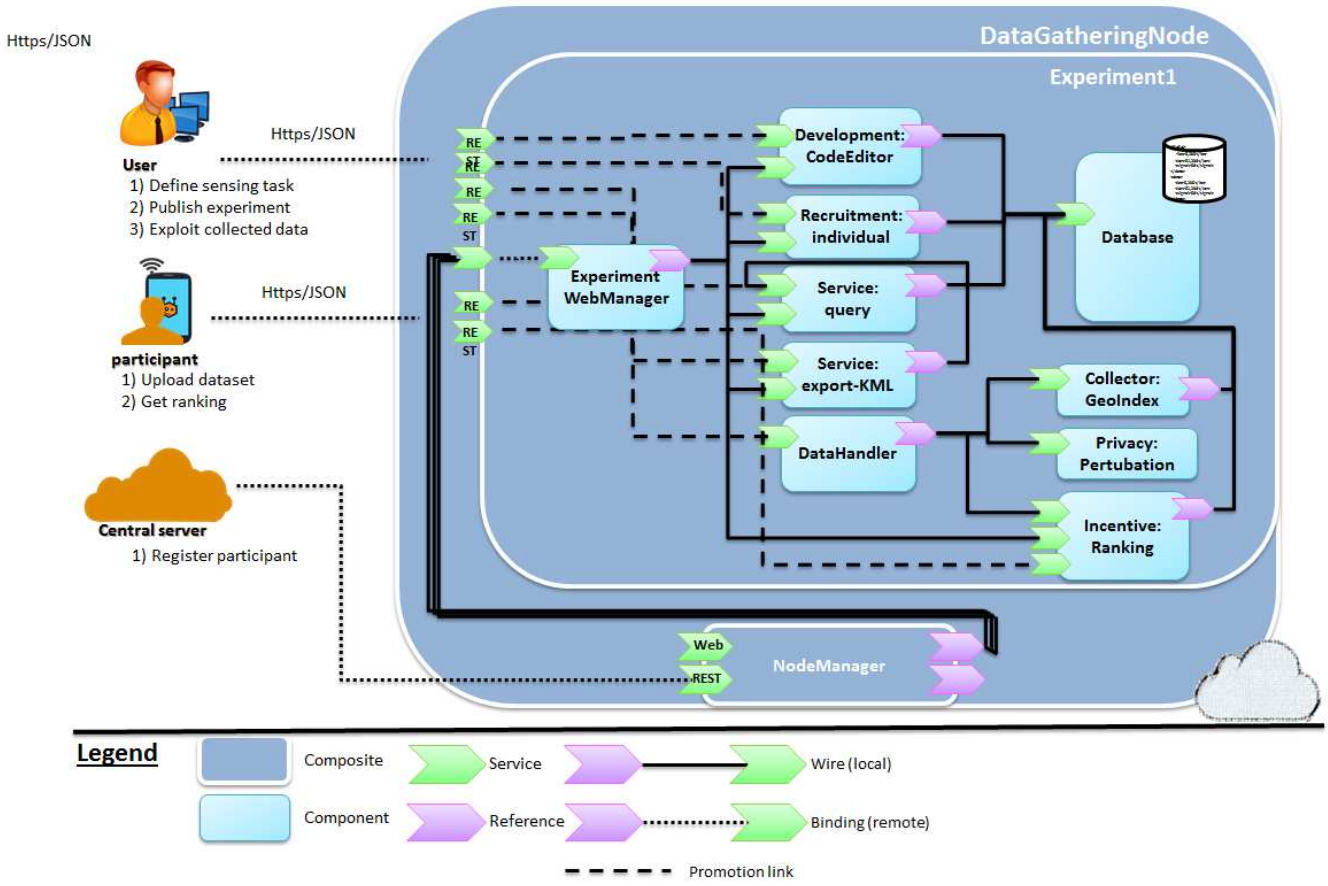

FIGURE 22 – Architecture SCA d'une campagne de collecte

d'éditer une tâche de collecte associée à la campagne (composant SensingTask :CodeEditor), de déployer cette tâche auprès du serveur central afin de recruter un ensemble de participants (composant Recruitement :Individuel), d'anonymiser et de sauvegarder les données collectées par les participants (composant Privacy :Perturbation et Collector :UserIndex), configurer le mécanisme de récompense (composant Incentive :Ranking) et finalement exploiter les données collecter (composant Service :Query et Service :Export :KML). Nous présentons de manière plus détaillée dans la section suivante comment ces composants interagissent entre eux lors d'une campagne de collecte configurée avec un modèle de recrutement Individuel. L'exécution d'une campagne communautaire sera décrite en section 4.5.

4.4.3 *Intéraction entre les composants*

Après avoir présenté les différents composants logiciels formant le serveur central et les nœuds de collecte de APISENSE®, nous présentons dans cette sous-section leurs interactions. La figure 23 montre un diagramme de séquence illustrant les différents messages échangés entre les acteurs du système (*c.-à-d.* utilisateurs et participants) et les composants de APISENSE®, lors de l'exécution d'une campagne de collecte personnelle. Comme le montre le diagramme, la réalisation d'une campagne de collecte comporte trois phases : la création de la campagne de collecte (phase 1), le déploiement d'une tâche de collecte vers les dispositifs mobiles des participants (phase 2) et la propagation des données vers un nœud de collecte. (phase 3).

phase 1 - création d'une campagne de collecte La création d'une nouvelle campagne de collecte est initiée par un utilisateur à partir de son nœud de collecte. La création de la campagne consiste à sélectionner un ensemble de caractéristiques, appartenant au MC, définissant la configuration de la campagne de collecte. À partir d'une configuration valide, le composant *ExperimentManager* télécharge les composants SCA associés aux caractéristiques auprès du serveur central et génère une nouvelle instance de la campagne (*cf.* figure 22). La nouvelle instance est un composite SCA, constitué d'un ensemble de composants qui exposent automatiquement de nouveaux services permettant le développement de la tâche de collecte (composant *SensingTask*), de la publier (composant *Recruitment*), de reporter les données collectées par les dispositifs mobiles (composant *DataHandler*) et de récupérer les données collectées (composant Query).

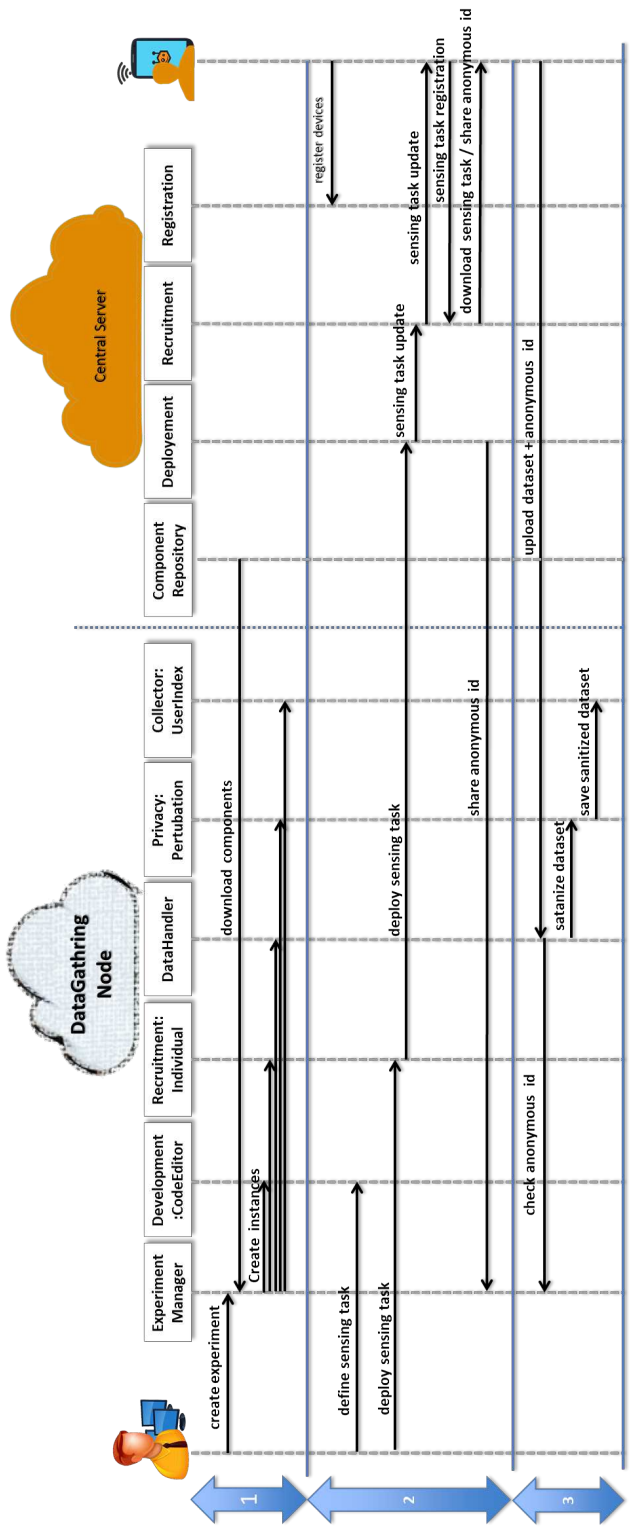

FIGURE 23 – Interaction des composants APISENSE®

PHASE 2 - DÉPLOIEMENT DE LA TÂCHE DE COLLECTE Une fois les services de la nouvelle campagne de collecte exposés, les utilisateurs peuvent alors développer la tâche de collecte associée à la campagne de collecte via le composant *SensingTask*, configurer les propriétés de déploiement de la tâche de collecte (*cf.* section 4.3.2) et la publier auprès du serveur central via le composant *Deployment*. Une fois publiée, le composant *Recruitment* du serveur central analyse les propriétés de la tâche de collecte et notifie tous les participants, correspondant à ces propriétés, de la disponibilité d'une nouvelle tâche de collecte est disponible. Les participants volontaires peuvent alors s'y inscrire. Dans ce cas, le dispositif mobile télécharge la tâche de collecte et l'identifiant anonyme généré lors de l'inscription qui est également partagé avec le composant *ExperimentManager* du nœud de collecte. Une fois téléchargée, la tâche de collecte est alors exécutée sur le dispositif mobile.

PHASE 3 - PROPAGATION DES DONNÉES La dernière phase consiste à reporter les données collectées durant l'exécution de la tâche de collecte. Les données sont directement reportées auprès du composant *DataHandler* du nœud de collecte ayant initié la tâche de collecte. Lors de la propagation des données, l'identifiant anonyme est également envoyé, permettant au nœud de collecte de vérifier si les données proviennent bien d'un utilisateur inscrit à l'expérience. Si l'identifiant est valide, le composant *DataHandler* fait appel au composant *Privacy* qui est responsable de l'assainissement des données, avant leur sauvegarde et leur indexation dans la base de données via le composant *Collector*. Les utilisateurs peuvent alors accéder aux données sauvegardées, exporter les données collectées, et également retourner à la phase 2 afin de modifier le comportement de la tâche de collecte en fonction des résultats observés.

Nous venons de décrire l'ensemble des interactions des composants APISENSE® permettant la réalisation d'une campagne de collecte de données. La section suivante présente le mécanisme d'extension de APISENSE® permettant le développement et l'intégration de nouveau composant dans la plate-forme.

4.4.4 *Extension des nœuds de collecte*

En définissant un nœud de collecte comme une application SCA, nous bénéficions aussi de sa modularité pour facilement étendre la plate-forme. Dans cette sous-section, nous présentons comment de nouveaux services peuvent être ajoutés à un nœud de

collecte. L'ajout d'un nouveau service est effectué à partir d'une archive ZIP, que nous appellerons une contribution. Comme le montre l'exemple illustré par la figure 24, une contribution contient la description d'une caractéristique, associée à un composite SCA, ainsi que l'ensemble des artéfacts implémentant le composite (*c.-à-d.* code compilé, scripts). La contribution présentée vise donc à ajouter un nouveau service responsable d'exporter les données collectées sous le format KML [5], un format d'encodage utilisant le formalisme XML destiné à la gestion de l'affichage de données géospatiales dans les systèmes d'information géographique.

Dans ce cadre, le premier élément vise à définir la contribution comme une caractéristique s'inscrivant dans le MC général présenté en sous-section 4.4.1. La description de la caractéristique définit donc le nœud parent (*c.-à-d.* caractéristique *Export* dans l'exemple), identifiant un nouveau point de variabilité, les caractéristiques requises, et le composite SCA associé. Dans cet exemple, le composite définit un nouveau service (*c.-à-d.* export service), exposé comme une ressource *REST*. Afin de récupérer les données collectées, le composant déclare une référence vers un composant fournissant un service de requête vers la base de données. Et finalement, le composant, déclare son implémentation comme étant une implémentation en Java. Les nœuds de collecte et le serveur central fournissent un service dédié aux déploiements de nouvelles contributions. Dans ce cadre, un utilisateur peut déployer une contribution directement dans son nœud de collecte, ou directement sur le serveur central pour la partager avec les autres utilisateurs du système.

4.5 CAMPAGNE DE COLLECTE COMMUNAUTAIRE

Dans la section précédente, nous avons présenté les principaux composants des nœuds de collecte et plus particulièrement comment ces composants peuvent être assemblés pour créer des services dédiés à une campagne de collecte spécifique. Nous avons également présenté le premier modèle de recrutement Individuel, où le recrutement est basé uniquement sur les propriétés des participants.

Dans cette section, nous décrivons le modèle de recrutement Communautaire. Ce type de campagne, également appelé *public sensing* dans la littérature [33], consiste à collecter continuellement des données relatives à l'environnement des participants (*par ex.* bruit ambiant, qualité réseaux, pollution de l'air) dans une zone géographique [47, 63, 50].

5. https://developers.google.com/kml/documentation

FIGURE 24 – Exemple d'extension : Export des données au format KML

Typiquement, une approche couramment utilisée dans ce type de campagne consiste à impliquer un maximum de dispositifs mobiles dans la collecte de données. Cependant, ces dispositifs collectent périodiquement (*c.-à-d.* toutes les x secondes) ces données environnementales sans prendre en considération si un autre nœud mobile placé à proximité collecte les mêmes données [27]. Cette approche naïve peut s'avérer inefficace principalement pour deux raisons. La première est qu'elle peut impliquer la remontée d'une grande quantité de données similaires, en particulier en milieu urbain ou une forte concentration d'utilisateurs (*c.-à-d.* situés à proximité) peuvent collecter simultanément ces données identiques. Cela demande par conséquent beaucoup plus de ressources côté serveur pour stocker ces données et les analyser. La seconde est qu'elle implique également une grande perte énergétique des appareils mobiles qui collectent ces données similaires. Récemment, Sheng et al. [59] ont montré qu'effectuer une collaboration entre les dispositifs, en fonction d'un objectif de couverture de la zone à observer, pouvait radicalement diminuer l'énergie consommée par l'ensemble des dispositifs, et par conséquent diminuer également le volume de données transmises sur le serveur. Dans ce cas, les objectifs de couverture définissent uniquement la quantité de données nécessaire pour la campagne, permettant de faire un compromis entre la quantité des données collectées, et le coût énergétique liée à leurs acquisitions. Pour effectuer la collaboration, cette approche fais l'hypothèse que les trajets de chaque appareil mobile est connue à l'avance. À partir de ces trajets, l'algorithme proposé dans cette approche permet d'assimiler les tâches de collecte uniquement aux mobiles capables de couvrir les objectifs de couvertures définis au préalable . Cependant, faire l'hypothèse que la mobilité de chaque appareil mobile est connue à l'avance peut être difficilement appliqué en déploiement réel, dû à la difficulté de prédire les trajectoires des participants [60].

