A Platform for Building Context-aware Mobile Crowdsensing Applications in Vehicular Social Networks

by

Xiping Hu

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

November 2015

© Xiping Hu, 2015
Abstract

In the past few years, many research works have demonstrated that mobile crowdsensing could be effectively applied in vehicular social networks (VSNs) to serve many purposes and bring huge economic benefits in transportation. In this thesis, we provide a crowdsensing platform which addresses the research challenges in the overall workflow of crowdsensing in VSNs in terms of task allocation and task execution. This platform supports the creation of different context-aware mobile crowdsensing applications and facilitates their real-world deployments in VSNs.

First, because of the inherent nature of crowdsensing, usually a crowdsensing task needs a group of different participants to finish it collaboratively. Thus, for task allocation in crowdsensing, we propose an application-oriented service collaboration model (ASCM). This ASCM automatically allocates multiple participants with multiple crowdsensing tasks across different mobile devices and cloud platform in an efficient and effective manner in VSNs.

Second, for task exaction of mobile crowdsensing applications in VSNs, the dynamic network connectivity of the underlying vehicular ad-hoc networks (VANETs) may cause failures of such applications during their executions. We design S-Aframe, an agent-based multi-layer framework, which provides a programming model to support creation and deployment of robust and reliable crowdsensing applications that self-adapt to the dynamic nature of VANETs. Furthermore, due to the dynamism of VANETs and the opportunism of user connections in VSNs, the changing environments of the users involved in the VSNs may also result in users’ dynamic contexts. We propose a context-aware semantic service (CSS), and incorporate this service with S-Aframe to improve the self-adaptiveness of
mobile crowdsensing applications to users’ dynamic contexts of VSNs.

Finally, we design and develop SAfeDJ, a crowdsensing-based situation-aware music recommendation application for drivers. The development of SAfeDJ has further demonstrated how our platform supports the creation of a context-aware mobile crowdsensing application, and facilitates the realization of such an application in real-world deployment in VSNs.
Preface

The following publications describe the work completed in this thesis. In some cases, the conference papers contain materials overlapping with the journal papers. All the chapters are based on these papers co-authored with Dr. Victor C.M. Leung, who has also supervised this research.

Journal Papers Published:


incorporated in Chapter 3 and Chapter 4, respectively.


**Conference Papers Published:**


Delivery for Drivers”, *ACM MobiCom 2014 Mobile App Competition final*. (acceptance rate = 14.0%)


The explanations about the co-authorship of all the papers are as follows:

- Journal paper 1 was co-authored with Mr. Terry H. S. Chu, Dr. Edith. C.-H. Ngai, Dr. Philippe Kruchten, and Dr. Henry. C.B. Chan. I proposed the topic and initial idea, studied and summarized the overall system architecture of MSNs and VSNs, and outlined the future research directions of MSNs. Mr. Terry H. S. Chu and Dr. Henry C. B. Chan helped to summarize the existing works about data forwarding and dissemination, and resource management and user behavior of MSNs (this part
is not presented in this thesis). Dr. Edith C.-H. Ngai helped to summarize the existing work about mobile internet of things (this part is not presented in this thesis). Dr. Philippe Kruchten provided the initial ideas about how to describe the system architecture of MSNs through three views.

- Journal paper 2 was co-authored with Mr. Junqi Deng, Dr. Jidi Zhao, Ms. Wenyan Hu, Dr. Edith C.-H. Ngai, Mr. Renfei Wang, Mr. Johnny Shen, Mr. Min Liang, and Dr. Xitong Li. I proposed the topic and initial idea of SAfeDJ, designed the overall architecture, all the key components and flows of SAfeDJ, and the experimental settings. Mr. Junqi Deng helped in the design of music feature extraction method, and set up the websites and recruited volunteers to do the surveys about music-mood mapping and driving context with music. Dr. Jidi Zhao helped to implement the ontology-based similarity computing method, and conducted the case study which involved human-being tests of SAfeDJ. Ms. Wenyan Hu and Dr. Edith C.-H. Ngai helped to implement the mood-fatigue detector on Android devices. Mr. Renfei Wang helped to implement the music mapping algorithm on cloud platform and Android devices. Mr. Johnny Shen helped to implement the music satisfaction calculation algorithm on Android devices. Mr. Min Liang helped to maintain the cloud platform and database during the periods of our experiments. Dr. Xitong Li provided suggestions about using the Stata 13 to conduct data statistics about the impact of context to music choice in driving situations.

- Journal paper 3 was co-author with Dr. Jidi Zhao, Dr. Boon-Chong Seet, Mr. Terry H. S. Chu, and Dr. Henry C. B. Chan. I proposed the topic and initial idea, and finished the designs of the S-Aframe framework and the CSS, and conducted the experiments. Dr. Jidi Zhao and her team helped to implement the CSS. Dr. Boon-
Chong Seet provided the initial idea of the ridesharing application example to demonstrate the idea and usefulness of S-Aframe and CSS, and provided some suggestions about the experimental setting. Mr. Terry H. S. Chu and Dr. Henry C. B. Chan helped to develop the prototype application example on Android devices.

- Journal paper 4 was co-authored with Dr. Xitong Li, Dr. Edith. C.-H. Ngai, and Dr. Philippe Kruchten. I proposed the topic and initial ideas about the architecture, finished the overall design, and conducted the experiments. Dr. Xitong Li helped in the design of the CCE model (this part is not presented in this thesis). Dr. Edith C.-H. Ngai helped to summarize multiple sources of sensing data and the classification of them (this part is not presented in this thesis). Dr. Philippe Kruchten provided suggestions about how to describe the overall system architecture in a more rigorous manner, and the usage of UML to describe some processes in the application example.

- Journal paper 5 was co-authored with Mr. Terry H. S. Chu, and Dr. Henry C. B. Chan. I proposed the topic and initial idea of ASCM, finished the design of ASCM, designed and developed the Vita platform and the application example, and conducted most of the experiments. Mr. Terry H. S. Chu and Dr. Henry C. B. Chan helped to implement the ASCM on Android devices, and developed the prototype application.

As the journal papers are extended versions of the conference papers, thus the co-authorship of the conference papers are corresponding to the journal papers.
# Table of Contents

Abstract ........................................................................................................................................... ii

Preface ............................................................................................................................................... iv

Table of Contents .............................................................................................................................. ix

List of Tables ..................................................................................................................................... xiii

List of Figures ................................................................................................................................... xiv

List of Abbreviations ......................................................................................................................... xvii

Acknowledgements .......................................................................................................................... xix

Dedication .......................................................................................................................................... xx

Chapter 1: Introduction ..................................................................................................................... 1

1.1 Overview of Mobile Crowdsensing Applications in VSNs (Trends, Challenges and Needs) .......................................................................................................................... 3

1.1.1 Task allocation of mobile crowdsensing in VSNs ................................................................. 4

1.1.2 Task execution of mobile crowdsensing in VANET-based VSNs ..................................... 6

1.2 Summary of Main Contributions ............................................................................................... 9

1.3 Thesis Organization .................................................................................................................... 13

Chapter 2: Literature Reviews and System Architecture of VSNs ............................................. 15

2.1 System Architecture of MSNs .................................................................................................. 15

2.1.1 Physical architecture view of MSNs .................................................................................. 17

2.1.2 Development view of MSNs ............................................................................................ 19

2.1.3 Logical view of MSNs ...................................................................................................... 24

2.2 Features, System Architecture and Challenges of VSNs ................................................... 26
Chapter 3: Application-oriented Service Collaboration Model for Task Allocation of Crowdsensing

3.1 Introduction ........................................................................................................... 31

3.2 System Design .................................................................................................... 33
  3.2.1 Application-oriented service collaboration model (ASCM) ......................... 34
    3.2.1.1 Modeling structures and relations of contexts ....................................... 35
    3.2.1.2 Calculation of distances between attributes ....................................... 38
    3.2.1.3 Optimizing the forming of crowd group ............................................. 40
  3.2.2 Working theory of ASCM in Vita for task allocation in crowdsensing .......... 43

3.3 Implementation Strategies .................................................................................. 46
  3.3.1 Implementation of the cloud platform of Vita .............................................. 46
  3.3.2 Implementation of the ASCM ...................................................................... 48
  3.3.3 Implementation of the mobile SOA framework .......................................... 49

3.4 Evaluation ............................................................................................................ 51
  3.4.1 Application example .................................................................................... 51
  3.4.2 System and task performance ...................................................................... 55
  3.4.3 Performance of ASCM ............................................................................... 59