Dans ce contexte, nous présentons dans cette section un modèle de recrutement collaboratif, permettant de coordonner l'exécution des tâches de collecte sans avoir une connaissance au préalable de la mobilité des participants. Le modèle proposé a pour objectif dans un premier temps de diminuer la consommation énergétique globale de tous les dispositifs impliqués dans la campagne, et également de diminuer la quantité de données propagées vers le nœud de collecte. Dans notre architecture, ce modèle est assuré par le composant associé à la caractéristique Collaboration du MC présentée en sous-section 4.4.1. Pour assurer la coordination des tâches, le composant se base sur des propriétés de couverture géographique et temporelle décrites par les utilisateurs, leur permettant de définir seulement la quantité de données nécessaires pour leur

campagne. Pour définir ces propriétés, ainsi que différentes exigences en relation avec leur campagne, nous proposons une interface de programmation complémentaire à celle présentée dans le chapitre précédent (*cf.* section 3.3). Dans la suite de cette section, nous présentons tout d'abord l'interface de programmation proposée suivie d'un exemple de campagne de collecte développée avec cette interface. Nous présentons par la suite les différents processus assurant l'exécution ainsi que la coordination des ces campagnes de collecte.

4.5.1 *Extension du modèle de programmation*

Dans cette section, nous présentons l'interface de programmation dédié à la définition de campagne de collecte de données communautaire. La table 11 illustre les principales fonctions proposées par cette interface.

Phase	Méthode	Description
Collecte	`sense(callback : function())`	Enregistrement d'une tâche de collecte
Recrutement	`accept(function())`	Définition des dispositifs autorisés à exécuter une tâche
	`ranking(function(users))`	Classement des dispositifs prioritaires pour l'exécution de la tâche
Couverture	`geoCoverage(bound : Array, coverage : String)`	Définition des propriétés de couverture géographique
	`timeCoverage(during : Number, every : Number)`	Définition des propriétés de couverture temporelle
	`duplicate(n : Number)`	Nombre de dispositifs attribués à une tâche selon les propriétés de couverture.

TABLE 11 – Interface de programmation dédiée à la définition des campagnes de collecte communautaires.

Typiquement, la définition d'une campagne de collecte communautaire comprend trois phases : collecte, recrutement, couverture.

COLLECTE : La première phase consiste à enregistrer la tâche de collecte qui sera distribuée auprès des nœuds mobiles. La tâche de collecte décrit les données qui doivent être collectées et reportées par les nœuds mobiles (*cf.* section 3.3, page 59). L'enregistrement de celle-ci est effectué en insérant une fonction de rappel

à la méthode `sense`. Cette tâche sera ensuite exécutée par les nœuds mobiles lorsqu'ils seront assignés à celle-ci.

RECRUTEMENT : La deuxième phase consiste à définir la stratégie de recrutement des nœuds mobiles. Cette phase permet de filtrer les nœuds mobiles, ou d'en privilégier certains en fonction de leur contexte lors de l'attribution de la tâche de collecte. Par exemple, cela permet de baser le recrutement sur certains attributs (*par ex.* type de connexion réseaux, vitesse de déplacement), ou de privilégier les nœuds mobiles ayant le niveau de batterie le plus élevé. Le filtrage des nœuds mobiles consiste à enregistrer une fonction de rappel à la méthode `accept`. Cette fonction sera exécutée par les nœuds mobiles avant l'attribution de la tâche de collecte à un ou plusieurs nœuds. Cette fonction a pour objectif d'observer le contexte du nœud mobile, et définir si celui-ci est autorisé à exécuter la tâche. Dans ce cas, cette fonction doit retourner une liste de propriétés (*par ex.* niveau de batterie, vitesse de déplacement, qualité du signal réseau), dans le cas contraire, elle doit retourner une valeur nulle. La méthode `ranking` permet de privilégier certains nœuds mobiles pour exécuter une tâche de collecte, basée sur les propriétés retournées par la méthode `accept`.

COUVERTURE : La troisième phase permet de définir les objectifs de couverture temporelle et géographique de la campagne de collecte. Cela permet par exemple de faire un compromis entre le coût énergétique utilisé par l'ensemble de nœuds mobiles, et la quantité de données reportées pendant la campagne. L'objectif de couverture temporelle peut être défini par la méthode `timeCoverage`, qui prend deux paramètres. Ces deux paramètres permettent de spécifier pendant combien de temps la tâche de collecte doit être exécutée et à quelle fréquence (*par ex.* exécuter la tâche pendant 30 minutes toutes les heures). L'objectif de couverture géographique peut être défini par la méthode `geoCoverage`, qui prend également deux paramètres. Ces paramètres permettent de spécifier la zone géographique globale de la campagne et à quelle granularité la tâche de collecte doit être distribuée (*par ex.* exécuter la tâche dans la ville de Paris tous les 500 mètres). La méthode `duplicate` permet de définir le nombre de dispositifs attribués à la tâche de collecte en fonction des propriétés de couvertures.

Exemple d'application Afin de mieux illustrer le modèle de programmation proposé, nous présentons ici un exemple de campagne de collecte de données. Nous utiliserons également cet exemple comme un fil conducteur tout au long de cette section. Dans cette campagne, décrites par le listing 4.1, nous voulons mettre en place une collecte de

données permettant d'élaborer une cartographie représentant la qualité des réseaux GSM en fonction des opérateurs des réseaux mobiles dans la ville de Paris.

La première phase consiste à enregistrer la tâche de collecte de cette campagne (ligne 2-9). La tâche de collecte consiste à exécuter 10 Ping (Packet INternet Groper) réseaux à une machine distante particulière lorsque le participant a changé de position (ligne 3), et à collecté la latence moyenne des requêtes effectuées (ligne 7), la position de l'utilisateur (ligne 6) ainsi que son opérateur de réseau mobile (ligne 5).

La deuxième phase consiste à définir la stratégie de recrutement des utilisateurs. Dans cette phase, nous voulons recruter uniquement les participants disposant d'une connexion de données mobiles (ligne 13). Pour les participants ayant ce type de connexion, nous renvoyons le niveau actuel de leur batterie pour pouvoir privilégier l'attribution de la tâches de collecte au participant disposant du niveau de batterie le plus élevé (ligne 18-20). Dans le cas contraire, le participant n'est pas autorisé à exécuter la tâche de collecte.

Dans la troisième phase, nous définissons les propriétés de couverture pour optimiser l'énergie utilisée par l'ensemble des dispositifs mobile, et minimiser la quantité de données reportées sur le serveur. Pour cela, nous définissons que la tâche de collecte doit être exécutée au maximum par deux participants (ligne 25) répartis tous les cinq-cents mètres dans la ville de Paris (ligne 23), et que la tâche doit être exécutée pendant trente minutes toutes les heures (ligne 24).

```
1  // phase 1 : définition de la tâche de collecte
2  sense(function() {
3        $location.onLocationChanged(function(event) {
4              $trace.add({
5                    operator : $telephony.operator(),
6                    loc: [event.latitude, event.longitude],
7                    latency : $network.ping(10,"http://...").average});
8              })}
9  })
10
11 // phase 2 : définition de la stratégie de recrutement
12 accept(function(){
13       if (network.connectionType() != "WIFI"){
14
15             return {battery : $battery.level()}
16       }else return undefined
17 })
18 ranking(function(users){
19   return users.sort("battery").select();
20 })
21
22 //phase 3 : définition des objectifs de couverture
```

```
23 geoCoverage([[50.614291,3.13282],[50.604159,3.15239]],"500 m")
24 timeCoverage("30 min","1 H")
25 duplicate(2)
```

Listing 4.1 – Tâche de collecte : Observer la qualité réseau

Dans cette section, nous avons présenté le modèle de programmation proposé pour définir des campagnes de collecte supportant la collecte collaborative de données. Nous décrivons dans la suite de cette section comment ces campagnes sont exécutées par les nœuds de collecte.

4.5.2 *Coordonner l'exécution des tâches de collecte*

Dans cette section, nous décrivons le modèle et les algorithmes mis en œuvre pour supporter la coordination des tâches de collecte en fonction des propriétés de couvertures géographique et temporelle. Le principal défi qui doit être adressé pour mettre en œuvre une coordination optimisée est de permettre au nœud de collecte d'avoir une vision aussi précise que possible de la réparation géographique des nœuds mobiles disponibles. Cependant, en utilisant une approche naïve qui consisterait à reporter continuellement la position des dispositifs mobiles vers le serveur ne seraient pas efficace pour plusieurs raisons. Premièrement, cela pourrait engendrer un trafic trop important de messages échangés entre les nœuds mobiles et le serveur, spécialement si de nombreux mobiles sont impliqués dans la campagne. Deuxièmement, cela impliquerait une trop grande consommation énergique des nœuds mobiles, dû à l'activation constante de capteur comme le GPS. Et finalement, classifier les nœuds mobiles en fonction de leur distance les uns par rapport aux autres, en utilisant des algorithmes de *clustering* [31] par example, entraînerait un surcharge du serveur, spécialement si cette classification doit être effectuée en permanence sur un grand volume de données.

Pour adresser ce défi, nous avons adopté pour une approche de virtualisation de capteur, initialement proposée dans les réseaux de capteurs (WSNs). [10]. Typiquement, la virtualisation consiste à rajouter une couche logicielle abstraite, appelée couche virtuelle, composée d'un ensemble de capteurs virtuels, superposée à une couche physique. Un capteur virtuel est une entité logicielle autonome, responsable de gérer le flux de messages entre le serveur et un groupe de capteurs caractérisés par des propriétés communes (*par ex.* zone géographique, type de capteurs). Dans notre approche, l'idée

principale est de diviser la zone globale de collecte en plusieurs zone géographiques en fonction des propriétés de couverture géographique de la campagne de collecte. Pour chaque zone, nous attribuons un capteur virtuel qui sera responsable de coordonner l'exécution des tâches entre les nœuds mobiles situé dans sa zone. Dans ce contexte, les nœuds mobiles ont uniquement besoin de partager leur position lorsqu'ils entrent ou sortent de la zone gérée par un capteur virtuel.

Le composant responsable de l'exécution des campagnes de collecte communautaire et de la gestion des capteurs virtuels est le composite Recruitment :Collaborative (*cf.* figure 25). Ce composant est associé à la caractéristique Collaborative du point de variabilité Recruitement dans le modèle de caractéristiques présenté en sous-section 4.4.1.

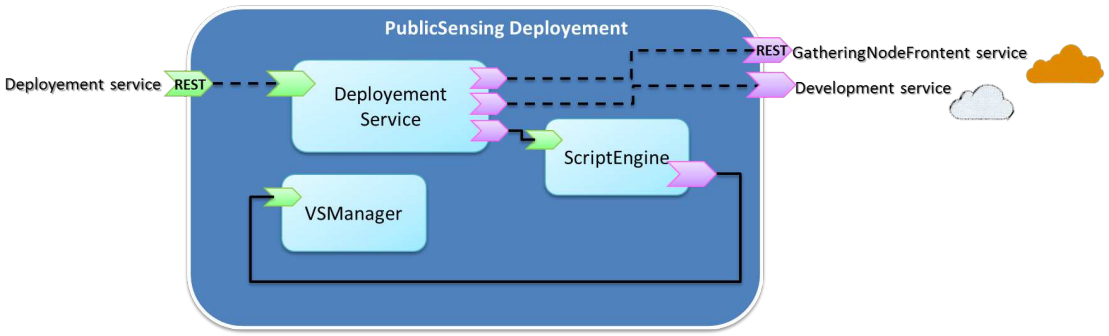

FIGURE 25 – Composant SCA responsable du recrutement collaboratif

Ce composite est constitué des trois composants : i) *ScriptEngine* responsable de l'exécution du script JavaScript définissant la campagne de collecte, ii) *VSManager* responsable de la création et la gestion des capteurs virtuels et iii) *Deployment* responsable d'interagir avec le serveur central pour déployer les scripts auprès des nœuds mobiles. L'exécution d'une campagne communautaire comprend trois phases : i) la phase de génération des capteurs virtuels, ii) la phase de connexion des nœuds mobiles avec les capteurs virtuels, iii) et la phase de coordination de l'exécution des tâches de collecte. Nous décrivons à présent l'ensemble de ces étapes.

Phase 1 : Génération des capteurs virtuels

L'objectif ici est de générer un ensemble de capteurs virtuels responsables de coordonner l'exécution des tâches de collecte entre les nœuds mobiles. La génération de ces capteurs virtuels est assurée par le composant VSManager. Chaque capteur virtuel est caractérisé par une propriété définissant la zone géographique incluse dans la zone

globale de collecte des données. La génération de ces capteurs est effectuée à partir de la méthode `geoCoverage(bound, coverage)` de l'interface de programmation, définissant la zone géographique globale de la collecte (paramètre `bound`), et la taille de la zone gérée par un capteur virtuel (paramètre `coverage`). À partir de ces paramètres, le composant VSManager génère et instancie un ensemble de capteurs virtuels, et les distribue pour couvrir l'intégralité de la zone globale de collecte (*cf.* figure 26). Typiquement, un capteur virtuel est un composite SCA disposant de deux services : composant RegistrationService et CoordinationService. Le premier service est dédié à l'enregistrement des nœuds mobiles. Ce service est exposé en tant que ressource REST, accessible à partir d'une URL générée à partir de la zone géographique dont il est responsable. Dans ce cas, les nœuds mobiles sont responsables de s'enregistrer auprès du capteur virtuel en fonction de leur position. Le deuxième service est responsable d'attribuer une tâche de collecte à un ou plusieurs nœuds mobiles qui se sont enregistrés au préalable. Nous détaillons ces différents points dans les section suivantes.