3.5 Related Work ...................................................................................................... 63

3.6 Summary ............................................................................................................ 65

Chapter 4: Agent-based Multi-layer Framework with Context-aware Semantic Service for Crowdsensing Applications in VANET-based VSNs

4.1 Introduction ......................................................................................................... 67

4.2 Related Works .................................................................................................... 69

4.3 Model .................................................................................................................. 71
List of Tables

Table 2.1  Taxonomy of the current survey works of MSNs ..................................................... 15
Table 2.2  Data forwarding approaches .................................................................................. 22
Table 2.3  Data dissemination approaches .............................................................................. 22
Table 2.4  Context-awareness ................................................................................................. 23
Table 2.5  Difference between normal MSNs and VSNs ............................................................ 27
Table 3.1  Summary of success criteria of the design of ASCM .............................................. 32
Table 3.2  Overall system performance of Vita ........................................................................ 56
Table 4.1  Summary of comparisons with related works............................................................. 69
Table 4.2  Summary of success criteria of the design of S-Aframe and CSS ......................... 77
Table 4.3  Summary of the test scenarios.................................................................................. 102
Table 4.4  Semantic matching results....................................................................................... 117
Table 5.1  User history table (UHT) in the crowd history dataset ............................................. 125
Table 5.2  Statistics about the impact of context to music choice in driving situations ......... 129
List of Figures

Figure 1.1 Overall positions about the challenges of mobile crowdsensing in VSNs ..................3
Figure 2.1 Relation between the views of MSN architecture .............................................17
Figure 2.2 The physical architecture of future MSNs ........................................................18
Figure 2.3 The development view of MSNs .....................................................................20
Figure 2.4 The physical architecture of VSNs .................................................................28
Figure 2.5 The development view of VSNs .....................................................................29
Figure 3.1 System architecture of Vita .............................................................................33
Figure 3.2 System model of ASCM ..................................................................................34
Figure 3.3 Social Vectors for task $T_A$ ..........................................................................38
Figure 3.4 Flow of ontology-based calculation of semantic distance ..............................39
Figure 3.5 GA-based clustering ........................................................................................41
Figure 3.6 K-means based clustering ...............................................................................42
Figure 3.7 Process flow of requestor of mobile crowdsensing ......................................44
Figure 3.8 Process flow of participants provide services in mobile crowdsensing ..........45
Figure 3.9 Implementation of the management interface ...............................................47
Figure 3.10 Implementation of the process runtime environment ..................................48
Figure 3.11 Overall implementation architecture of mobile SOA framework .............50
Figure 3.12 The process flow of the mobile SOA framework ..........................................51
Figure 3.13 Smart City - context-aware mobile crowdsensing application ..................52
Figure 3.14 Execution time and battery consumption of the N-queens puzzle (E=1) .......... 58
Figure 3.15 Execution time and battery consumption of the N-queens puzzle (E=3) .......... 58
Figure 3.16 Evaluation results of GA, K-means clustering and random forming of crowd groups (different no. of tasks) ................................................................. 61
Figure 3.17 Evaluation results of GA, K-means clustering and random forming of crowd groups (different no. of attributes) ................................................................. 62
Figure 4.1 Programming model of S-Aframe ................................................................. 72
Figure 4.2 Architecture of S-Aframe ............................................................................. 78
Figure 4.3 Implementation flow of S-Aframe ................................................................. 81
Figure 4.4 Semantic-based model of application specific service of S-Aframe .............. 87
Figure 4.5 Semantic-based model of context information ............................................. 88
Figure 4.6 A rule example ............................................................................................. 89
Figure 4.7 An example of ontology taxonomy .............................................................. 91
Figure 4.8 CSS mechanism ......................................................................................... 93
Figure 4.9 Flow chart of the CSS ............................................................................... 96
Figure 4.10 The map area of the experiment ............................................................... 98
Figure 4.11 Tasks execution success rate-first case ...................................................... 100
Figure 4.12 Tasks execution success rate-second case ............................................... 101
Figure 4.13 Status monitoring of mobile agent ............................................................. 103
Figure 4.14 Time efficiency of one owner application with one or multiple mobile agents. 104
Figure 4.15 Time efficiency (same applications) of multiple owner application ............ 105
Figure 4.16 Time efficiency (different applications) of multiple owner applications ....... 105
Figure 4.17 Time efficiency of multiple owners with multiple mobile agents ............... 105
Figure 4.18 Communication overhead of S-Aframe ..................................................... 107
Figure 4.19 Effectiveness of the semantic-based matching .................................................. 110
Figure 4.20 Screen-shots of the smart ride user interface ............................................... 113
Figure 4.21 Domain ontology of service function descriptions for rideshare ..................... 115
Figure 4.22 Fragment of the road information ontology .................................................. 115
Figure 4.23 Fragment of the rideshare application service ............................................. 116
Figure 4.24 Fragment of the service request .................................................................. 116
Figure 4.25 Fragment of the service from Victor ......................................................... 116
Figure 5.1 System architecture of SAfeDJ .................................................................. 122
Figure 5.2 Overall work flow of SAfeDJ .................................................................... 123
Figure 5.3 Structure of the data used in the driver’s mood-fatigue detector .................. 124
Figure 5.4 Example of similarity computing in CMM ................................................. 126
Figure 5.5 Overhead of SAfeDJ in smartphones ....................................................... 134
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>AS</td>
<td>Application server</td>
</tr>
<tr>
<td>ASCM</td>
<td>Application-oriented service collaboration model</td>
</tr>
<tr>
<td>CF</td>
<td>Collaborative filtering</td>
</tr>
<tr>
<td>CMM</td>
<td>Context-music mood mapping</td>
</tr>
<tr>
<td>CSS</td>
<td>Context-aware semantic service</td>
</tr>
<tr>
<td>DSRC</td>
<td>Dedicated Short Range Communications</td>
</tr>
<tr>
<td>EI</td>
<td>Eye-lid</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td>GPS</td>
<td>Global positioning system</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent transport system</td>
</tr>
<tr>
<td>HMC</td>
<td>Human mood-fatigue context</td>
</tr>
<tr>
<td>Hr</td>
<td>Heart rate</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>MM</td>
<td>Music-mood mapping</td>
</tr>
<tr>
<td>MMD</td>
<td>Music-mood dictionary</td>
</tr>
<tr>
<td>MSN</td>
<td>Mobile social network</td>
</tr>
<tr>
<td>OBD</td>
<td>On-board diagnostic</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>PKI</td>
<td>Public key infrastructure</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>POI</td>
<td>Points of interest</td>
</tr>
<tr>
<td>SC</td>
<td>Social context</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SCDSCF</td>
<td>Semantic-based context-aware dynamic service composition framework</td>
</tr>
<tr>
<td>TC</td>
<td>Traffic context</td>
</tr>
<tr>
<td>Tht</td>
<td>Throttle input</td>
</tr>
<tr>
<td>UHT</td>
<td>User history table</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>V2P</td>
<td>Vehicle-to-pedestrian</td>
</tr>
<tr>
<td>V2R</td>
<td>Vehicle-to-roadside</td>
</tr>
<tr>
<td>V2V</td>
<td>Vehicle-to-vehicle</td>
</tr>
<tr>
<td>VANET</td>
<td>Vehicular ad-hoc network</td>
</tr>
<tr>
<td>VSN</td>
<td>Vehicular social network</td>
</tr>
<tr>
<td>WHO</td>
<td>World Health Organization</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to express my great gratitude to my supervisor Dr. Victor C.M. Leung for his insightful guidance, invaluable support and constant encouragement from the early stages of my PhD research until the very end. Without his wisdom and constructive suggestions, this thesis would not be possible.

I would like to thank my thesis committee members, Dr. Philippe Kruchten, for his comments on defining the research problems in a more rigorous manner, and valuable suggestions on the description of MSN architectures. I would like to thank Dr. Matei Ripeanu, for his advice on the definition of mobile crowdsensing and experimental results in a more precise manner. I would like to thank Dr. Sathish Gopalakrishnan, for his suggestions on the literature review. Also, thanks to Dr. Alexandra Fedorova, Dr. Alan Wagner, and Dr. Chung-Horng Lung, for reading my thesis and providing positive assessments.

I am also thankful to my friends and international partners who helped me along the way with their constructive suggestions and continuous support, especially Dr. Xitong Li from MIT in USA and HEC Paris in France, Dr. Edith C.-H. Ngai from Uppsala University in Sweden, Mr. Terry H. S. Chu from Bravolol in Hong Kong, Dr. Jidi Zhao from East China Normal University and Shanghai Jiaotong University in China, Dr. Weichang Du from University of New Brunswick in Canada, Mr. Min Liang from IBM China, Dr. Boon-Chong Seet from Auckland University of Technology in New Zealand, Dr. Jon Crowcroft from University of Cambridge in UK, and Dr. Lei Wang from University of Twente in Netherlands.

Lastly, I would like to thank my family for their support during these years.

This study was supported in part by the Canadian Natural Sciences and Engineering Research Council (NSERC) and through NSERC-DIVA Strategic Research Network, TELUS and other industrial partners, and the University of British Columbia Tuition Scholarship.
Dedication

With love, to my family.
Chapter 1: Introduction

According to an investigation by the U.S. Department of Transportation, personal vehicles account for 21% of the global energy consumption, while the average occupancy rate of personal vehicle trips is 1.6 persons per vehicle-mile [1]. Given that each personal vehicle can commonly carry up to 4 persons, it means 60% of the carriage capacity is wasted during personal trips. Also, the statistics published by the World Health Organization (WHO) [2] shown that there are 1.2 million people die and 50 million people are injured or disabled on roads every year. A number of research works have demonstrated that applying mobile devices on-board vehicles to intelligent transportation system (ITS) could potentially help to improve transportation efficiency and prevent traffic accidents, and promote human society significantly [3, 4, 5].

The wide-spread deployments of sensors in transportations and the popularity of mobile devices, such as smartphones and tablets, have provided us with numerous sources of sensing data. Contemporary mobile devices are commonly equipped with sensors such as cameras, Global Positioning System (GPS) receivers and accelerometers. The sensing data from wireless sensors like On-Board Diagnostic (OBD) and mobile devices on board vehicles enable the development of sophisticated context-aware mobile applications and services in vehicular networks, which has motivated a wide range of innovative research [6].

Mobile crowdsensing refers to a broad range of community sensing paradigms [7]. As a special form of crowdsourcing, mobile crowdsensing aims to provide a mechanism to involve participants from the general public to efficiently and effectively contribute and utilize context-related sensing data from their mobile devices in solving specific problems in collaborations. Also, different from conventional sensing solutions using specialized networks of sensors, mobile crowdsensing leverages human intelligence to collect, process,
and aggregate sensing data using individuals’ mobile devices (e.g., using a smartphone to share a specific tour information), so as to realize a more effective and efficient sensing solution. Thus, similar to crowdsourcing, a crowdsensing task could be divided to multiple smaller tasks, and the smaller tasks are then distributed to multiple participants’ mobile devices to finish them collaboratively. Furthermore, different from common collaborative applications, the participants of mobile crowdsensing are from undefined public in a mobile environment, rather than being commissioned from a specific/named group. In addition, mobile crowdsensing involves participatory sensing or opportunistic sensing at the two ends [7]: participatory sensing requires the active involvement of individuals to contribute sensing data (i.e., reporting a road closure) related to some large-scale phenomenon, and opportunistic sensing is more autonomous and involves minimal user involvement (e.g., continuous location sampling without the explicit action of the user).

Also, a remarkable trend in mobile computing is the increasing use of mobile devices to access social networking services. The wide availability of sensing modules in mobile devices enables social networking services to be extended to incorporate location based services, etc. Therefore, there is growing interest in fusing social networking services with real-world sensing, such as crowdsensing [8]. Mobile social networks (MSNs) [9] not only can provide an ideal and ubiquitous platform to enable mobile users to participate in crowdsensing, but can also help to improve the context-awareness of mobile applications and better assist users in mobile crowdsensing by analyzing and utilizing their social contexts. Vehicular social networks (VSNs) as an emerging kind of MSN for transportation applications, and a number of research works have identified that crowdsensing in VSN can be effectively used for many purposes and bring huge economic benefits, e.g., traffic management and safety improvement. However, there are still several key research
challenges that currently exist, which may significantly limit the applications of mobile crowdsensing in VSNs. In this thesis, we provide a crowdsensing platform which addresses the challenges in two aspects in the overall workflow of crowdsensing in VSNs: task allocation and task execution, so as to support the creation of different context-aware mobile crowdsensing applications and facilitate the real-world deployment of them in VSNs.

The rest of this chapter is organized as follows. In Section 1.1, we mainly focus on the work flow of mobile crowdsensing, and identify the specific research challenges of crowdsensing in VSNs that are not yet to be addressed by existing work. Also, we explain how solving these problems contributes to the overall goal of our thesis. Section 1.2 provides a summary of the contributions that we provide in our thesis and discusses the significance and novelty of our solutions compared with the existing solutions. Finally, the organization of this thesis is described in Section 1.3.

1.1 Overview of Mobile Crowdsensing Applications in VSNs (Trends, Challenges and Needs)

![Diagram of overall work flow of mobile crowdsensing](image)

Figure 1.1: Overall positions about the challenges of mobile crowdsensing in VSNs

As shown in Figure 1.1, the overall workflow of a typical mobile crowdsensing application that is used in transportation mainly consists of three phases: task allocation, task
execution and utilization of crowd data. For example, in a crowdsensing based music recommendation application for drivers, this application consists of these three phases in a process of music recommendation. First, this application needs to find out the most suitable participants who may contribute valuable data for music recommendation in task allocation. After that, this application needs to make sure the contextual data contributed by the participants can match the requester (i.e., the driver of this application wants to enjoy preferable music while driving) in task execution time. Finally, this application needs to utilize the crowd data from different participants to infer a preferable song which could fit for the driver’s current driving situation and his listening behaviors.

In this thesis, we focus on three specific research challenges in the overall workflow of crowdsensing. In Subsection 1.1.1, we discuss Challenge 1 about task allocation of mobile crowdsensing in VSNs. In Subsection 1.1.2, we analyze Challenge 2 and Challenge 3 about task execution of mobile crowdsensing applications in VSNs, which exist in vehicular ad-hoc networks (VANETs) and varying virtual communities upon VANETs. All of the three research challenges are of particular concern for requesters and participants of mobile crowdsensing in VSNs, and developers of mobile crowdsensing applications for VSNs.

1.1.1 Task allocation of mobile crowdsensing in VSNs

As discussed in Section 1.1, similar to crowdsourcing, a crowdsensing task could be divided to multiple smaller tasks, and the smaller tasks are then distributed to multiple participants’ mobile devices to finish them collaboratively. Obviously, a good crowd group for a crowdsensing task should be formed by participants that have different experiences or strengths. For example, in a mobile crowdsensing task for collaborative route planning to points of interests (POIs), we can divide this task to two smaller tasks, e.g., one is to find out
the tour information to preferable fashion stores in a specific region, and another one is to locate the suitable restaurants in the same region. Also, if we assume there are M participants to join these two crowdsensing tasks, the easiest way to construct two crowd groups is to assign these M participants to the two tasks randomly. However, some participants may have experiences in a particular field such as familiar with the locations and comments of POIs in this region, while some others familiar with the traffic conditions and routes to these POIs. If the participants who have similar experiences are assigned to a same task rather than distributed to two tasks, both of these two tasks may not be finished in an effective manner. Thus, each task should have participants who have different experiences or strengths. Therefore, the allocation between multiple mobile crowdsensing tasks and multiple crowdsensing participants is important.

However, based on our best understanding, there is no research about how to allocate different participants to form crowd groups for finishing crowdsensing tasks yet. Also, a number of researchers have identified that the current crowdsensing approaches are missing automatic methods for the allocation of multiple mobile crowdsensing tasks efficiently and effectively [10, 11, 12]. Furthermore, such approaches are fundamental to enable broad-based application support for crowdsensing in VSNs [12, 13, 14], e.g., collaborative route planning to POIs as mentioned above.