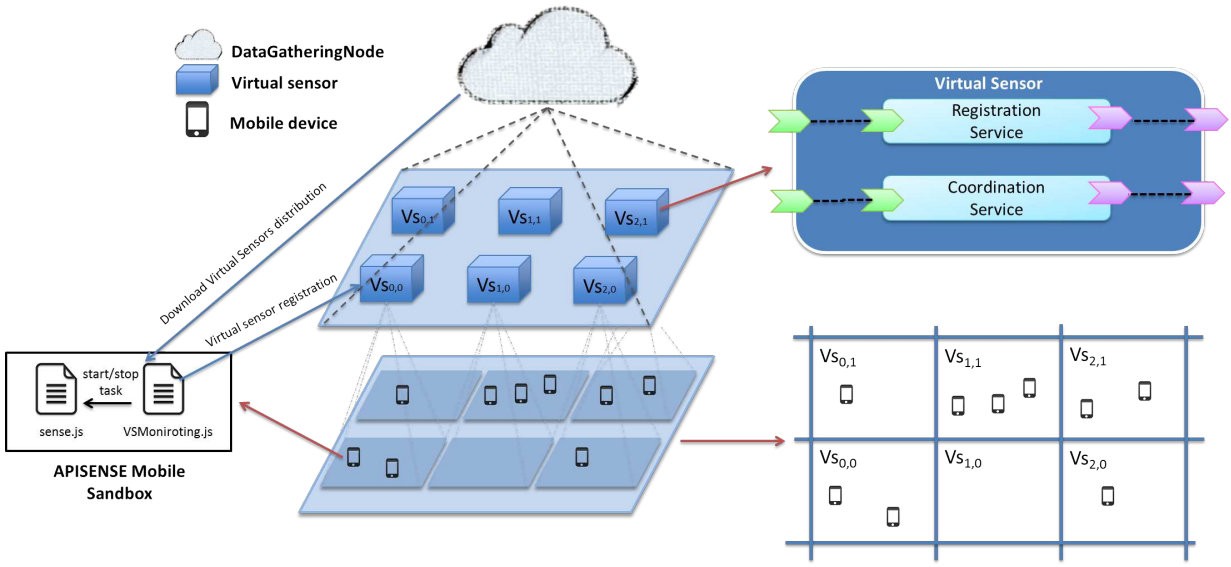

FIGURE 26 – Distribution des capteurs virtuels

Phase 2 : Connexion des appareils mobiles et des capteurs virtuels

La deuxième phase consiste à établir une connexion entre les nœuds mobiles et les capteurs virtuels. Pour établir cette connexion, les nœuds mobiles ont tout d'abord besoin de télécharger la tâche de collecte auprès du serveur central. Le déploiement de la tâche de collecte vers le serveur central, assurée par le composant DeploymentService,

suit le même procédé qui celui présenté en sous-section 4.4.4. Comme le montre la figure 26, la tâche de collecte comporte deux fichiers JavaScript : sense.js correspondant au code JavaScript responsable de la collecte des données (ligne 3-8 dans l'exemple décrit par le listing 4.1), et VSmonitoring.js. Dans un premier temps, seulement le fichier VSmonitoring.js est exécuté. Ce dernier script télécharge dans un premier temps l'ensemble des propriétés des capteurs virtuels générés (*c.-à-d.* zone géographique et l'URL du service d'enregistrement). À partir de ces propriétés, ce script observe continuellement la position du nœud mobile pour déterminer la présence du nœud mobile se situe dans une zone gérée par un capteur virtuel, et est responsable de l'enregistrement et du dé-enregistrement auprès du capteur virtuel associé.

En ce qui concerne la surveillance de la position du nœud mobile, nous utilisons un algorithme couramment utilisé dans la littérature pour déterminer si un dispositif se trouve dans une zone spécifique, tout en minimisant le coût énergétique. Typiquement, l'idée principale derrière est i) d'utiliser alternativement les différents mécanismes permettant de déterminer la position d'un utilisateur — entre le GPS fournissant une grande précision, mais consommant beaucoup d'énergie, et le réseau qui, au contraire, consomme moins d'énergie, mais est beaucoup moins précis —, ii) et la fréquence d'échantillonnage en fonction de la distance entre l'utilisateur et le capteur virtuel le plus proche de sa position actuelle.

Phase 3 : Coordonner l'exécution des tâches de collecte

Dans cette section, nous présentons l'algorithme utilisé par les capteurs virtuel pour coordonner l'exécution des tâches de collecte entre les nœuds mobiles enregistrés auprès d'un capteur virtuel. Le principal défi dans cette partie est de faire face à la connectivité imprévisible des nœuds mobiles. En effet, on ne peut pas partir du postula que tous les nœuds enregistrés sont réellement disponibles pour exécuter une tâche. De nombreux phénomènes peuvent empêcher les nœuds mobiles d'avertir un capteur virtuel qu'ils ne sont plus disponibles, comme la perte d'un signal réseau, un épuisement complet de la batterie ou tout simplement l'extinction du téléphone mobile par le participant. Pour prendre en considération ces phénomènes, l'idée principale de notre approche consiste à dupliquer le nombre de nœuds mobiles assignés à une tâche de collecte, et à intégrer une phase de demande d'exécution afin d'éviter d'avoir à assigner une tâche à un nœud qui n'a pas annulé son enregistrement au capteur virtuel.

L'algorithme (*cf.* algorithme 1) est déclenché périodiquement en fonction des propriétés de couverture temporelle qui peuvent être définies par l'interface de programmation avec la méthode `geoCoverage(duration,every)`.

Algorithm 1 Algorithme de coordination des tâches de collecte

Require:
connectedDevices : List of connected devices
activeDevices : List of activated devices
t : Time threshold
$Task(t_{start},t_{end})$: Temporal properties of the sensing task
$duplicate$: Maximum number of devices to assign the sensing task

if (size($activeDevices$) $<$ $duplicate$) **then**
  $candidates \Leftarrow connectedDevices - activeDevices$
  if notEmpty($candidates$) **then**
    $availableCandidates \Leftarrow \oslash$
    broadcastTaskRequestEvent($candidates$)
    repeat
      receive($Device(id, n)$)
      add($availableCandidates$,$Device(id, properties)$)
    until timeout t is reached
    for $(i = 0 \rightarrow (size(activeDevices) - duplicate)))$ **do**
      $device \Leftarrow$ ranking($availableCandidates$)
      $activeDevices \Leftarrow device + activeDevices$
      $availableCandidates \Leftarrow availableCandidates - device$
      broadcastTaskExecutionRequest($device, t_{start}, t_{stop}$)
    end for
  end if
end if

L'algorithme vérifie tout d'abord si le nombre de dispositifs mobile déjà assigné à une tâche (size($activeDevice$)) est inférieur à la propriété de duplication. Dans ce cas, nous définissons une liste candidate comme un ensemble de dispositifs connectés et n'exécutant pas actuellement une tâche de collecte. Pour tous les candidats, un message leur est envoyées avec la fonction de recrutement définie par l'interface de programmation (ligne 12-17 dans l'exemple du listing 4.1), — dans notre exemple, nous voulons recruter uniquement les nœuds mobiles disposant d'une connexion GSM —.

Le capteur virtuel attend ensuite un temps t la réponse des dispositifs mobiles. Les nœuds correspondant à la méthode de recrutement répondent ensuite au capteur virtuel avec un ensemble de propriétés indiquant leurs états actuels (*par ex.* niveau de

batterie, type de connexion de données). Ces nœuds sont ensuite ajoutés à une liste indiquant leurs disponibilités pour exécuter la tâche de collecte.

Ces nœuds sont ensuite classifiés par la méthode *ranking*. Cette méthode correspond à celle définie par l'interface de programmation (ligne 18-20 du listing 4.1), permettant de privilégier certains nœuds par rapport à leurs propriétés. Dans notre exemple, nous privilégions les noeuds mobiles disposant du niveau de batterie le plus élevé.

À la fin du processus, un nouveau message est envoyé aux nœuds mobiles leur demandant d'exécuter la tâche de collecte. Les noeuds mobiles recevant ce message démarrent l'exécution de la tâche de collecte jusqu'à la date définie par t_{end} ou lorsque les nœuds quittent de la zone gérée par le capteur virtuel. Cela permet de réattribuer la tâche de collecte à un autre nœud. À la fin de l'exécution, toutes les données collectées sont ensuite propagées directement sur le nœud de collecte.

4.6 CONCLUSION

APISENSE® propose un environnement facilitant la gestion de campagnes de collecte de données.

Pour supporter une grande diversité de campagne de collecte, nous avons proposé un modèle permettant de configurer un environnement serveur (*c.-à-d.* honey) responsable de l'exécution d'une campagne de collecte. Ainsi, ce modèle est utilisé pour configurer les services d'un nœud de collecte en fonction des spécificités des campagnes de collecte. Cette configuration comprends cinq points : i) configurer l'environnement de développement des tâches de collecte, ii) choisir un modèle de recrutement des participants, iii) choisir une série de traitements sur les données collectées avant leur insertion dans la base de données (*par ex.* anonymisation des données), iv) spécifier l'indexation des données dans la base de données, v) exposer des services pour exploiter les données collectées.

À partir de ce modèle, un nœud de collecte peut être ensuite généré et déployé par les utilisateurs vers une infrastructure qui peut être publique ou privée selon la nature des données collectées. Cela leur permet de garder un contrôle total sur l'infrastructure hébergeant les données collectées durant leurs campagnes.

APISENSE® se distingue également par son architecture décentralisée, permettant d'améliorer le passage à l'échelle du système. Son architecture est ainsi composée d'un ensemble de nœuds de collecte organisé autour du serveur central. Concernant le serveur central, son principal objectif est d'assurer le déploiement des tâches de

collecte soumises par les nœuds de collecte vers les participants de la plate-forme qui se sont enregistrée au préalable. Ce procédé a principalement deux avantages. Le premier est d'assurer une première couche d'anonymat des participants, dans le sens où tous les nœuds de collecte ne peuvent pas déployer les tâches de collecte sans passer par le serveur central. Le deuxième permet aux nouveaux utilisateurs de bénéficier d'un ensemble de participants déjà disponibles pour exécuter des tâches de collecte, leur évitant ainsi une longue période de recrutement.

Finalement, nous avons proposé une extension de APISENSE® dédiée à l'optimisation de l'exécution de campagne de collecte communautaire. L'optimisation proposée permet de coordonner l'exécution des tâches de collecte entre les dispositifs mobiles en fonction de propriétés de couvertures géographique et temporelle. Principalement, cette optimisation a pour objectif dans un premier temps de diminuer la consommation énergétique globale de tous les dispositifs impliqués dans la campagne, et également de diminuer la quantité de données propagées vers le nœud de collecte.

La description des contributions de ce manuscrit est à présent terminée. Dans les chapitres suivants, nous présentons les évaluations de la plate-forme APISENSE®.

Troisième partie

Évaluations

5

PRATIQUES CULTURELLES ET USAGES DE L'INFORMATIQUE CONNECTÉE

Sommaire

Dans cette partie, nous présentons les évaluations de la plate-forme APISENSE® en deux chapitres.

Dans le premier chapitre, nous présentons une campagne de collecte déployée auprès d'une centaine d'utilisateurs, réalisée au sein de l'étude PRATIC (Pratiques Culturelles et Usages de l'Informatique Connectée). Ce chapitre a pour objectif de montrer le gain en terme de coût de développement apporté par notre plate-forme, et également de montrer la faisabilité de notre approche dans un cas réel d'utilisation.

Le deuxième chapitre évalue le passage à l'échelle de APISENSE® pour la réalisation de campagne de collecte communautaire. Nous évaluons notamment le gain du modèle de recrutement collaboratif, permettant de faire un compromis entre l'énergie consommée par les dispositifs mobiles et la quantité de données générées tout en gardant une bonne couverture de collecte.

5.1 INTRODUCTION

Dans la partie précédente de ce manuscrit, nous avons présenté en intégralité API-SENSE®, la plate-forme résultante des travaux de cette thèse. Comme nous l'avons déjà évoqué, le principal objectif de cette thèse est d'ouvrir l'accès du *Mobile Crowd Sensing* à de nombreux acteurs privés ou académiques, en leur permettant de facilement développer et déployer leurs campagnes de collecte de données. Dans ce chapitre, nous présentons une campagne réalisée au sein d'une étude sociologique nommée PRACTIC [1](Pratiques Culturelles et Usages de l'Informatique Connectée), qui vise à comprendre l'usage des technologies numériques connectées en matière d'habitudes et de routines de consommations culturelle et médiatique. Cette étude a été réalisée par une équipe pluridisciplinaire, composée de deux ingénieurs en informatique pour le développement des scripts de collecte, et deux chercheurs en sciences de l'information et de la communication pour l'interprétation des données collectées. Par le biais de la description de cette étude, nous cherchons tout d'abord à montrer la faisabilité de notre approche pour être utilisée dans un cas réel d'utilisation, et plus particulièrement évaluer le gain en terme de cout de développement apporté par APISENSE®.

5.2 CONTEXTE ET OBJECTIF DE L'ÉTUDE PRACTIC

Le principe de l'étude PRACTIC est né de notre rencontre avec des chercheurs en sociologie, dans le cadre du projet de l'Observatoire Scientifique de l'Internet [2] initié par INRIA [3]. Un premier objectif de cette collaboration est de voir en quelle mesure les nouveaux mécanismes de collecte de données, comme le *Mobile Crowd Sensing*, peuvent augmenter les méthodes traditionnelles de la recherche numérique en sciences sociales. En effet, de nos jours, la nouvelle génération des terminaux intelligents comme les tablettes ou les *smartphones* sont fortement intégrés dans la vie sociale et culturelle des individus. Cette intégration représente de nouvelles opportunités pour les sciences sociales, qui peuvent ainsi collecter massivement et sur le long terme, des données comportementales des individus de manière non intrusive. De plus, cela nous a permis de confronter notre plate-forme à un déploiement réel, et d'être utilisée par des personnes tierces à notre équipe.

1. http ://beta.apisense.fr/practic
2. http ://metroscope.org/
3. www.inira.fr

L'étude PRACTIC est composée majoritaire de deux phases. La première est une phase de collecte de données auprès d'un ensemble de participants volontaires, combinant les collectes de données opportuniste et participative (*c.-à-d.* questionnaire). Dans le cadre de PRACTIC, la confrontation de la collecte opportuniste et participative permet de mesurer l'écart entre la représentation des individus sur l'utilisation de leur *smartphone* et leur pratique effective. La seconde phase comprend des traitements statistiques sur les données collectées qui aboutiront sur quelques entretiens individuels. Dans ce chapitre, nous décrivons uniquement la première phase de l'étude qui est directement en relation avec la plate-forme APISENSE®. Nous décrivons à présent le développement de cette phase de collecte.

5.3 DÉVELOPPEMENT DE L'ÉTUDE PRACTIC

La phase de collecte a entièrement été développée par APISENSE®, en utilisant son modèle de programmation (*cf.* chapitre 3 section 3.3) pour la description des données collectées sur les environnements mobiles, et un nœud de collecte déployé sur un serveur privé pour la persistance des données. Cette phase a été développée par un Ingénieur Jeune Diplômé (IDJ) en informatique, intégré pour les besoins de l'étude, qui n'avait aucune compétence particulière en programmation mobile au début du développement. L'application PRACTIC a d'abord connu plusieurs phases de test auprès d'un sous ensemble de participants. Ces différentes itérations ont permis d'identifier certains bugs, et d'ajouter des fonctionnalités non anticipées qui ont permis de consolider APISENSE® et l'enquête. Durant cette phase de test, l'ingénieur en question été responsable de faire la transition entre les exigences des sociologues, et les capacités offertes par notre plate-forme.

La figure 27 illustre les principales données collectées par l'application PRACTIC. Un aspect intéressant ici est qu'elle utilise un large éventail des fonctionnalités proposées par APISENSE®. La collecte de données comprend principalement trois parties différentes que sont la collecte opportuniste de données, la collecte participative et les retours utilisateurs, correspondant respectivement à plusieurs scripts de collecte. Nous décrivons à présent brièvement ces différents points, et comparons à la fin de cette section le gain en terme de coût de développement apporté par notre modèle de programmation.