Currently, various community creation approaches have been proposed, such as dynamic social group formation [15]. However, from the aspects of participatory sensing, unlike the social communities which are usually formed by similar social entities, the crowdsensing teams should be formed by dissimilar social entities (i.e., participants with different skills/experiences/positions in VSNs [16]) as discussed above. Thus, the existing
social grouping approaches in mobile networks may not fit the task allocation for multiple participants of crowdsensing in VSNs. Also, opportunistic sensing based approaches mostly focus on the allocation of computing tasks (e.g., avoiding overprovisioning of the servers to mobile devices) [17] and lack effective solutions to quantify the relationship among participants in crowdsensing. Thus, it could not effectively allocate the computing tasks and human based tasks of crowdsensing among individuals in VSNs simultaneously. Therefore, the Challenge 1 about task allocation of crowdsensing in VSNs is:

*Challenge 1: The current crowdsensing approaches are missing automatic methods to allocate multiple participants to form appropriate crowd groups for finishing crowdsensing tasks. An approach is needed to automatically match multiple participants with multiple mobile crowdsensing tasks in VSNs.*

1.1.2 Task execution of mobile crowdsensing in VANET-based VSNs

VSNs are built on top of vehicular networks that provide connectivity between users and devices participating in the VSN as well as the Internet at-large. While cellular networks can provide such connectivity, the cost may be too high and the latency too large. Instead, in-vehicle and road-side users who are close-by (e.g., traveling on the same street or stretch of highway) can use their mobile devices to inexpensively establish a VANET without the need for accessing the cellular infrastructure. Nevertheless, in VSNs, the network connectivity of the underlying VANETs is dynamic and frequently changing as mobile nodes like vehicles can move at high speeds; consequently the wireless links may be unreliable and have short lifetimes. The dynamic network connectivity may cause failures of mobile crowdsensing applications during their executions in VSNs. To enable the adaptation of mobile crowdsensing applications to such dynamic network conditions,
beyond many challenges already addressed in network protocol designs in all layers of the protocol stack [18, 19], in the application layer, its major challenge is that the applications need to handle possible disconnections and reconnections [18, 20, 21].

However, in the application layer, existing solutions on self-adaptive mobile applications in dynamic networks still have several constrains and could not address this challenge effectively over VANETs. First, traditional approaches like JADE [22] and SPRINGS [23] adopt a centralized approach to track and update the network status for mobile applications. These approaches may not be suitable for VANETs, where centralized nodes may not exist and direct communications are not always possible [21]. Second, several solutions such as [24] strictly constrain the behaviors of mobile applications in finishing some simple tasks and is not scalable for extension to finish more complicated tasks. In addition, some solutions depend on specific hardwares and have only been designed for specific applications [25, 26]. Thus, they do not give a paradigm that can be used for different mobile crowdsensing applications that self-adapt to dynamic network connectivity status in VSNs, and a new programming framework with features of automatic and decentralized network management, scalability, and universality (i.e., platform/hardware independence) still needs be investigated. Consequently, the Challenge 2 about task execution of mobile crowdsensing applications in VANET-based VSNs is:

**Challenge 2:** In the application layer, a framework for the creation and deployment of different robust and reliable mobile crowdsensing applications that self-adapt to the dynamic nature of VANETs is needed but not yet available.

Furthermore, in VSNs, because of the dynamism of VANETs discussed above and the opportunism of user connections in VSNs, the changing contexts of the users involved in the
VSNs may also result in users’ dynamic contexts (which may vary with changing VSN context, such as user location, time periods, identities of neighbors and objects, etc.). Also, as discussed in Section 1.1, different from common collaborative applications, the participants in mobile crowdsensing are from undefined public in the mobile environment, rather than being commissioned from a specific/named group. Thus, in users’ dynamic contexts of VSNs, beyond adaptation to dynamic network connectivity status, the mobile crowdsensing applications also need the capabilities to autonomously adapt their behaviors to match appropriate service and information with different users in crowdsensing during run-time [14]. This is important, as such capabilities could maximize the relevance of data for the requesters of crowdsensing by delivering them with appropriate services relevant to their application context. Also, for the participants in crowdsensing, such capacities enable them to contribute sensing data that are relevant and valuable to their activities, and interests in VSNs.

However, in the traditional approaches in mobile computing, the descriptive information of the service requester (i.e., crowdsensing requester) is compared to that of the service provider (i.e., crowdsensing participant), and their similarity is measured using traditional service matching by simple string or key-word matching, e.g., location based [24], identities based [27] methods. Such approaches cannot work well as it is not realistic to require two entities to use exactly the same contexts (i.e., words about the vehicle users’ destinations/preferences) in open and dynamically changing environments of VSNs. Also, most of the existing high level service matching approaches like [28] are based on complex system, and may cause much redundancy in information representations that can hardly be processed in mobile devices, and thus are not necessarily suitable in the context of VSNs.
Therefore, the Challenge 3 about task execution of mobile crowdsensing applications in VSNs is:

**Challenge 3:** In users’ dynamic contexts of VSNs, it is challenging for mobile crowdsensing applications to autonomously match appropriate service and information with different users (requesters and participants) in crowdsensing, while a systematic approach is lacking.

1.2 **Summary of Main Contributions**

In this thesis, we first conduct a comprehensive survey to study the system architecture of MSNs and VSNs [139], which is presented in Chapter 2. After that, we propose a crowdsensing platform which carefully considers the challenges, and requirements of various mobile crowdsensing applications in VSNs, particularly on task allocation and task execution of crowdsensing as discussed in Section 1.1. Our proposed platform to approach the context-awareness of ubiquitous mobile crowdsensing applications in VSNs are presented in three chapters as follows:

- As identified in Subsection 1.1.1, the existing approaches still could not support automatically allocating multiple crowdsensing participants with multiple crowdsensing tasks in an efficient and effective manner in VSNs. In Chapter 3, we design a RESTful web service-based mobile cloud platform called Vita, which consists of a novel application-oriented service collaboration model (ASCM) to achieve the automatic allocation of multiple mobile crowdsensing tasks with multiple users. ASCM models the structures of user context (i.e., number of similar tasks the users executed, computation and communication capacities of the mobile devices they carried), crowd group context and crowdsensing task context, and the relations between them. Then, ASCM calculates the distance between each entities of tasks and users,
such as it calculates the semantic distance between a crowdsensing participant’s social attribute and a specific requirement of a crowdsensing task. This can retrieve the implicit context and make the matching between the participants and tasks in a more precise manner. Based on these, ASCM integrates intelligent algorithms to optimize the forming of crowd groups when allocating multiple participants to multiple crowdsensing tasks in mobile devices’ run-time. Finally, ASCM adopts service composition approach to integrate the contextual data from different participants. This enables the data transmission between mobile devices and cloud platform be more application-related, and makes the instance object fields a better match with the corresponding method and avoid unnecessary network overhead. To the best of our knowledge, ASCM is the first work that investigates how to quantify the many to many relationships between each context in crowdsensing, and supports automatic allocations of multiple crowdsensing tasks with multiple crowdsensing participants across mobile devices and cloud platform in an efficient and effective manner. Also, we implemented the Vita platform, to verify the feasibility and investigate how to facilitate the realization of our ASCM approach in the real-world deployment. In addition, the Vita platform provides a generic model to deploy, examine and evaluate different context-aware solutions for mobile crowdsensing applications. To demonstrate the effectiveness of ASCM, we develop a prototype crowdsensing application based on the Vita platform to demonstrate the expressivity of ASCM. Also, the experimental results show that ASCM performs better in the allocation of multiple crowdsensing tasks with multiple participants, compared to the existing solutions for mobile crowdsensing. Parts of this chapter have been included in two published journal articles [140, 141], and two published conference papers [142, 143].
As discussed in Subsection 1.1.2, two main challenges should be dealt with in the task execution of mobile crowdsensing applications in VSNs. In Chapter 4, we propose S-Aframe, which is an agent based multi-layer framework with context-aware semantic service (CSS) to enable the self-adaptivity of different mobile crowdsensing applications in high dynamic environment of VANET-based VSNs. Compared to alternative agent-based solutions, S-Aframe has the following advantages. First, unlike the existing works [22, 23] that adopt centralized nodes to support the adaptation of mobile applications, the applications developed on S-Aframe do not need to be concerned with the failures of any specific nodes involved in the VANETs, as mobile agents can collaborate with resident agents in every node to self-update the service (e.g., network status) during their migrations. Second, compared to the former agent-based solutions in mobile computing [13, 24, 29], which provide services to mobile agents through hosting middleware/components, we are the first to adopt resident agents to provide a reusable set of services to mobile agents. This approach can: (i) facilitate resource reuse for multiple crowdsensing applications on each mobile device during run-time; (ii) enable more flexible operations of the mobile agents (through agent communications instead of specific communications with hosting component like tuple space [11]); and (iii) provide scalable options to different mobile devices as the hardware capacities of them are heterogeneous, e.g., on-demand deployment of resident agents and application services. Furthermore, since the programming model of S-Aframe is independent of specific hardware and platform, hence S-Aframe provides a modeling paradigm which could be used for different mobile crowdsensing applications self-adapting to dynamic network connectivity status in VANETs. Also,
compared to other alternative service matching approaches over dynamic networks [23, 28, 29], our proposed CSS has the following advantages. First, in terms of the range of context information, CSS proposes a comprehensive context model covering the major parties in a VSN. Second, compared to the semantic representation model in [28], CSS adopts a much more straight-forward formula, labeled links, to establish the relationships between services and their semantics, as well as between a context and its semantics, which is more lightweight and scalable for resource-constrained mobile devices. Furthermore, to improve interoperability and self-adaptiveness, CSS models not only the semantics of application specific services and context information with labeled links and ontologies, but also the semantics of application domain logics and user-specified preferences information with logic rules. Compared to the ontology based framework proposed in [30], CSS also allows application developers and end users to further specify rules on how to interpret context information and model application domain logics. Finally, CSS utilizes a semantic matching mechanism on top of ontology reasoning and rule inference to match user requests and services with respect to specific context information so as to improve its adaptability over VSNs. Thus, CSS enables mobile crowdsensing applications built on S-Aframe to intelligently determine what and how services and information should be delivered to users in crowdsensing, and autonomously adapt the behaviors of these services to users’ dynamic contexts of VSNs during run-time. A practical implementation and experimental evaluations of S-Aframe are presented to demonstrate its reliability and efficiency in terms of computation and communication performance on popular mobile devices. In addition, a VSN based smart ride application is developed to demonstrate the functionality and practical usefulness of S-Aframe and CSS. Parts of this chapter
have been included in two published journal articles [141, 144], and one published conference papers [145].