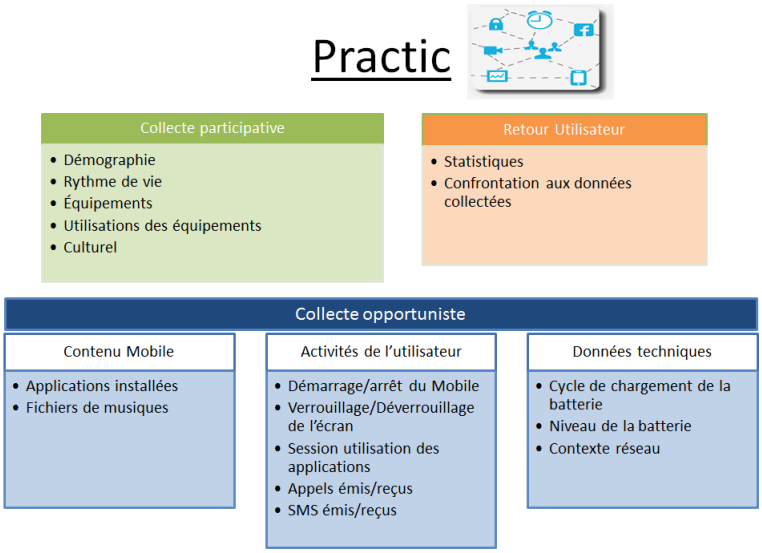

FIGURE 27 – Données collectées par l'étude PRACTIC

5.3.1 *Collecte opportuniste*

La première partie vise à collecter la diversité des rythmes de vie ainsi que des usages sociaux ou culturels des dispositifs mobiles. Toutes ces données sont collectées automatiquement par l'application sans nécessiter une intervention des participants. Typiquement, trois types de données sont collectées : *Contenu Mobile*, *Activité de l'utilisateur*, et *Contexte mobile*. Chaque type de données correspond à un script de collecte spécifique. Nous présentons brièvement un exemple simplifié des scripts de collecte développés.

Contenu Mobile

Le contenu mobile (*cf.* listing 5.1) vise à identifier des profils culturels des participants en fonction du contenu de leur appareil mobile. Dans le cadre de PRACTIC, la dimension culturelle est vue à deux niveaux : le type des applications mobiles installées (*c.-à-d.* jeux,divertissement, communication), les genres musicaux des participants. Le listing 5.1 décrit le script de collecte associé. La première partie du script (ligne 1-6) est déclenchée une seule fois lors de la première exécution du script. Dans cette partie, un ensemble de méta-données est collecté comprenant la liste des applications installées (nom, date d'installation, catégorie), et celle des fichiers musicaux présents (nom, artiste, format d'encodage, taille en Ko). La deuxième partie vise à identifier les cycles d'installation et de désinstallation des applications. Cette partie est réalisée

automatiquement lorsqu'un événement concernant l'installation ou la désinstallation est déclenchée par le système mobile.

```
1  $schedule.once(function(){
2        // store media files meta-data
3        $trace.add($media.getAudioFiles());
4        // store installed applications meta-data
5        $trace.add($app.getLaunchableApplications());
6  });
7
8  $app.onAppInstallEvent(function(app){
9        // store new installed or uninstalled application meta-data
10       $trace.add(app);
11 });
```

Listing 5.1 – Collecte du contenu mobile

Activité utilisateur

Cette partie vise à collecter les différents rythmes de vie et d'usages des dispositifs mobiles. Les usages sont principalement séparés en deux catégories correspondant aux usages natifs des téléphones mobiles (*c.-à-d.* appels et SMS) et les nouveaux usages introduis par la nouvelle génération des terminaux intelligents. Le listing 5.2 décrit l'utilisation de la façade `$phone` permettant de collecter des données sur les appels et les SMS émis et reçus. Pour des raisons de confidentialité, le contenu des appels et des SMS ainsi que l'identifiant des correspondants ne sont pas accessibles à partir des scripts. Seules la durée des communications ainsi que la taille des SMS sont disponibles.

```
1  $phone.onCallCompleted(function(call){ $trace.add(call) });
2
3  $phone.onSMS(function(sms){ $trace.add(sms) })
```

Listing 5.2 – Collecte des usages natifs des dispositifs mobile

La deuxième catégorie, illustrée par le listing 5.4 identifie les sessions d'utilisation des dispositifs. Celles-ci comprennent les sessions générales d'utilisations (écran allumé et éteint) et les sessions d'utilisation des applications. L'identification des sessions générales peut être effectuée par le biais de la façade `$screen` permettant d'intercepter les événements liés à l'écran du dispositif (lignes 2 et 22). Une session commence lorsque l'événement `onScreenOn` est déclenché. À la suite de cet événement, le script vérifie toutes les secondes (ligne 5) l'application exécutée en premier plan du dispositif (ligne 6), et collecte une session applicative lorsque l'application détectée est différente de la précédente (lignes 7-20). La session générale se termine lorsque l'événement

`onScreenOff` est déclenché, dans ce cas la vérification des applications exécutées en premier plan est annulée (ligne 30), et cela déclenche également une nouvelle collecte de données correspondant à la durée de la session générale (lignes 25-29).

```
1  var subscription,startSession,subscription,appName,startApp;
2  $screen.onScreenOn(function(screen){
3
4        startSession = screen.time;
5        subscription = $schedule.every("1 s",function(event){
6              var currentApp = $app.currentApplicationName();
7              if (appName != currentApp){
8
9                    appName = currentApp;
10                   startApp = event.time;
11
12                   // store application session
13                   $trace.add({
14                         event : "ApplicationSession",
15                         start : startApp,
16                         end  : event.time,
17                         name : appName
18                   });
19             };
20       });
21 });
22 $screen.onScreenOff(function(event){
23
24       // store screen session
25       $trace.add({
26             event : "ScreenSession",
27             start : startSession,
28             end  : event.time
29       });
30       subscription.suspend();
31 });
```

Listing 5.3 – Collecte des usages des applications mobile

Contexte mobile

Finalement, la dernière catégorie des données collectées permet de mettre en perspective l'utilisation des applications et leurs contextes (*cf.* listing 5.4). Cela permet d'identifier par exemple si certaines applications sont utilisées uniquement lorsque le participant se trouve à son domicile ou sur son lieu de travail,ou encore d'observer si la qualité du réseau ou le niveau de batterie influe sur l'utilisation de certaines applications. Cette catégorie comprend les cycles de chargement et le niveau de la batterie (lignes 10-12), le contexte réseau (ligne 7-9) constitué du type de connexion

de données (*c.-à-d.* Wi-Fi, 3G/4G), et de la force du signal réseau GSM (lignes 1-6). Pour déterminer si l'utilisateur est en situation de mobilité, le choix a été fait de collecter uniquement les identifiants des antennes cellulaires, qui permettent d'avoir une approximation de la localisation du participant sans entraîner une consommation énergétique supplémentaire de l'application mobile [65].

```
1  $telephony.onSignalStrengthChanged(function(signal){
2        $trace.add({
3              level : signal.level,
4              cellId : $telephony.cellId()
5        });
6  });
7  $network.onNetworkStateChange(function(networkEvent){
8        $trace.add(networkEvent);
9  });
10 $battery.onBatteryStateChange(function(batteryEvent){
11       $trace.add(batteryEvent);
12 });
```

Listing 5.4 – Collecte des usages des applications mobiles

5.3.2 *Collecte participative*

Le questionnaire vise à collecter des données déclaratives sur l'équipement, les usages et le rapport à la vie privée des participants. Il est composé de 152 questions, dont certaines ne sont pas obligatoires. Par exemple, dans la partie des usages, le fait de ne pas posséder une tablette par exemple permet de ne pas répondre à un sous-groupe de questions. Entre 20 et 30 minutes sont nécessaires pour répondre intégralement au questionnaire. Les questions sont organisées au sein de quatre grandes parties : à propos de vous (16 questions), équipements (36), pratiques culturelles et usage de l'informatique connectée (60) et publicité en ligne et privée (19). Le questionnaire peut être complété à tout moment par les participants à partir de leurs dispositifs. Le listing 5.5 montre une très brève partie du questionnaire développé. Typiquement, les réponses du questionnaire vont être croisées par les sociologues avec les données collectées automatiquement par l'application.

```
1  var survey = $survey.create("PRACTIC");
2  // ...
3  var q88 = survey.close("Quel est le lecteur musical que vous utilisez le plus ?",[
4   "Chaine Hi-Fi / Platine Vinyle",
5   "Baladeur/MP3",
```

```
6   "Smartphone",
7   "L'ordinateur (musique enregistrée sur le disque dur ou CD)",
8   "La radio",
9   "La television (chaines musicales comme MTV)",
10  "Autre lecteur de musique"
11  ]);
12  var q88b = survey.open("Quel autre lecteur utilisez-vous ?");
13  var q89 = survey.close("Quel est votre maniére d'écoute principale avec votre lecteur ?",[
14   "Casque audio",
15   "Enceinte/Ampli",
16   "Sans rien"
17  ]);
18  q88.then(function(response){
19
20        if (response == "Autre lecteur de musique"){
21              return  q88b;
22        }
23        else return q89;
24  });
```

Listing 5.5 – Extrait du questionnaire PRACTIC

5.3.3 *Retour utilisateur*

La dernière partie des scripts de collecte vise à donner un aperçu aux participants des données collectées. Typiquement, l'objectif de cette partie est d'agir en toute transparence vis-à-vis des participants sur les données qui sont collectées sur leurs dispositifs, et d'apporter un aspect ludique à l'application. L'idée est de permettre aux participants de confronter leur perception de l'utilisation de leur dispositif avec leur usage réel. La figure 28 montre un exemple de retour utilisateur sur la durée totale d'utilisation de son appareil. Cette partie a été développée intégralement en HTML/JavaScript.

5.3.4 *Discussions*

Dans cette section, nous avons décrit l'ensemble des scripts de collecte qui a été nécessaire dans le cas de l'étude PRACTIC. Nous allons maintenant identifier le gain apporté par APISENSE® pour le développement de ce cas d'étude. L'application mobile APISENSE®, responsable du téléchargement, de l'exécution des scripts et de la propagation des données collectées a été développée en utilisant la version 2.3 du SDK de Android. L'application APISENSE® fournit de nombreuses abstractions qui simplifient

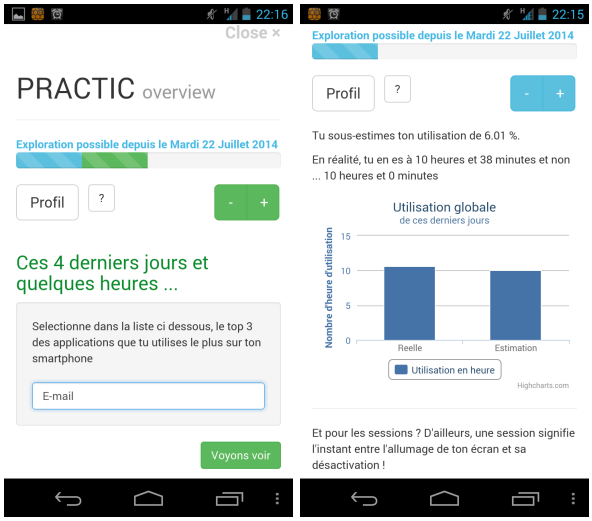

FIGURE 28 – Exemple de retour utilisateur PRACTIC

le développement d'une application de collecte de données. La première permet tout d'abord d'avoir une abstraction complète du SDK Android pour accéder aux données fournies par Android, et intercepter les différents événements internes au dispositif mobile grâce aux façades. Le tableau 12 récapitule les façades qui ont été utilisées pour développer PRACTIC, et leur nombre de lignes de code Java qui ont été nécessaires à leur développement. Au total, ces façades comptent approximativement 6000 lignes de code. Pour les besoins de l'étude, en collaboration avec l'ingénieur responsable du développement de PRACTIC, nous avons dû intégrer deux nouvelles façades pour lister les fichiers musicaux (façade `$media`) et pour observer les évènements liés à l'écran d'un dispositif (façade `$screen`). Ces façades peuvent être à présent disponibles pour le développement de nouveaux scripts de collecte. L'application APISENSE® dispose également d'un ensemble de services responsables de l'exécution des scripts de collecte en tâche de fond,d'interagir avec le serveur central d'APISENSE® pour le téléchargement et de la mise à jour des scripts de collecte, d'interagir avec les nœuds de collecte des scientifiques pour propager automatiquement les données collectées, et de contrôler les ressources consommées par les scripts de collecte (*cf.* chapitre 3 section 3.4). Cette partie comporte approximativement plus de 20000 lignes de code. Pour finir, l'application mobile APISENSE® possède également une interface graphique générique destinée aux participants (6000 lignes de code).

Façade	Description	LOC
$survey	Génération et publication d'un questionnaire	1277
$feedback	Intégration de visualisation HTML/JavaScript	2144
$network	Observation du contexte réseau	259
$telephony	Observation du réseau cellulaire	400
$phone	Observation des communications	600
$battery	Observation des états de la batterie	158
$app	Observation des applications	362
$trace	Collecte et propagation des données	376
$screen (nouvelle)	Observation de l'écran	235
$media (nouvelle)	Liste du contenu	210

TABLE 12 – Façade utilisée pour les besoins de l'application PRACTIC

La figure 29 met en évidence le gain en terme de lignes de codes obtenu grâce à APISENSE® par rapport aux scripts développés. Au total, seulement 7 % de codes spécifiques ont eu besoin d'être développés pour l'étude PRACTIC. Les principaux efforts de développement résident dans la description des 150 questions et leurs enchaînements (760 lignes) et dans les interfaces graphiques effectuant un retour sur les données obtenues (1600 lignes). Dans la section suivante, nous présentons le protocole de déploiement de l'application.

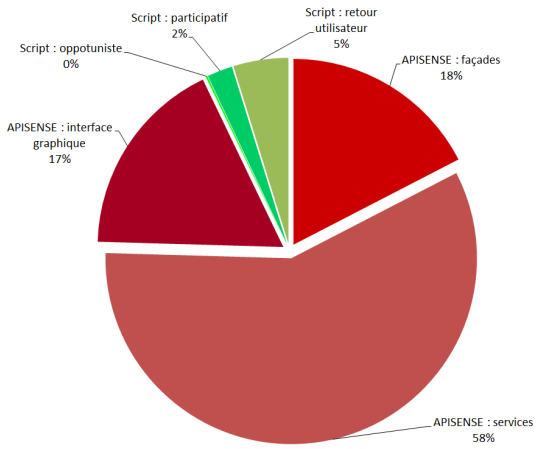

FIGURE 29 – Gain en terme de lignes de code obtenu grâce à APISENSE® pour le développement de l'étude PRACTIC

5.4 DÉPLOIEMENT DE L'ÉTUDE PRACTIC

Après avoir présenté dans son ensemble les scripts de collecte de l'étude PRACTIC, nous présentons dans cette section son protocole de déploiement et discutons de la participation à cette étude.

5.4.1 *Protocole du déploiement*

Pour la mise en place de l'étude PRACTIC, un nœud de collecte a tout d'abord été déployé sur un serveur privé des scientifiques. Le nœud de collecte a été configuré pour utiliser un modèle de recrutement individuel, pour indexer les données collectées par utilisateur et pour utiliser un système de classement à point (*cf.* chapitre 4 section 4.4.1) afin de maximiser la participation des utilisateurs. Pour que les données collectées soient pleinement exploitables par les sociologues, ils ont besoin qu'elles soient collectées au minimum pendant 14 jours par participant, et que celui-ci ait répondu au questionnaire de l'enquête. Dans ce contexte, les points sont attribués de la manière suivante :

- 50 points pour les participants ayant installé l'application sur une tablette et un *smarpthone*, remplis le questionnaire et ayant généré plus de 14 jours de données.
- 20 points pour les participants qui ont installé sur un seul appareil (smartphone ou tablette), avec le questionnaire rempli et toujours un minimum de 14 jours de données générées.
- 1 point par jour de données générées
- 2 points si le questionnaire est rempli

Des lots ont été prévus pour récompenser les meilleurs participants comprenant une tablette tactile, un *smarpthone*, une liseuse ainsi que des disques durs externes et des clés USB.

Le nœud de collecte a également été utilisé pour le développement des scripts de collecte ainsi que leurs déploiements auprès d'un serveur central que nous avions au préalablement déployé sur un service Microsoft Azure[4]. Pour des raisons éthiques et légales, l'étude PRACTIC a été soumise et validée par la Commission National de l'Informatique et des Libertés[5] (CNIL).

4. http ://azure.microsoft.com/fr-fr/
5. http://www.cnil.fr/

Les participants intéressés à participer à l'étude PRACTIC doivent tout d'abord télécharger et installer l'application mobile APISENSE® publiée sur le site internet APISENSE® [6] en flashant un *QR code*. À partir de l'application, les participants pourront alors s'enregistrer auprès de l'étude PRACTIC. L'enregistrement déclenchera automatiquement le téléchargement et l'exécution des scripts de collecte après la validation d'une charte de confidentialité. Un identifiant unique est généré et accordé à chaque participant lors de son inscription. Ce n'est qu'en acceptant de communiquer cet identifiant avec les chercheurs qu'ils pourront être identifiés. Nous présentons dans la suite de cette section le niveau de participation à PRACTIC, et discutons des premiers retours de l'expériences.

5.4.2 *Participation*

L'étude PRACTIC s'est déroulée du 10 MARS au 19 Avril 2014. La figure 30 montre le nombre d'utilisateurs inscrits par jour, ainsi que l'accumulation du nombre d'inscriptions durant l'étude. Au total, 88 participants ont été recrutés durant l'étude.

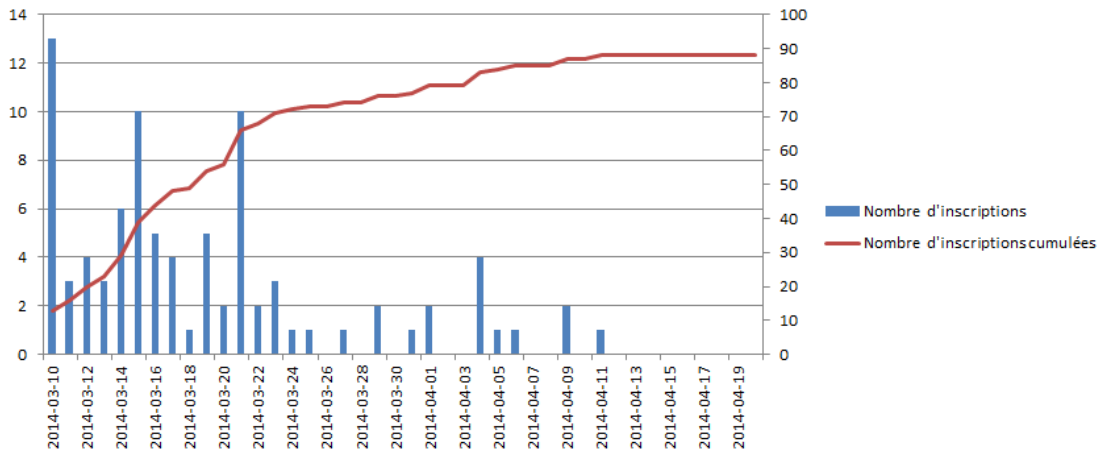

FIGURE 30 – Nombre d'inscriptions par jour à PRACTIC

Un aspect intéressant des participants recrutés est la diversité de leurs équipements (*cf.* figure 31), ce qui nous a permis de tester APISENSE® sur 45 modèles de smartphones différents.

Outre le nombre d'inscriptions, le succès de l'étude repose principalement sur le taux de participation des utilisateurs tout au long de l'étude. Pour que les données

6. http://www.apisense.fr

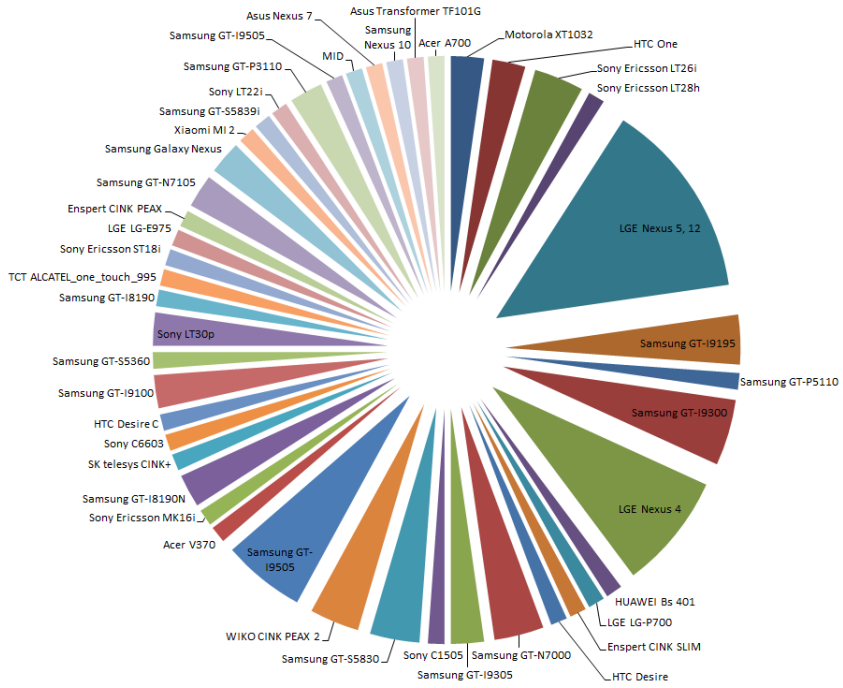

FIGURE 31 – Diversité des équipements mobiles

d'un participant soient exploitables par les chercheurs, il est nécessaire qu'elles aient été collectées au moins sur une période de 14 jours, et que celui-ci ait répondu au questionnaire de l'application.

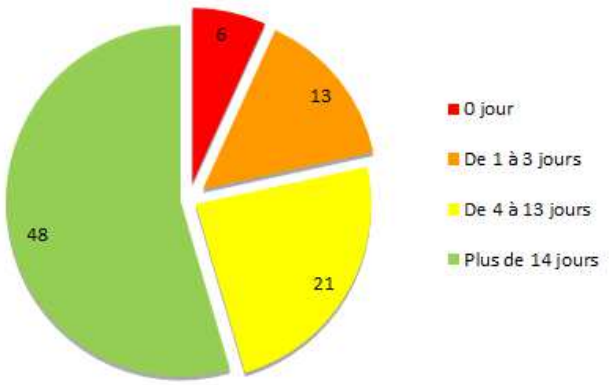

FIGURE 32 – Taux de participation à la collecte opportuniste

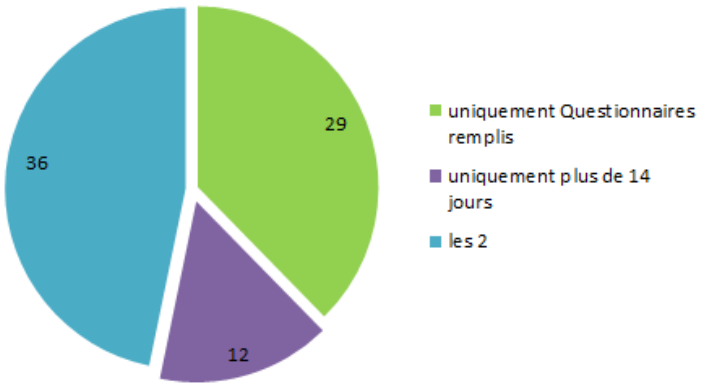

FIGURE 33 – Taux de participation à la collecte participative et opportuniste

Sur l'ensemble des participants, comme la montre la figure 32, 48 ont collecté des données plus de 14 jours, 21 de 4 à 13 jours, 13 de 1 à jour 6 qui n'ont collecté aucune donnée. Les participants ayant répondu au questionnaire sont au nombre de 25, et les participants ayant à la fois répondu au questionnaire et collecté au moins 14 jours de données sont au nombre de 12 (*cf.* figure 34). Ces derniers représentent les cas d'étude concrets pour les scientifiques. Bien que ce nombre soit inférieur à celui escompté au début de l'étude, un point positif des premiers résultats obtenus permet de valider que les dispositifs sont bien au cœur des pratiques de certains participants. La figure 34 par exemple représente le nombre de sessions applicatives, ainsi que leurs durées, sur une période de 40 jours Cette courbe met bien en évidence les différents rythmes des usages d'un participant en particulier, qui utilise son dispositif tout au long de la journée, et qui intensifie son utilisation en fin de journée.

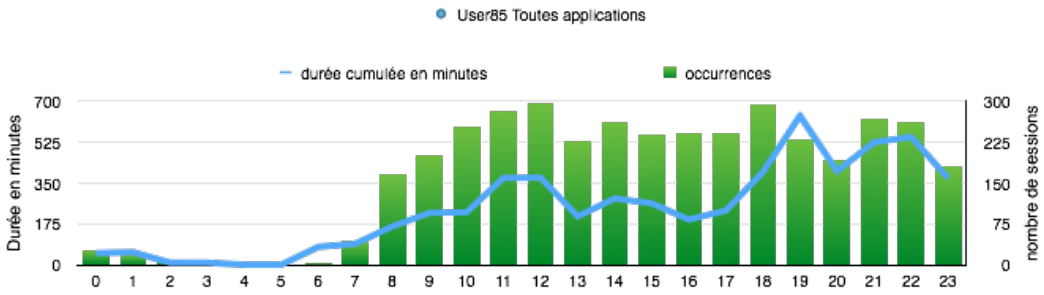

FIGURE 34 – Représentation des sessions d'allumage de l'écran du terminal d'un participant cumulées au cours de 40 jours

5.5 CONCLUSION

Dans ce chapitre, nous avons présenté un cas réel de campagne de collecte de données développée et déployée avec APISENSE® au sein d'une étude sociologique. Nous avons montré que APISENSE® permettait réduire le temps de développement d'une campagne de collecte en proposant un modèle de programmation permettant de s'abstraire complètement des technologies mobiles. Dans le cas de cette étude, le modèle de programmation a permis d'économiser plus de 90% de code nécessaire à son développement. La campagne développée a été déployée auprès de 88 utilisateurs sur une période d'un mois et demi.

6

ÉVALUATION DU MODÈLE COLLABORATIF

Sommaire

6.1 INTRODUCTION

Dans ce chapitre, notre objectif est de montrer la validité du modèle de recrutement collaboratif présenté dans le chapitre 4 en section 4.5, page 113 qui permet de coordonner l'exécution des tâches de collecte entre les dispositifs mobiles. Nous rappelons que le but de cette coordination est dans un premier temps de répartir les charges énergétiques entre les dispositifs mobiles durant la campagne de collecte, puis dans un second temps de diminuer la quantité de données propagées vers les nœuds de collecte en limitant la propagation de données dupliquées.

La validité du modèle est évaluée sur quatre critères : i) la quantité de données propagées sur les nœuds de collecte, ii) la quantité d'énergie utilisée par les dispositifs mobiles ainsi que iii) le surcoût induit par notre approche pour effectuer la coordination des tâches, iv) la couverture géographique des données obtenues.

Pour valider notre modèle, nous avons mené une expérience et développé un environnement de simulation que nous détaillons dans la section suivante.

6.2 MISE EN PLACE DE L'EXPÉRIENCE

Pour cette évaluation, nous reprenons l'exemple de l'application présentée dans la section 4.5.1 page 116 qui est une version simplifiée d'une collecte de données exposée

dans LiveLab [61]. L'objectif de cette application est d'utiliser les dispositifs mobiles pour mesurer la qualité des réseaux GSM en milieu urbain. Le listing 6.1 décrit la tâche de collecte associée à cette application qui consiste à exécuter 10 *Pings* réseaux (Packet INternet Groper) toutes les trente secondes et à collecter la latence moyenne des *Pings* effectués ainsi que la position du dispositif obtenue avec par GPS.

```
1 $location.onLocationChanged({period : "30s", provider : "GPS"},function(event) {
2       $trace.add({
3             loc: [event.latitude, event.longitude],
4             latency : $network.ping(10,"http://...").average});
5 })}
```

Listing 6.1 – Tâche de collecte : Mesure de la latence réseau

Pour évaluer notre approche, nous avons simulé le recrutement des dispositifs suivant un modèle où tous les dispositifs exécutent la tâche de collecte indépendamment les uns des autres, et suivant le modèle collaboratif, où les capteurs virtuels coordonnent l'exécution des tâches de collecte entre les dispositifs mobiles (*cf.* 4.5.2, page 119).

Pour une évaluation réaliste et à grande échelle, nous utilisons des traces de mobilités réelles provenant de 10000 taxis équipés de GPS dans la ville de Pékin en Chine [67]. Ces données ont été collectées sur une période d'une semaine allant du 2 au 8 février 2008. Afin d'utiliser ces données, nous avons développé un simulateur illustré par la figure 35. Le simulateur comprend trois composants : un MANAGER, un ensemble d'AGENTS et un nœud de collecte (DATAGATHERING NODE).

AGENTS : Les agents sont responsables de générer le trafic réseau simulant les interactions entre les dispositifs mobiles et le nœud de collecte. Ces interactions sont simulées à partir d'un fichier de mobilité comprenant une série de tuples (*date*, *latitude*, *longitude*) représentant les déplacements d'un taxi à travers le temps. Pour l'expérience, nous avons implémenté deux types d'agents : i) les agents individuels et ii) les agents collaboratifs. Les agents individuels exécutent la tâche de collecte en continu tout au long de l'expérience, c'est-à-dire qu'ils collectent toutes les 30 secondes leur position selon le fichier de mobilité, et propagent l'ensemble des données collectées sur le nœud de collecte toutes les 5 minutes. Quant aux agents collaboratifs, ils s'enregistrent auprès des capteurs virtuels selon leur position à un instant t, et exécutent la tâche de collecte lorsqu'ils ont reçu une demande d'exécution de la part du capteur virtuel. Pour assurer une montée en charge du simulateur, les agents sont répartis auprès de plusieurs machines

virtuelles hébergées sur la plate-forme *Microsoft Windows Azure* [1]. Des machines virtuelles de taille *Small (A1)* [2] sont utilisées, correspondant à un processeur à 1 coeur, 1,75 Go de mémoire et une carte réseau de 100 Mbit/s.

DATAGATHERING NODE : Pour la simulation, nous avons déployé un nœud de collecte (*cf.* section 4.4, page 100) sur une machine virtuelle hébergée sur *Microsoft Windows Azure*. Une machine virtuelle de taille *Medium (A2)* est utilisée, correspondant à un processeur à 2 coeurs, 3,5 Go de mémoire et une carte réseau de 200 Mbit/s.