- In Chapter 5, we design and develop SAfeDJ, a crowdsensing-based situation-aware music recommendation application for drivers. The development of SAfeDJ further demonstrates how the solutions designed in our platform support the creation of context-aware mobile crowdsensing applications, and facilitate the realization of such applications in real-world deployment in VSNs. Based on ASCM and CSS designed in our platform, SAfeDJ enables different smartphones to collaboratively recommend preferable music to drivers according to each driver’s specific situations via crowdsensing, hence diminishing fatigue and negative emotion of drivers. Also, our practical experiments have demonstrated the feasibility of the SAfeDJ application built on our crowdsensing platform for real-world deployment in VSNs. Parts of this chapter have been included in an published journal article [146], and two published conference papers [147, 148].

1.3 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we study the system architecture of MSNs in terms of physical architecture view, development view and logical view, and discuss the unique features and system architecture of VSNs compared with MSNs. Our application-oriented service collaboration model (ASCM) for the automatic allocation of multiple mobile crowdsensing task with multiple users in an efficient and effective manner, is presented in Chapter 3. We propose S-Aframe in Chapter 4, an agent-based multi-layer framework with context-aware semantic service (CSS) for the execution of
crowdsensing applications, which provides a systematic approach that enables the self-adaptivity of different crowdsensing applications in high dynamic environment of VSNs. Based on the work introduced in Chapter 3 and Chapter 4, we further design and implement a novel situation-aware music recommendation application called SAfeDJ for drivers, which is described in Chapter 5. Finally, Chapter 6 summarizes the main results, conclusions and possible directions for future research. Each of the main chapters in the thesis is self-contained and included in separate journal articles and conference papers. For any of the research problems studied in each chapter, a corresponding study of the related work for the particular problem is included in the chapter accordingly.
Chapter 2: Literature Reviews and System Architecture of VSNs

As outlined in Chapter 1, our work aims to provide systematic solutions to facilitate the context-awareness of ubiquitous mobile crowdsensing applications in the context of VSNs, and VSNs are an emerging kind of MSN for transportation applications. Thus, to form a good foundation for our research targets, we first do a comprehensive survey on the system architecture of MSNs. After that, we review and present the unique features and system architectures of VSNs compared with MSNs, and discuss the specific challenges of mobile applications in VSNs.

2.1 System Architecture of MSNs

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Survey works of MSNs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[9] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40]</td>
</tr>
<tr>
<td>MSN software system layer</td>
<td></td>
</tr>
<tr>
<td>Applications and services</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Web-based services</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Platforms</td>
<td></td>
</tr>
<tr>
<td>Mobile software platforms - commercial and based on Internet</td>
<td>✓</td>
</tr>
<tr>
<td>Mobile software platforms - based on hybrid network for distributed computing/application development</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>Mobile operating systems</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>MSN network layer</td>
<td></td>
</tr>
<tr>
<td>MSN network architectures</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Middleware designs for network services</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Protocol designs</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Wireless access infrastructures</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

Currently, several surveys on MSNs can be found in the literature. As shown in Table 2.1, corresponding to the media layers and host layers, we classify the survey works into two subjects - MSN network layer and MSN software system layer. The applications and network architectures of MSNs are summarized in [9], primarily from the perspective of
communications and focusing on issues and approaches related to protocol designs in MSNs. The network architectures and social properties of MSNs are studied in [31]. Design mythologies of software services and software platform for MSN applications are summed up in [32] and [33], respectively. The features, commercial solutions and related architectures of MSNs are reviewed in [34]. A survey about the current dominant mobile operating systems (OSs) on which the MSN applications are running is presented in [35]. Also, the middleware techniques that could be used to improve the network performances of MSNs are reviewed and compared in [36] and [37]. Similarly, [38] and [39] focus on the design of network services of MSNs and outline the corresponding future research directions. In addition, [40] presents a unified architectural model with a new taxonomy for context data distribution across different layers of mobile ubiquitous systems. However, from Table 2.1, we can observe that a comprehensive survey on the overall design of system architectures of MSNs in mobile networks, from the perspective of mobile software system and mobile computing, is still lacking. Such a survey not only can outline the positions of different software and mobile computing techniques in MSNs, but also can facilitate effective organizations and constructions of MSN software systems, and hence would be of interest to researchers, service providers, mobile application developers, and users of MSNs.

Thus, in this part, we study the system architecture of MSNs through three architectural views from different stakeholders’ perspectives [41, 42]: physical view from the aspects of system engineers of MSNs, development view from the aspects of MSN application developers, and logical view from the aspects of end users of MSNs. The three views and their relations are as illustrated in Figure 2.1.
2.1.1 Physical architecture view of MSNs

Recent technological advances have given mobile devices such as smartphones the capability to generate, store, and share contents directly without central servers. Spatial sensing data (e.g., on location, temperature, movement speed) are also available by multiple sensing models and contextual computing capabilities [43] embedded in a mobile device. Such contents may be of interest to specific groups of people in a specific time or geographic area. Many opportunistic contacts may also be available to form opportunistic networks [44] for sharing or collecting data between mobile devices when people move around in their daily lives. Apart from using smartphones to get social contents and services from social networks through the traditional communication network (i.e., the Internet), opportunistic networks formed with smartphones allow social messages to be forwarded in an ad-hoc manner using opportunistic channels between people. Opportunistic network is especially suitable for data sharing between people in the same geographical vicinity. Mobile users can exchange data through short-range communications by WiFi Direct using their smartphones. Since opportunistic data sharing does not rely on any network infrastructure, it reduces data traffic through the Internet.
As shown in Figure 2.2, the MSNs have a hybrid architecture (client-server and peer-to-peer), which is an integration of the traditional Internet and opportunistic networks. Mobile devices can communicate with each other with or without infrastructure. In addition to data such as news, weather forecasts, traffic alerts, and social media that can be retrieved from the Internet, user-generated data such as messages, photos, and micro-blogs can also be collected and shared through opportunistic contacts between mobile devices. Thus, comparing to the conventional client-server architecture of online social networks, the future MSNs have three additional capabilities: (i) opportunistic data exchange, (ii) multi-hop communications, and (iii) mobile opportunistic computing.

The traditional Internet architecture assumes a connected path from source to destination with a low propagation delay and packet loss rate [45], but it cannot take advantage of the benefits arising from opportunistic contacts. In contrast, opportunistic networks do not assume that a connected path exists over the Internet. Instead, opportunistic interactions between mobile devices take place when data is sent from a source node to one or more destination nodes via either direct communications between devices using Wi-Fi.
Direct or Bluetooth, or other mobile devices as relay nodes using a store, carry and forward approach. Opportunistic data exchange is particularly suitable for people located physically close to each other. This mechanism can provide alternate sources of data and significantly reduce the traffic load of the cellular networks, e.g., the Internet based vehicular networks.