MANAGER : Ce dernier composant est responsable de répartir les fichiers de mobilité auprès des agents déployées sur les machines virtuelles, et de lancer le début de la simulation afin de synchroniser les agents.

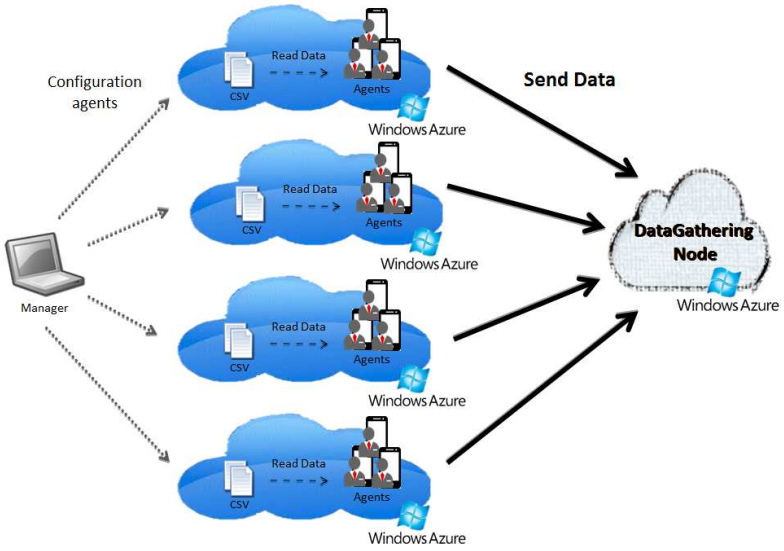

FIGURE 35 – Architecture du simulateur de traces de mobilité

Pour les simulations présentées dans la suite de ce chapitre, nous utilisons les données du 3 février 2008 représentant la journée où le plus de données ont été produites. Les tâches de collecte sont déployées dans une zone de collecte de 10 km^2 situés dans le centre-ville de Pékin, durant une durée de 30 minutes.

1. http://azure.microsoft.com
2. http://azure.microsoft.com/en-us/pricing/details/virtual-machines

6.3 RÉSULTATS

Dans cette section, nous présentons les résultats obtenus à travers les différentes expériences menées grâce au simulateur développé. Nous discutons des résultats en fonction de la quantité, de la couverture géographique des données collectées ainsi que de l'énergie utilisée par les dispositifs mobiles durant les expériences et du surcoût engendré par notre approche. Pour évaluer les gains de notre modèle de recrutement, nous avons déployé, à travers le simulateur, la tâche de collecte présentée dans le listing 6.1 suivant trois modèles appelé *individual*, *coll(1000 m)*, et *coll(500 m)*.

individual : dans ce modèle, les dispositifs exécutent la tâche de collecte indépendamment les uns des autres. Ils obtiennent une nouvelle position du GPS toutes les 30 secondes, et exécutent la tâche de collecte lorsque leur position se situe dans la zone de collecte définie.

coll(1000 m) : dans ce modèle, l'exécution de la tâche de collecte est coordonnée par les capteurs virtuels déployés dans le nœud de collecte. L'objectif de couverture est fixé tous les 1000 mètres dans la zone de collecte avec 5 dispositifs attribués à la tâche (*cf.* section 4.5.1).

coll(500 m) : ce modèle est identique au précédent sauf que nous avons fait varier l'objectif de couverture tous les 500 mètres dans la zone de collecte. Cela nous permet d'étudier l'impact de la couverture définie sur la quantité de données collectées et sur la consommation énergétique des dispositifs mobiles.

6.3.1 *Quantité de données et couverture géographique*

Tout d'abord, nous comparons la quantité des données collectées. Les courbes illustrées par la figure 36 comparent la quantité de données collectées (en kilobyte) pour les trois tâches de collecte déployées en fonction du nombre de dispositifs. La première tâche, illustrée par la courbe *individual*, représente la quantité de données lorsque les dispositifs collectent individuellement la tâche de collecte. Les deux suivantes, illustrées par les courbes *coll(1000 m)* et *coll(500 m)* représentent les tâches de collecte déployées en utilisant le modèle collaboratif. Nous avons fait varier entre 500 et 10000 le nombre de dispositifs simulés durant cette expérience. Comme le montre cette figure, lorsqu'aucune collaboration n'est effectuée, la quantité de données générées augmente linéairement, allant jusqu'à 12000 KB de données produites pour 10000 dispositifs. En effectuant une collaboration, la quantité de données se stabilise autour des 6000 KB pour

un objectif de couverture de 500 m^2 et 2500 KB pour un objectif de 1000 m^2. Pour 10000 dispositifs, cela représente respectivement une diminution de 50% et 80% de quantité de données propagées vers le serveur. Ce gain s'explique, car lorsque suffisamment de dispositifs se trouvent dans une même zone (*par ex.* plus de 5 dispositifs dans une zone de 500 m^2 pour la tâche *coll(500 m)*), le capteur virtuel associé à cette zone attribut seulement à 5 dispositifs la tâche de collecte, évitant ainsi aux autres dispositifs de la même zone d'exécuter la tâche de collecte.

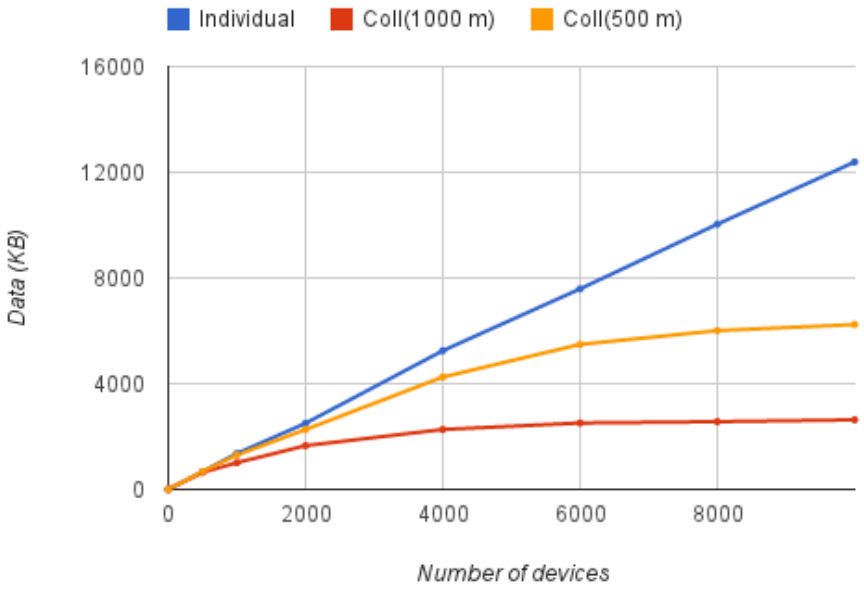

FIGURE 36 – Comparaison de la quantité de données collectées en fonction du nombre de dispositifs mobiles et de l'objectif de couverture géographique

Cependant, pour valider notre approche, il est essentiel de comparer la quantité de données avec la couverture réellement obtenue. Pour cela, nous avons mené une deuxième expérience impliquant 10000 dispositifs mobiles, et nous avons observé la couverture géographique obtenue à travers différentes périodes de la journée par rapport à la quantité de données collectées. La couverture est définie par le pourcentage de zone couverte par les données collectées, soit 400 zones de 500 m^2 dans la zone de collecte d'une surface de 10 Km^2. La figure 37 illustre le résultat de cette expérience. Dans cette figure, les courbes bleu et verte représentent respectivement la couverture obtenue par les approches individuelle (*individual-SP*) et collaborative (*coll(500 m)-*

SP), et les colonnes rouge (*individual-Data*) et jaune (*coll(500 m)-Data*) la quantité de données collectées par ces deux approches. Comme le montre cette figure, la couverture des données obtenue varie tout au long de la journée, pour atteindre un pic à 17 H représentant une forte mobilité des taxis dans la zone de collecte. Ce que nous pouvons observer d'intéressant, c'est que les couvertures obtenues par ces deux approches sont relativement équivalentes, bien que la quantité de données collectées par l'approche collaborative soit toujours inférieure à celle de l'approche individuel. La différence des couvertures obtenues entre les deux approches est due à la mobilité des taxis. En effet, si un taxi se déplace à grande vitesse, celui-ci peut traverser une zone avant que le capteur virtuel associé n'est le temps de lui attribuer une tâche. Cependant, dans cette simulation, la perte de la couverture obtenue ne varie pas au delà des 2%.

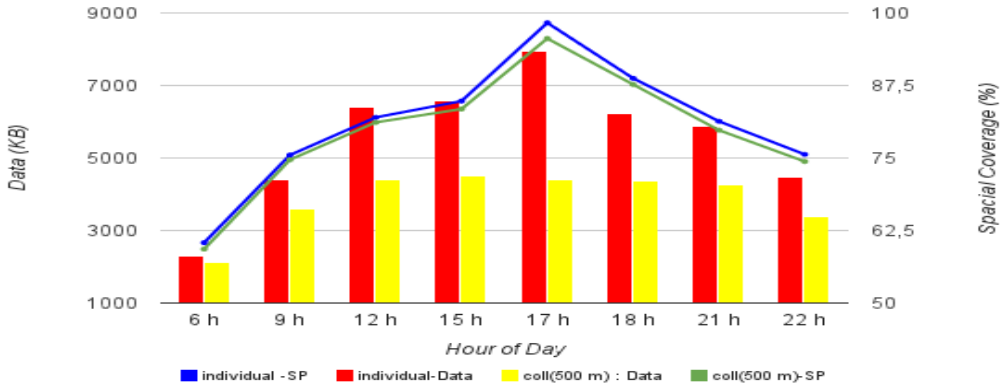

FIGURE 37 – Comparaison de la couverture géographique selon différentes périodes de la journée

À travers ces deux expériences, nous avons montré que l'approche collaborative que nous avons proposée permet de diminuer la quantité de données propagées vers le serveur tout en gardant une couverture presque similaire à une approche classique. Nous allons maintenant évaluer les gains de cette approche sur la consommation énergétique des dispositifs mobiles.

6.3.2 *Consommation énergétique*

Pour évaluer l'énergie consommée par les dispositifs mobiles liée au trafic réseau et à l'acquisition de la position par le GPS, nous utilisons deux modèles énergétiques très référencés dans la littérature.

Le premier modèle, proposé par Balasubramanian et al. [5], permet de déterminer l'énergie liée à la transmission de données par des réseaux sans fil. Dans le contexte de cette évaluation, nous assumons que toutes les données sont transmises par un réseau 3G. Ce modèle divise l'énergie consommée en trois parties. La première est liée à l'activation de l'interface de communication évaluée à 3,5 Joules. La deuxième est liée à la transmission des données évaluée à 0.025 Joule/KB. Et la dernière est liée au temps que l'interface de communication soit désactivée après la transmission des données. Ce temps est évalué à 12,5 secondes et l'interface consomme 0,62 Joule/s jusqu'à ce qu'elle soit désactivée. Dans le cadre où deux transmissions de données sont effectuées en dessous des 12,5 secondes, l'énergie consommée par cette dernière partie est négligée.

Pour le deuxième modèle, nous utilisons celui proposé par Lin et al. [43], qui évalue l'énergie liée à l'acquisition périodique d'une position par le GPS à 1,4 Joules par position obtenue.

Les courbes illustrées par la figure 38 comparent la consommation énergétique des trois tâches de collecte déployées dans la première expérience de la section précédente par rapport à la concentration de dispositifs mobiles dans la zone de collecte. L'énergie est donnée en Joule et correspond à la moyenne énergétique consommée par l'ensemble des dispositifs mobiles.

Comme attendue, la consommation énergétique de la tâche individuelle (*individual*) est indépendante de la concentration des dispositifs mobiles, et reste constante autour des 600 Joules en moyenne par dispositif. En revanche, les tâches collaboratives (*coll(1000 m)* et *coll(500 m)*) bénéficient de cette concentration croissante en évitant de collecter des données non nécessaires pour atteindre les objectifs de couverture, et ainsi réduire la consommation énergétique moyenne par dispositif. Pour *coll(1000 m)*, le gain sur la consommation énergétique des dispositifs varie entre 43% pour 500 dispositifs et 82% pour 10000 dispositifs, tandis que pour *coll(500 m)*, ce gain varie entre 40% et 70%.

Afin de mieux comprendre où le gain énergétique est effectué, les graphiques de la figure 39 montrent la répartition des charges énergétiques consommées durant l'expérience pour les trois tâches *individual (a)*, *coll(1000 m) (b)* et *coll(500 m) (c)*. Dans cette figure, la partie bleue (*Localisation*) des colonnes représente la consommation

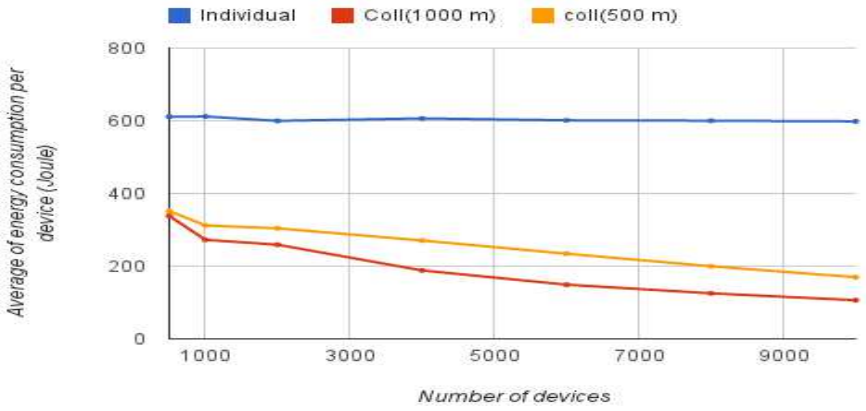

FIGURE 38 – Comparaison des moyennes de la consommation énergétique en fonction de la concentration de dispositifs mobiles dans la zone de collecte

causée par l'acquisition d'une position par le GPS, la partie rouge (*Sensing Task*) à l'exécution des Pings réseaux (*cf.* listing 6.1 ligne 4) et la partie orange (*VS Messages*) aux communications avec les capteurs virtuels.

Comme le montre cette figure, le gain de la collaboration sur la consommation énergétique est particulièrement liée à l'acquisition des positions par le GPS. En effet, dans cette approche, les dispositifs ont seulement besoin d'obtenir une position pour déterminer à quel capteur virtuel ils doivent s'enregistrer, permettant à ces dispositifs de réduire la fréquence de l'acquisition d'une nouvelle position si leur vitesse est relativement réduite ou en position statique (*cf.* section 4.5.2, page 121). Cela représente dans cette expérience un gain respectif de 86% et 80% pour les tâches *coll(1000 m)* et *coll(500 m)*. Dans ce contexte, une grande perte énergétique du modèle de recrutement individuelle s'explique par le fait que même les dispositifs ne se situant pas dans la zone de collecte sont obligés de continuellement activer leur GPS afin de déterminer s'ils sont dans la zone étudiée ou non. En ce qui concerne l'énergie consommée par l'exécution des *Pings* réseaux, le gain reste limité lorsque peu de dispositifs sont impliqués dans l'expérience. Pour le modèle coll(1000 m), le gain est seulement de 4% et devient négatif pour *coll(500 m)*. Ceci s'explique par le surcoût énergétique engendré par les communications des dispositifs avec les capteurs virtuels. Néanmoins, le gain énergétique s'accroît lorsque plus de dispositifs sont impliqués pour atteindre 81% pour *coll(1000 m)* et 55% pour le modèle *coll(500 m)*.