Apart from data sharing between direct neighbors, multi-hop communications have also been explored in opportunistic networks. Due to the mobility of users, it is challenging to provide end-to-end communications between two specific users who are not direct neighbors. This is usually done via multiple intermediate users to relay the messages. Since the exchange of data between nodes consumes resources such as energy and storage, only encountered nodes that have a higher probability of delivering the data to the destination node(s) should be selected as relay nodes. The delivery probability is estimated based on contact history, mobility pattern, social relationships, or common interests between users.

In addition, from computing aspects, mobile opportunistic computing can be considered to be an evolution of mobile computing, in which multiple autonomous mobile computing devices interact with each other to cooperate and achieve a common goal in an ad-hoc and opportunistic manner, such as solving a big computation problem through distributed task executions [46]. Similarly, in opportunistic networks, many autonomous mobile devices communicate with each other in order to execute a task, such as mobile crowdsensing. However, applications or systems in opportunistic networks may need to contend with intermittent connectivity [47].

2.1.2 Development view of MSNs

As shown in Figure 2.3, different from the conventional client-server architecture of online social networks, data exchange between users over opportunistic network will be used
for MSNs. Mobile users who are in the same vicinity can exchange data directly through short-range communication capabilities equipped on their mobile devices. Short-range communications can significantly reduce power consumption and avoid unnecessary communications over the wide-area network infrastructures. The developers of future MSNs need to focus on three layers from the bottom to the top: opportunistic communications, MSN applications on top of the mobile OSs, and MSN server side. In this part, we first review and analyze the two key technologies: ad-hoc connectivity and opportunistic data routing, which could support the development of opportunistic communications; and then summarize the technology about context-awareness, which may play an important role to facilitate collaborations across the upper MSN application layers and opportunistic communications over future MSNs.

![Figure 2.3: The development view of MSNs](image-url)
Ad-hoc connectivity: In addition to wide-area wireless networking interfaces such as 2G, 3G, and 4G, recent mobile devices are also equipped with short-range radio interfaces such as WiFi Direct and Bluetooth, which enable local peer-to-peer communications at a low-cost. Such technologies enable future MSNs to operate even without infrastructure. Currently, Wi-Fi Direct has peer-to-peer transfer speeds of up to 250Mbps with a maximum distance of 656 feet, while Bluetooth 4.0 has lower power consumption and operates with speeds up to 25Mbps over a distance of at least 200 feet.

Opportunistic data routing - forwarding and dissemination: Due to their dynamic and volatile nature, opportunistic networks operate under a completely new networking paradigm such that traditional routing protocols cannot be applied to them. In fact, they introduce new technical challenges and problems. There has been much research on data routing approaches in opportunistic networks, which store, carry and forward messages from source to destination nodes via intermediate nodes in a decentralized and distributed environment. The simplest approach is epidemic routing [48]. When a node encounters another node, they exchange messages that the other node does not possess. The protocol obviously incurs the highest transmission and storage costs. However, in practice the number of messages exchanged during each contact between two mobile nodes is limited by the duration of the contact. To improve the message delivery success ratio, nodes coming into contact should exchange only those messages that have a higher probability of being delivered to their destinations when processed by the receiving nodes. Many different approaches exist to optimize epidemic routing by reducing the number of copies of messages sent over the network. Some of these approaches are summarized in Table 2.2. The basic idea of the existing work is to spread the message to a small set of nodes that have a high
probability to deliver the message to the destination [48]. Much of the existing work made predictions based on the mobility pattern [43] and the contact history [49] of users. Recently, social relationships have also been explored to divide users into different social groups to better predict their meeting probability [49-52]. The intuition is that people meet more often if they are friends, family, or colleagues. By exploring the social relationships, we can understand and predict the meeting pattern of people more accurately.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Example(s)</th>
<th>Description</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message reduction based</td>
<td>Spray and Wait [48]</td>
<td>The number of message copies entering the network is limited.</td>
<td>Low delivery ratio and high network overhead</td>
</tr>
<tr>
<td>Mobility pattern based</td>
<td>Message Duplication Reduction [53]</td>
<td>When nodes carrying the same message meet, the node predicted to have the earliest encounter with the destination node keeps the message while the other nodes discard the message.</td>
<td>Low delivery ratio and low network overhead</td>
</tr>
<tr>
<td>Contact history based</td>
<td>PROPHET [54] SimBet [49]</td>
<td>Uses the history of past encounters between nodes to predict whether or not an encountered node can deliver a message to a destination.</td>
<td>High delivery ratio and high network overhead</td>
</tr>
<tr>
<td>Predefined Interest based</td>
<td>Dynamic Groups based [52]</td>
<td>Messages are sent via relay nodes in the same group with same declared interest in the network.</td>
<td>High delivery ratio and low network overhead</td>
</tr>
<tr>
<td>Social relationship based</td>
<td>MobiClique [50] SimBet [49] Bubble Rap [52]</td>
<td>Messages are forwarded based on the social relationship declared/found in the network.</td>
<td>High delivery ratio but a long warm up time is required</td>
</tr>
</tbody>
</table>

Table 2.3: Data dissemination approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Supported by</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic groups based [49]</td>
<td>Predefined interests</td>
<td>Group messages are flooded within that interest group.</td>
</tr>
<tr>
<td>Cooperative user-centric information based [55]</td>
<td>Predefined interests</td>
<td>When nodes meet, they exchange summaries of stored data items and request all interested data items.</td>
</tr>
<tr>
<td>Cognitive heuristics based [56]</td>
<td>Predefined interests</td>
<td>Each node associates a utility function with each data item. The utility function is determined based on the number of encountered nodes that are interested and the number of times that the data item has already been disseminated.</td>
</tr>
<tr>
<td>Contentplace [57]</td>
<td>Estimated social relationships</td>
<td>The utility function used in this approach is the sum of a data item’s access probability for its community divided by the size of the data item and the social weight of a user’s association with a community.</td>
</tr>
<tr>
<td>Publish/subscribe based [58]</td>
<td>Estimated social relationships</td>
<td>A socially aware overlay network is built between nodes that have a higher closeness centrality within their communities. Data are disseminated using publish and subscribe operations via these nodes.</td>
</tr>
</tbody>
</table>

Some research has also been conducted on data dissemination approaches that disseminate a type of message to as many interested nodes as possible. These approaches are
summarized in Table 2.3.

**Context-awareness**: Context-aware services [59] have emerged as an exciting new area of research in the mobile and ubiquitous computing communities, which has the potential to create many new revenue streams for content/service providers of MSNs. Due to the mobility of users in MSNs, it is very difficult for content/service providers to be aware of the personal status of users at a specific time and specific place in an efficient manner, since each user’s activities, interests, and objectives are very diverse and depend on many unknown parameters. Therefore, one key application of context-awareness for MSNs is to ensure that the server side of MSNs can provide appropriate, useful, and relevant contents and services to mobile subscribers anytime and anywhere.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Examples</th>
<th>Description</th>
<th>Features</th>
</tr>
</thead>
</table>
| Recommendation system based     | Contextualized recommendation [60] SocialFusion [61] | Evaluates a set of social conditions and context-aware actions, and to prove whether or not the conditions have been logically verified for both individual and groups of MSN users | - Focus on analyzing and discovering the social relationships between users  
- Fuse the data streams from mobile users, social networks, and sensors |
| Social identity based           | WhozThatFor [62]          | Supports increasingly complex MSN applications                              | - Online social identities with environment adaptive ecosystem                                                                  |
| Proximity based                 | G2G [63]                  | Provides a location-aware MSN platform for personalized recommender systems and gaming | - Detect preferences of nearby friends  
- Collection of geographic-proximity information about GSM cells                                                                 |
| Architectural system based      | Agent-based system architecture [64] | Provides system level approaches to effectively collect, process and utilize contextual data from different source; and facilitate development and deployment of context-aware MSN applications | - Based on open interfaces between social networking platforms |

Currently, a smartphone may utilize its components such as camera, GPS, microphone, and Bluetooth radio to collect sensing data. As well, the application programming interfaces (APIs) of mobile OSs may provide access to contact lists and calendars that can reveal some basic information about the smartphone users. These data can provide context information,
such as a user’s current location and activity, to the MSN platform that runs on the mobile OS of the smartphone. Then, based on such information, the MSN platform can gather and process the information, determine its value, interact with the MSN application on top of it, and upload such information to the server side, hence extending the services and functions of the MSN application by making the application context-aware. Some research work in this area found in the literature is summarized in Table 2.4. However, as identified in Chapter 1, the current crowdsensing approaches are missing automatic methods to form appropriate crowd groups for finishing crowdsensing tasks, e.g., the allocations of participants in the collection of different contextual data in MSNs. Thus, an approach is needed to automatically match multiple participants with multiple mobile crowdsensing tasks in MSNs.

2.1.3 Logical view of MSNs

The end users are at the terminal ends of MSNs. They are only concerned with the functionalities of MSNs. Meanwhile, the end users could provide feedbacks on the MSN applications to the developers, and update specific social contents to the social website via their mobile devices. With the integration of the traditional Internet communication and opportunistic network architectures, many new features and services can be extended to end users in future MSNs.

**Messaging/File transfer:** Internet connectivity is sometimes expensive and slow. It may not even be available at some locations, e.g., in rural regions and disaster areas. Future MSNs will enable users without Internet connectivity to access the Internet via the connectivity of peers who are willing to provide relaying services by receiving, storing and forwarding messages [65] or files [66] through their mobile devices.

**Media streaming:** Cooperative media streaming services are proposed in [67] for future
MSNs. All mobile devices send their location information to a centralized server via the Internet. The centralized server sends commands to mobile devices so that all of the mobile devices can connect together via ad-hoc connectivity such as WiFi Direct by moving to a specific location. Some of the mobile devices are further connected to the centralized server via the Internet. Media streaming services can then be shared among these mobile devices with the advantage of the high speed ad-hoc connectivity.

*Social contents dissemination:* With the integration of the traditional communication and opportunistic network architectures, social contents such as news, weather forecasts, traffic alerts, and social media can be retrieved from the traditional communication network by any node initially and then shared with all others over opportunistic networks. Even if the traditional communication network becomes unavailable, MSNs may still be able to provide some services over opportunistic networks. In addition, whereas conventional online social networks rely on traditional communication networks to support interactions between users (e.g., when user A sends a message to user B, the message is first passed to the server side and then retrieved by user B, over the traditional communication network in both cases), in MSNs, peer-to-peer data exchanges will be used to support direct short-range communications between users in the same vicinity. For example, exchange of business cards with colleagues and photos with friends will be carried out through short-range communications between the mobile devices. Users can exchange data with each other opportunistically, which gives more flexibility in communications without relying on the traditional network infrastructures.

Besides, mobile users can access a great deal of useful local contents such as news, weather forecast, traffic alerts, and social media via the Internet. The contents are often of
interest to nearby users. MSNs enable users to get such contents from other mobile users without accessing the Internet. Opportunistic data exchange in future MSNs facilitates context-aware and social-oriented information sharing considering the locations, environmental context, and social interaction of the mobile users.

For example, in [58], contents are disseminated among mobile devices using a publish-and-subscribe model. In addition, micro-blogging services such as Twitter enable users to send short messages that are followed by a public audience as in conventional MSNs. Future MSNs will allow users to share micro-blogs directly over opportunistic networks [68]. The localized social structures may help to deliver micro-blogs to interested recipients in an effective manner. It will also be possible to search for information locally in future MSNs. A query can initially be propagated to a mobile device in a specific geographic area via a centralized server, and then further propagated between neighboring nodes over opportunistic networks [69].

**Neighbor discovery:** Neighbor discovery will be a vital service in future MSNs. Interactions between physically proximate people were facilitated in [70] by using Bluetooth discovery to find nearby devices and a centralized server to match the profiles of users. With this service, conference participants can find the right people to meet, large companies can facilitate internal collaboration between employees, and individuals can find people with common interests in various social environments.

### 2.2 Features, System Architecture and Challenges of VSNs

In many parts of the world, people spend a considerable amount of time on their daily commute to and from work. Commuters often follow the same routes at about the same time. Since their travel patterns are highly predictable and regular, it is possible to form recurring
virtual mobile communication networks and communities between these travelers and/or their vehicles [25]. VSN involves social networking that occurs in a virtual community of vehicles, their drivers, and passengers, where one or more individuals in such an environment have similar interests, objectives, or other characteristics, and have the capability to interact with each other. Research has shown that knowledge of the social interactions of nodes can help to improve the performance of mobile systems [4]; therefore it is anticipated that VSN applications can be widely used in many fields. The three most common types of applications over VSNs are: (a) Safety improvements: applications that improve the safety of drivers; (b) Traffic management: applications that provide users with up-to-date traffic information enabling them to improve route selection, traffic efficiency and driving behavior [71]; (c) Infotainment: applications that enable the dissemination, streaming, downloading, or sharing of location-dependent information such as advertisement, and multimedia files such as audio and video over the VSN [72].

<table>
<thead>
<tr>
<th>Types of social relationships</th>
<th>Normal MSNs</th>
<th>VSNs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Between human and human</td>
<td>• Between human and human</td>
</tr>
<tr>
<td></td>
<td>• Between human and machine</td>
<td>• Between human and machine</td>
</tr>
<tr>
<td></td>
<td>• Between machine and machine</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network architecture</th>
<th>Internet and opportunistic networks</th>
<th>Internet and VANETs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic rate of network topology</td>
<td>Random, depends on the specific scenarios</td>
<td>Usually very high due to the movement speed of vehicles</td>
</tr>
<tr>
<td>Energy constrains</td>
<td>High constrains and sensitive for most mobile devices</td>
<td>It may not sensitive, as mobile devices could be charged in vehicles</td>
</tr>
<tr>
<td>Real-time requirements</td>
<td>Normal, limited time-latency is acceptable</td>
<td>High, especially for safety applications</td>
</tr>
</tbody>
</table>

Table 2.5: Difference between normal MSNs and VSNs

Different from normal MSNs, where the participants are human beings who interact with one another using smartphones and mobile devices, the participants of VSNs are heterogeneous, and include vehicles, devices onboard vehicles, as well as drivers and passengers. Thus, three types of relationships are found in VSNs: (i) between humans, (ii)
between humans and machines, and (iii) between machines and machines. Also, due to the features of vehicles and the special application environments, the VSN systems have unique characteristics that distinguish them from normal MSN systems, as summarized in Table 2.5.

Figure 2.4: The physical architecture of VSNs

The architecture of a VSN system is shown in Figure 2.4, in which the “embedded system in vehicles”, “pedestrian”, and “roadside infrastructures” can be mapped to the common mobile nodes of future MSNs shown in Figure 2.2, and VANET can be considerable as a kind of opportunistic networks in transportations. VSN systems are built on top of vehicular networks that provide connectivity between users and devices participating in the VSN as well as the Internet at-large. While cellular networks can provide such connectivity, the cost may be too high and the latency too large. Instead, a vehicular ad-hoc network (VANET) may be established to connect the users and devices onboard vehicles that are physically close to each other [73]. For example, drivers or passengers may carry smartphones with various forms of wireless communication capabilities (e.g., 3G cellular, WiFi Direct), or vehicles may be equipped with embedded computing and networking systems, which support the use of Dedicated Short Range Communications (DSRC) for vehicle-to-pedestrian (V2P) [26], vehicle-to-vehicle (V2V), and vehicle-to-
roadside (V2R) communications, with deployed roadside infrastructures in the latter case.

To realize this architecture, as shown in Figure 2.5, we should first establish a vehicular network, such as a VANET, which enables spontaneous communications among the devices onboard vehicles and users' mobile devices, and determine its communication specifications. The second step is to develop a software platform (e.g., a middleware) based upon mobile OSs on which to develop and install different applications that function in the VSNs. A number of research projects about VANETs had been conducted, such as investigating stable routing protocols [74], and adopting distribution-adaptive distance with channel quality for multi-hop wireless broadcast [19].

On the other hand, existing research works on designing VANET applications fall mainly into middleware approach [75]. Middleware is a layer of software that manages the interactions between applications and the underlying network by providing various services to the applications. Using a middleware to provide a common set of services for VSNs can simplify the application development process. However, existing middleware for mobile networks are tightly coupled with the applications and thus can hardly meet the highly heterogeneous and ubiquitous service requirements of different users in VSNs [76].
Consequently, a new programming framework with features of automatic and decentralized network management, scalability, and universality (i.e., platform/hardware independence) still needs be investigated.