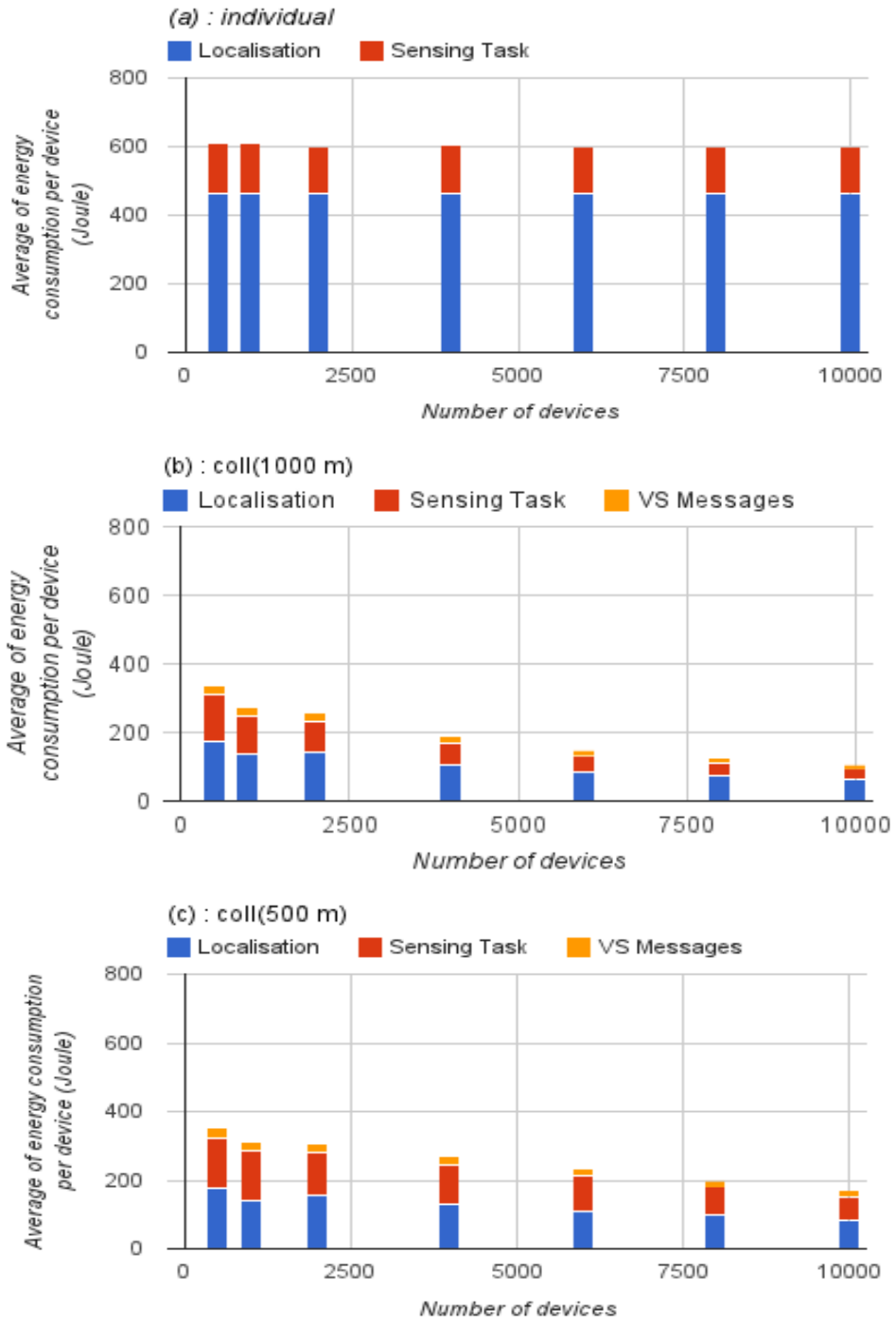

FIGURE 39 – Répartition des charges énergétiques suivant les approches : (a) individuelle, (b) collaborative avec une objectif de couverture de 1000 m ^{2}, (c) collaborative avec un objectif de couverture de 500 m^2

Comme attendu, ces chiffres montrent que le gain énergétique lié à la coordination des tâches varie selon la concentration des dispositifs mobiles dans la zone de collecte et également des objectifs de couvertures définies. Cependant cela révèle également les limitations du modèle évalué dans ce chapitre. En effet, dans le contexte où la concentration des dispositifs mobiles est trop faible pour permettre une coordination des tâches, notre modèle consommerait plus qu'un modèle classique à cause des communications nécessaires avec les capteurs virtuels. D'autre part, notre modèle reste limité si l'objectif de couverture nécessaire pour la campagne de collecte doit être très élevé (*par ex.* tous les 50 mètres). Cela engendrerait un trafic réseau trop important, car les dispositifs seraient sens cesse en train de communiquer avec les capteurs virtuels, spécialement dans un cas de forte mobilité. Dans ce contexte, la solution proposée ici reste principalement efficace pour les campagnes de collecte nécessitant une couverture géographique de collecte d'une centaine de mètres ou plus. Quelques exemples de ce type de campagnes sont la mesure de la qualité de l'air [19], ou la mesure de la qualité des réseaux sans fil comme présentés dans ce chapitre [61].

6.4 CONCLUSION

Nous avons présenté dans ce chapitre l'évaluation du modèle de recrutement collaboratif proposé dans APISENSE®. Ce modèle s'appuie sur la forte concentration des dispositifs en milieu urbain pour coordonner l'exécution des tâches de collecte entre ces dispositifs selon un objectif de couverture géographique.

Pour évaluer ce modèle, nous avons présenté plusieurs expériences basées sur des traces réelles de mobilités provenant de 10000 taxis. Par le biais des expériences menées, nous avons montré qu'en coordonnant l'exécution des tâches, le modèle proposé comporte principalement deux avantages par rapport à un modèle de recrutement où tous les dispositifs les exécutent de manière indépendante. Les expériences ont montré que notre approche permet dans un premier temps de diminuer la quantité de données propagée sur le serveur de plus de 50% lorsque 10000 dispositifs mobiles sont impliqués, tout en gardant une couverture géographique des données collectées similaire. Nous avons également montré que la coordination des tâches de collecte permettait de diminuer la consommation énergétique de ces dispositifs jusqu'à 70%.

Dans le prochain chapitre, nous concluons en faisant un bilan des contributions présentées dans ce manuscrit, et discutons des perspectives de développements et de recherches issues des travaux de cette thèse.

Quatrième partie

Conclusion

7

CONCLUSION

Sommaire

Ce chapitre, qui finalise la présentation faite au travers de ce manuscrit, s'organise de la manière suivante. La section 7.1 rappelle le contexte dans lequel se sont inscrits les travaux réalisés et la problématique qui en est à l'origine. La section 7.2 résume les contributions décrites dans ce manuscrit. Enfin la section 7.3 définit un ensemble de pistes de recherche et de développement en lien avec les travaux de cette thèse.

7.1 RAPPEL DES MOTIVATIONS

Le *Mobile crowdsensing* est une nouvelle alternative exploitant *la foule* de terminaux intelligents déjà déployés à travers le monde pour la collecte massive de données environnementales ou comportementales de la population. Ces dernières années, ce type de collecte de données a suscité l'intérêt d'un grand nombre d'acteurs industriels et académiques dans des domaines tels que l'étude de la mobilité urbaine, la surveillance de l'environnement, la santé ou l'étude des comportements sociaux.

Malgré le potentiel du Mobile crowdsensing, les campagnes de collecte réalisées impliquant un grand nombre d'utilisateurs mobiles sont rares et restent principalement développées par des experts. Ceci est particulièrement dû à l'absence de prise en compte d'aspects non fonctionnels des systèmes proposés dédiés au MCS. En effet, réaliser une campagne de collecte impliquant un grand nombre d'utilisateurs mobiles demande de faire face à de nombreux défis. Ces défis incluent la protection de la vie privée des utilisateurs, les ressources énergétiques limitées des terminaux mobiles, la

mise en place de modèles de récompense et de déploiement adaptés pour recruter les utilisateurs les plus à même de collecter les données désirées, ainsi que de faire face à l'hétérogénéité des plate-formes mobiles disponibles (*par ex.* Android, iOS...). À ce jour, beaucoup d'efforts ont principalement portés sur la réalisation de systèmes monolithiques, difficilement réutilisable dans un contexte non anticipé. Ce manque d'approche réutilisable demande généralement de devoir réinventer la roue pour chaque nouveau système, entravant ainsi l'adoption du MCS pour de nombreux acteurs n'ayant pas les compétences, ni le temps de développer un système capable de répondre à leurs exigences.

Dans cette thèse, nous avons cherché à réétudier les architectures des systèmes dédiés au MCS pour adresser les limitations actuelles liées au développement, au déploiement et à l'exécution d'une campagne de collecte de données. Les différentes contributions proposées sont articulées autour APISENSE®, la plate-forme résultante des travaux de cette thèse. APISENSE® propose un environnement distribué permettant de simplifier le développement et le déploiement de campagnes de collecte de données à travers des utilisateurs mobiles. Nous décrivons dans la section suivante un résumé des contributions décrites dans ce manuscrit.

7.2 RAPPEL DES CONTRIBUTIONS

Dans cette section, nous synthétisons nos contributions présentées dans le chapitre 3 et le chapitre 4 au travers de l'implémentation de la plate-forme APISENSE®.

7.2.1 *Collecte mobile de données*

Dans notre première contribution, nous avons tout d'abord proposé un modèle de programmation pour simplifier le développement de tâches de collecte de données (*cf.* chapitre 3 section 3.3, page 59). Pour concevoir ce modèle, nous avons tenu compte de trois propriétés qui sont la portabilité, la généralité, et l'accessibilité. En ce qui concerne la portabilité, nous avons basé notre modèle sur le langage de script JavaScript, qui est facilement exécutable sur de nombreux systèmes d'exploitation des dispositifs mobiles actuels. Pour l'accessibilité, nous avons proposé une interface de programmation simple et efficace fournissant une abstraction complète des technologies mobiles. Et pour la généralité, l'interface a été conçue pour supporter une grande variété d'activités de

collecte, comprenant la collecte opportuniste et participative (*cf.* chapitre 2 section 2.1.2, page 26). Plus particulièrement, l'interface proposée permet de facilement écouter les changements d'état des capteurs mobiles pour la collecte de données opportuniste, d'interagir avec l'utilisateur mobile pour la collecte de données participative ainsi que définir les stratégies de propagation des données.

Nous avons également présenté dans ce chapitre l'architecture de l'environnement mobile (*cf.* chapitre 3 section 3.4, page 68) assurant l'exécution des tâches de collecte. Cette architecture met principalement l'accent sur la sécurité et le respect de la confidentialité des participants. Pour atteindre cet objectif, tous les scripts sont exécutés dans un bac à sable dédié. Ainsi, cela nous permet de contrôler tous les accès des scripts vers les capteurs des dispositifs mobiles. Par le biais d'une interface graphique, nous laissons la possibilité aux utilisateurs mobiles de définir des contraintes empêchant toute activité de collecte non désirées. Ces contraintes sont classifiées en quatre catégories : temporelles, géographiques, de ressources consommées et de capteurs utilisés.

7.2.2 *Collecte répartie de données*

Dans cette deuxième contribution, le défi adressé concerne la généralité de la plate-forme pour concevoir une grande variété de campagnes de collecte ainsi que son passage à l'échelle.

Pour supporter une grande diversité de campagnes de collecte, nous avons proposé un Modèle de Variabilité (MC) [32] (en anglais FM pour *Feature Model*), capturant les différentes configurations d'une application serveur responsable de la gestion des campagnes de collecte (*cf.* chapitre 4 section 4.4.1, page 101). Cette configuration réside en cinq points : i) configurer l'environnement de développement des tâches de collecte, ii) choisir un modèle de recrutement des participants, iii) choisir une série de traitements sur les données collectées avant leur insertion dans la base de données (*par ex.* anonymisation des données), iv) spécifier l'indexation des données dans la base de données, v) exposer des services pour exploiter les données collectées. Ce modèle est ensuite fourni aux utilisateurs pour leur permettre de définir des exigences selon la spécificité des campagnes qu'ils veulent mener. À partir des ces exigences, une application serveur (appelée nœud de collecte) est générée et configurée fournissant un environnement dédié à la gestion des campagnes de collecte des utilisateurs.

En ce qui concerne le passage à l'échelle de la plate-forme, nous avons proposé une architecture décentralisée, séparant les préoccupations du déploiement ainsi que de la

collecte et l'analyse des données (*cf.* chapitre 4 section 4.2, page 92). L'architecture résultante est composée d'un ensemble de nœuds de collecte et d'un serveur central. Dans cette architecture, le serveur central est responsable du déploiement des tâches de collecte soumises par les nœuds de collecte vers les participants de la plate-forme qui se sont enregistrés au préalable. Les nœuds de collecte, quant à eux, fournissent un environnement dédié permettant aux utilisateurs de développer de nouvelles tâches de collecte, de les déployer auprès du serveur central, et d'exploiter les données qui ont été collectées durant l'exécution des tâches de collecte.

Ce procédé a principalement deux avantages. Le premier est d'assurer une première couche d'anonymat des participants, dans le sens où tous les nœuds de collecte ne peuvent pas déployer les tâches de collecte sans passer par le serveur central. Le deuxième permet aux nouveaux utilisateurs de bénéficier d'un ensemble de participants déjà disponibles pour exécuter des tâches de collecte, leur évitant ainsi une longue période de recrutement.

Finalement, nous avons proposé une extension de APISENSE® dédiée à l'optimisation de l'exécution de campagne de collecte communautaire (*cf.* chapitre 4 section 4.5, page 113). L'optimisation proposée permet de coordonner l'exécution des tâches de collecte entre les dispositifs mobiles en fonction de propriétés de couvertures géographique et temporelle. Principalement, cette optimisation a pour objectif dans un premier temps de diminuer la consommation énergétique globale de tous les dispositifs impliqués dans la campagne, et également de diminuer la quantité de données propagées vers le nœud de collecte.

7.3 PERSPECTIVES

Dans cette section, nous présentons un ensemble de pistes de développement et de recherche en lien avec les travaux présentés dans cette thèse.

Une première contribution au développement d'APISENSE® serait dans un premier temps de porter l'application mobile sur les divers systèmes d'exploitation mobiles disponibles sur le marché (*par ex.* iOS, Windows Mobile). Bien que nous avons effectué une preuve de concept de la portabilité de l'application mobile d'APISENSE® sur iOS, le prototype réalisé n'est pas encore opérationnel pour être déployé à grande échelle.

Concernant la partie serveur de APISENSE®, plusieurs améliorations à court terme peuvent être envisagées. La première concerne la sécurité lors de la propagation des données des dispositifs vers les nœuds de collecte. En effet, actuellement l'emplacement du participant peut être déterminé en identifiant l'adresse IP du point d'accès utilisé pour propager les données, permettant potentiellement d'identifier son domicile même si les données en elles-mêmes ne permettent pas de révéler cette information. Dans ce contexte, une priorité serait d'utiliser des routeurs ou relais spécifiques (*par ex.* routeur oignon [18]) pour cacher l'adresse IP lors de la propagation des données. Une autre amélioration à court terme serait d'intégrer des outils de visualisation génériques permettant de rapidement explorer les données collectées à travers une Interface web. Par exemple, il serait intéressant d'établir des connexions entre les nœuds de collecte et des services tels que CartoDB [1] ou Google Fusion Tables [2] qui proposent ce type de fonctionnalité. Une autre amélioration serait de supporter le déploiement automatique des nœuds de collecte vers les infrastructures des utilisateurs. Cette tâche reste encore manuelle et peut demander une certaine expertise pour administrer le système responsable de leurs exécutions. Certains travaux dans notre équipe ont déjà été initiés dans ce sens [53, 54] et nous prévoyons de rapidement les intégrer dans APISENSE®.

D'autres pistes de développement seraient d'implémenter les différents modèles de protection de la vie privée et de récompense proposés dans la littérature. Dans ce contexte, il serait intéressant de déployer des campagnes utilisant ces différents modèles sur des échantillons de participants, afin d'évaluer leurs impacts sur leurs participations. Nous pensons que APISENSE® peut être une plate-forme idéale pour permettre aux scientifiques travaillant sur ces problématiques pour les aider à mettre au point et valider leurs algorithmes.

Bien que le modèle de programmation proposé dans cette thèse fournisse une abstraction complète des technologies mobiles, une expertise en programmation est quand même nécessaire pour le développement des tâches de collecte. Dans ce contexte, il pourrait être intéressant de proposer un niveau d'abstraction encore plus élevé ne nécessitant aucune expertise. Ce niveau d'abstraction peut s'appuyer par exemple sur certains travaux déjà existant comme App Inventor [3] ou Alice [15] qui proposent

1. http://cartodb.com
2. http://tables.googlelabs.com
3. http://appinventor.mit.edu

des métaphores visuelles pour le développement des applications, ou encore sur le langage naturel. Dans ce contexte, le modèle de programmation que nous avons proposé pourrait être utilisé comme langage pivot avec ce nouveau niveau d'abstraction.

Une autre perspective serait de mettre en place des métriques permettant d'évaluer la qualité des données collectées. Très peu de recherche ont encore été menées dans ce sens, bien que cela représente une problématique importante de ce type d'applications. En effet, de nombreux facteurs peuvent influer sur la qualité des données comme le type de matériel utilisé, la mobilité des utilisateurs, etc. En effet, sans connaître le niveau de fiabilité des données, il peut être très difficile de savoir si les données collectées sont représentatives du monde réel ou du comportement des participants. La mise en place de ces métriques pourrait également permettre de mettre en place de nouveaux modèles de recrutement des participants dans les campagnes de collecte. Dans ce contexte, selon les exigences de la campagne, les participants fournissant les données de plus grande qualité pourraient être privilégiés.

Dans cette thèse, nous avons particulièrement investi sur le développement d'un système de collecte de données avec APISENSE®. Cependant, la collecte en elle même ne représente qu'une partie des futurs systèmes d'informations se basant sur le *pouvoir de la foule* pour atteindre une quantité massive de données. De nombreux défis doivent encore être relevés pour traiter ces données, en extraire des comportements complexes et diffuser des informations en temps réel à partir de leurs analyses [68]. Un cas d'utilisation que nous souhaitons investir est l'utilisation du *Mobile crowdsensing* pour élaborer un système de détection de tremblement de terre. Certaines initiatives ont déjà vu le jour comme le projet Quake-Catcher Network [13], qui permet de recueillir les données issues des capteurs de mouvement présents dans les disques durs des ordinateurs, transformant alors ceux-ci en sismographes. Comme mentionné par Faulkner et al. [23], cette approche peut être complétée en utilisant les accéléromètres intégrés dans les dispositifs mobiles qui peuvent fournir une mesure plus précise et plus complète de la distribution géographique des secousses lors d'un tremblement de terre. Cependant, développer un système fiable, capable d'analyser continuellement les données produites par des centaines de milliers de capteurs, extraire uniquement les informations pertinentes et propager une alerte sismique seulement en quelques secondes restent actuellement un véritable défi.

BIBLIOGRAPHIE

[1] Nadav Aharony, Wei Pan, Cory Ip, Inas Khayal, and Alex Pentland. Social fmri : Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing*, 7(6) :643–659, 2011.

[2] Marc P Armstrong, Gerard Rushton, Dale L Zimmerman, et al. Geographically masking health data to preserve confidentiality. *Statistics in medicine*, 18(5) :497–525, 1999.

[3] Patrick Baier, Frank Durr, and Kurt Rothermel. Psense : Reducing energy consumption in public sensing systems. In *Advanced Information Networking and Applications (AINA),26th International Conference on*, pages 136–143. IEEE, 2012.

[4] Rajesh Krishna Balan, Khoa Xuan Nguyen, and Lingxiao Jiang. Real-time trip information service for a large taxi fleet. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 99–112. ACM, 2011.

[5] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones : a measurement study and implications for network applications. In *Proceedings of the 9th conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[6] Niels Brouwers and Koen Langendoen. Pogo, a middleware for mobile phone sensing. In *Proceedings of the 13th International Middleware Conference*, pages 21–40. Springer-Verlag New York, Inc., 2012.

[7] J Burke, D Estrin, M Hansen, A Parker, N Ramanathan, S Reddy, and MB Srivastava. Participatory sensing. In *In : Workshop on World-Sensor-Web (WSW'06) : Mobile Device Centric Sensor Networks and Applications*, 2006.

[8] Iacopo Carreras, Daniele Miorandi, Andrei Tamilin, Emmanuel R Ssebaggala, and Nicola Conci. Matador : Mobile task detector for context-aware crowd-sensing campaigns. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), International Conference on*, pages 212–217. IEEE, 2013.

[9] Guanling Chen, David Kotz, et al. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[10] NM Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5) :862–876, 2010.

[11] Delphine Christin, Andreas Reinhardt, Salil S Kanhere, and Matthias Hollick. A survey on privacy in mobile participatory sensing applications. *Journal of Systems and Software*, 84(11) :1928–1946, 2011.

[12] Paul Clements and Linda Northrop. Software product lines : practices and patterns. 2002.

[13] Elizabeth S Cochran, Jesse F Lawrence, Carl Christensen, and Ravi S Jakka. The quake-catcher network : Citizen science expanding seismic horizons. *Seismological Research Letters*, 80(1) :26–30, 2009.

[14] Ionut Constandache, Shravan Gaonkar, Matt Sayler, Romit Roy Choudhury, and Landon Cox. Enloc : Energy-efficient localization for mobile phones. In *INFOCOM*, pages 2716–2720. IEEE, 2009.

[15] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. In *ACM SIGCSE Bulletin*, volume 35, pages 191–195. ACM, 2003.

[16] Tathagata Das, Prashanth Mohan, Venkata N Padmanabhan, Ramachandran Ramjee, and Asankhaya Sharma. Prism : platform for remote sensing using smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 63–76. ACM, 2010.

[17] Linda Deng and Landon P Cox. Livecompare : grocery bargain hunting through participatory sensing. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, page 4. ACM, 2009.

[18] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor : The second-generation onion router. Technical report, DTIC Document, 2004.

[19] Prabal Dutta, Paul M Aoki, Neil Kumar, Alan Mainwaring, Chris Myers, Wesley Willett, and Allison Woodruff. Common sense : participatory urban sensing using

a network of handheld air quality monitors. In *Proceedings of the 7th ACM conference on embedded networked sensor systems*, pages 349–350. ACM, 2009.

[20] Shane B Eisenman, Emiliano Miluzzo, Nicholas D Lane, Ronald A Peterson, Gahng-Seop Ahn, and Andrew T Campbell. Bikenet : A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1) :6, 2009.

[21] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2) : 114–131, 2003.

[22] Hossein Falaki, Ratul Mahajan, and Deborah Estrin. Systemsens : a tool for monitoring usage in smartphone research deployments. In *Proceedings of the sixth international workshop on MobiArch*, pages 25–30. ACM, 2011.

[23] Matthew Faulkner, Michael Olson, Rishi Chandy, Jonathan Krause, K Mani Chandy, and Andreas Krause. The next big one : Detecting earthquakes and other rare events from community-based sensors. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 13–24. IEEE, 2011.

[24] Zachary Fitz-Walter and Dian Tjondronegoro. Exploring the opportunities and challenges of using mobile sensing for gamification. In *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp'11) : Workshop on Mobile Sensing*, 2011.

[25] Jon Froehlich, Mike Y Chen, Sunny Consolvo, Beverly Harrison, and James A Landay. Myexperience : a system for in situ tracing and capturing of user feedback on mobile phones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 57–70. ACM, 2007.

[26] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Show me how you move and i will tell you who you are. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, pages 34–41. ACM, 2010.

[27] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing : Current state and future challenges. *Communications Magazine, IEEE*, 49(11) :32–39, 2011.

[28] Chunming Gao, Fanyu Kong, and Jindong Tan. Healthaware : Tackling obesity with health aware smart phone systems. In *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*, pages 1549–1554. IEEE, 2009.

[29] Shravan Gaonkar, Jack Li, Romit Roy Choudhury, Landon Cox, and Al Schmidt. Micro-blog : sharing and querying content through mobile phones and social participation. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 174–186. ACM, 2008.

[30] Kuan Lun Huang, Salil S Kanhere, and Wen Hu. Preserving privacy in participatory sensing systems. *Computer Communications*, 33(11) :1266–1280, 2010.

[31] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[32] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[33] Wazir Zada Khan, Yang Xiang, Mohammed Y Aalsalem, and Quratulain Arshad. Mobile phone sensing systems : a survey. *Communications Surveys & Tutorials, IEEE*, 15(1) :402–427, 2013.

[34] Donnie H Kim, Younghun Kim, Deborah Estrin, and Mani B Srivastava. Sensloc : sensing everyday places and paths using less energy. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 43–56. ACM, 2010.

[35] Stephen F King and Paul Brown. Fix my street or else : using the internet to voice local public service concerns. In *Proceedings of the 1st international conference on Theory and practice of electronic governance*, pages 72–80. ACM, 2007.

[36] John Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6) :391–399, 2009.

[37] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2) :74–82, 2011.

[38] Nicholas D Lane, Shane B Eisenman, Mirco Musolesi, Emiliano Miluzzo, and Andrew T Campbell. Urban sensing systems : opportunistic or participatory ? In *Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 11–16. ACM, 2008.

[39] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9) :140–150, 2010.

[40] Juong-Sik Lee and Baik Hoh. Dynamic pricing incentive for participatory sensing. *Pervasive and Mobile Computing*, 6(6) :693–708, 2010.

[41] Juong-Sik Lee and Baik Hoh. Sell your experiences : a market mechanism based incentive for participatory sensing. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 60–68. IEEE, 2010.

[42] P Lilly. Mobile devices to outnumber global population by 2017. *URL http ://tinyurl. com/pbodtus [Accessedon : 2013-08-06]*.

[43] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 285–298. ACM, 2010.

[44] Pengfei Liu, Yanhua Chen, Wulei Tang, and Qiang Yue. Mobile weka as data mining tool on android. In *Advances in Electrical Engineering and Automation*, pages 75–80. Springer, 2012.

[45] Hong Lu, Nicholas D Lane, Shane B Eisenman, and Andrew T Campbell. Bubble-sensing : Binding sensing tasks to the physical world. *Pervasive and Mobile Computing*, 6(1) :58–71, 2010.

[46] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. Reference model for service oriented architecture 1.0. *OASIS Standard*, 12, 2006.

[47] Nicolas Maisonneuve, Matthias Stevens, Maria E Niessen, and Luc Steels. Noise-tube : Measuring and mapping noise pollution with mobile phones. In *Information Technologies in Environmental Engineering*, pages 215–228. Springer, 2009.

[48] David J Malan and Henry H Leitner. Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 39(1) :223–227, 2007.

[49] Emiliano Miluzzo, Nicholas D Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B Eisenman, Xiao Zheng, and Andrew T Campbell. Sensing meets mobile social networks : the design, implementation and evaluation of the

cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350. ACM, 2008.

[50] Prashanth Mohan, Venkata N Padmanabhan, and Ramachandran Ramjee. Nericell : rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[51] Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, and Péter Boda. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 55–68. ACM, 2009.

[52] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering : foundations, principles, and techniques*. Springer, 2005.

[53] Clément Quinton, Romain Rouvoy, and Laurence Duchien. Leveraging feature models to configure virtual appliances. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, page 2. ACM, 2012.

[54] Clément Quinton, Nicolas Haderer, Romain Rouvoy, and Laurence Duchien. Towards multi-cloud configurations using feature models and ontologies. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 21–26. ACM, 2013.

[55] Moo-Ryong Ra, Bin Liu, Tom F La Porta, and Ramesh Govindan. Medusa : A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 337–350. ACM, 2012.

[56] Kiran K Rachuri, Mirco Musolesi, Cecilia Mascolo, Peter J Rentfrow, Chris Longworth, and Andrius Aucinas. Emotionsense : a mobile phones based adaptive platform for experimental social psychology research. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 281–290. ACM, 2010.

[57] Sasank Reddy, Deborah Estrin, Mark Hansen, and Mani Srivastava. Examining micro-payments for participatory sensing data collections. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 33–36. ACM, 2010.

[58] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch : programming for all. *Communications of the ACM*, 52 (11) :60–67, 2009.

[59] Xiang Sheng, Jian Tang, and Weiyi Zhang. Energy-efficient collaborative sensing with mobile phones. In *INFOCOM, 2012 Proceedings IEEE*, pages 1916–1924. IEEE, 2012.

[60] Xiang Sheng, Xuejie Xiao, Jian Tang, and Guoliang Xue. Sensing as a service : A cloud computing system for mobile phone sensing. In *Sensors, 2012 IEEE*, pages 1–4. IEEE, 2012.

[61] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab : measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3) :15–20, 2011.

[62] Minho Shin, Cory Cornelius, Dan Peebles, Apu Kapadia, David Kotz, and Nikos Triandopoulos. Anonysense : A system for anonymous opportunistic sensing. *Pervasive and Mobile Computing*, 7(1) :16–30, 2011.

[63] Sebastian Sonntag, Lennart Schulte, and Jukka Manner. Mobile network measurements-it's not all about signal strength. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4624–4629. IEEE, 2013.

[64] Latanya Sweeney. k-anonymity : A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05) :557–570, 2002.

[65] Emiliano Trevisani and Andrea Vitaletti. Cell-id location technique, limits and benefits : an experimental study. In *Mobile Computing Systems and Applications (WMCSA). Sixth IEEE Workshop on*, pages 51–60. IEEE, 2004.

[66] Yu Xiao, Pieter Simoens, Padmanabhan Pillai, Kiryong Ha, and Mahadev Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 9. ACM, 2013.

[67] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2011.

[68] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. Sensing as a service and big data. *arXiv preprint arXiv :1301.0159*, 2013.

[69] Gabe Zichermann and Christopher Cunningham. *Gamification by design : Implementing game mechanics in web and mobile apps*. " O'Reilly Media, Inc.", 2011.