Furthermore, due to the dynamism of VANETs and the opportunism of user connections in a VSN, dynamic network connectivity (e.g., as vehicles move at high speeds, the wireless links may become unreliable and have short lifetimes) and diverse service requirements of different users in VSNs, which may vary with user locations, time periods, and/or traffic situations, are unique and important challenges of the VSN applications [77]. Thus, effective solutions which can improve the autonomy and self-adaptivity of crowdsensing applications to dynamic service requirements of users in VSNs are needed. Context information in a VSN refers to the information related to the characteristics and specific situations of users, such as their locations, identities, roles, activities, the time at which they engage in the activities, their preferences in interactions with others, local environmental conditions, the current network status, and so on. Also, semantic techniques can be used to formally represent the context information of users and service properties at a high level of abstraction, thus enabling automated reasoning on the represented context information. Such reasoning could facilitate the interactions between entities even if their statuses are unknown to each other [28]. Therefore, the integration of context information and semantic techniques in VSNs can potentially provide a systematic approach to improve the context-awareness of crowdsensing applications for VSN users in highly dynamic vehicular environments [78].
Chapter 3: Application-oriented Service Collaboration Model for Task Allocation of Crowdsensing

3.1 Introduction

Because of the inherent nature of crowdsensing, usually a crowdsensing task needs a group of different users to finish it collaboratively, thus the matching between multiple crowdsensing tasks and multiple users is important. Also, such approaches are fundamental to enable broad-based application support for crowdsensing in VSNs [12, 13, 14], e.g., collaborative route POIs in traveling [12]. However, as we discussed in Subsection 1.1.1, a number of researchers have identified that the current crowdsensing approaches are missing automatic methods to form appropriate crowd groups for finishing crowdsensing tasks [10, 11, 12]. Neither the existing social communication creation approach [15] nor the opportunistic sensing-based approach [17] can address this challenge well, thus:

- An approach that can automatically match multiple participants with multiple crowdsensing tasks to form appropriate crowdsensing groups, so as to finish the crowdsensing tasks efficiently and effectively in VSNs, needs be explored.

In this chapter, we design a novel application-oriented service collaboration model (ASCM), so as to achieve automatic matching between multiple users and multiple mobile crowdsensing tasks efficiently and effectively. Also, one of the key challenges to validate our ASCM approach is that currently there is no baseline and simulation platform could be used, since the research about automatic matching of mobile crowdsensing still in the infancy stage. Therefore, we develop a generic mobile software platform called Vita [140], to verify the effectiveness and feasibility of the ASCM approach, and investigate how to facilitate the realization of ASCM in real-world deployment. The success criteria of the
design of ASCM are summarized in Table 3.1 as following.

Table 3.1: Summary of success criteria of the design of ASCM

<table>
<thead>
<tr>
<th>Category</th>
<th>Success criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility of ASCM in real-world deployment</td>
<td>(i). Prototype mobile crowdsensing applications, which can demonstrate the usefulness and expressive of ASCM in helping task allocation of crowdsensing to users in real-world, e.g., assisting users to find out the POIs when they travel in a new city.</td>
</tr>
<tr>
<td></td>
<td>(ii). The overhead of mobile crowdsensing applications deployed on Vita should be reasonable for popular mobile devices (i.e., smartphones) when applying ASCM, e.g., basic data overhead, computational overhead, and overhead in finishing concurrent tasks should be acceptable for popular smartphones.</td>
</tr>
<tr>
<td>Efficiency of ASCM</td>
<td>Time delay in task allocation of mobile crowdsensing applications should be low, e.g., the response time between cloud servers and mobile devices should be minimized. Although there are no standards to define the time efficiency in crowdsensing, a number of research and surveys have demonstrated time efficiency is a concern for both participants and requesters of crowdsensing in common situations.</td>
</tr>
<tr>
<td>Effectiveness of ASCM</td>
<td>Every participant in the crowd groups should have knowledge and resource in a wider range in finishing the crowdsensing tasks, when allocating multiple participants to multiple mobile crowdsensing tasks in mobile devices’ run-time.</td>
</tr>
</tbody>
</table>

Our major contributions presented in this chapter are as follows:

• Based on our best understanding, ASCM is the first work that investigates how to quantify the many to many relations between each context in crowdsensing, and supports automatic allocations of multiple crowdsensing tasks with multiple crowdsensing participants across mobile devices and cloud platform in an efficient and effective manner.

• We design and develop a generic RESTful Web Service based mobile cloud platform called Vita, which provides practical operating experience to verify the feasibility of ASCM and facilitate the realization of our ASCM approach in real-world deployment in VSNs. Also, the Vita platform provides a generic model to deploy and evaluate different context-aware solutions for mobile crowdsensing applications.

The rest of this chapter is organized as follow. The design and architecture of Vita, in
particular the design of ASCM and its working theory for task allocation of mobile crowdsensing applications are elaborated in Section 3.2. The implementation strategies of the Vita cloud platform, ASCM, and the mobile SOA framework are presented in Section 3.3. In Section 3.4, we first introduce a qualitatively prototype crowdsensing application called Smart City developed on Vita to demonstrate the expressivity of ASCM, and then we do quantitative evaluations and comparisons to validate the efficiency and effectiveness of our ASCM approach. Section 3.5 reviews the related works, and analyzes the advantages and disadvantages of our solution compared to them. Section 3.6 summarizes the chapter.

3.2 System Design

![System Architecture of Vita](image)

As shown in Figure 3.1, the overall architecture of the Vita platform consists mainly of two parts: the mobile platform and cloud platform. The mobile platform provides the initial environment and ubiquitous services to enable users to participate in and operate crowdsensing tasks through their mobile devices. The cloud platform provides a central coordinating platform to allocate multiple mobile crowdsensing tasks to different mobile
devices, and to store the diverse data and tasks from crowdsensing and social networking service providers.

**Mobile SOA framework:** The mobile SOA framework (along with the mobile SOA server shown in Figure 3.1) is an extensible and configurable framework that is based on the specifications and methodologies of RESTful Web Services [79]. It integrates popular social networking services (i.e., Facebook and Google+) and works as a bridge between the mobile platform and cloud platform of Vita over the Internet. The implementation strategies of this framework will be introduced in Subsection 3.3.3.

**Vita cloud platform:** For the design of the cloud platform of Vita, we adopt the RESTful Web Service-based architecture design methodologies and specifications, so as to provide an open, extensible and seamless architecture across the mobile and cloud platform of Vita. The cloud platform of Vita mainly consists of four components: management interface, storage service, deployment environment, and process runtime environment. More implementation details about it will be provided in Subsection 3.3.2.

### 3.2.1 Application-oriented service collaboration model (ASCM)

![Figure 3.2: System model of ASCM](image)

As shown in Figure 3.1 and Figure 3.2, the ASCM mainly consists of three parts: *Task Client*, *Transformer*, and *social vector*. The *Transformer* can transfer the different mobile crowdsensing application requests and tasks (which are *task instance data* based on the...
specifications of WS-HumanTask [80]) from users to a standard data format as a web service [81, 82] during a mobile device’s run-time; implementation details about it will be introduced in Subsection 3.4.2. The Task Client could receive the application requests of crowdsensing users and invoke the existing services in the mobile device help to accomplish the tasks. Also, the Task Client can invoke the social networking services incorporated in the mobile devices to obtain social information. The social vector can obtain application tasks information from the Transformer, and service and social information from the mobile SOA framework through the Task Client. Based on these, the social vector quantifies the distance and relationship between two physical elements (i.e., people, mobile devices, server of the cloud platform) and virtual elements (i.e., software service, computing/human based task) to facilitate the deployment of computing tasks and human based tasks among different devices and users of mobile crowdsensing according to different specific application scenarios in VSNs. As shown in Figure 3.2, the process flow of social vector consists of three steps: A. modeling structures and relations of contexts, B. calculating distances between entities, and C. optimizing the forming of crowd group.

3.2.1.1 Modeling structures and relations of contexts

Different from the former work [83-86] about opportunistic sensing in mobile embedded system, which mainly focus on the optimization of the system’s computation, in the design of the ASCM, based on the human-centric principle, we adopt a top-down design methodology. We suppose that for each mobile crowdsensing application, since every element has some inherent relations with a crowdsensing user, such as a task is finished by some people using their mobile devices, and a mobile device (contains the services available on it) belongs to a user, thus the allocation of the tasks (both computing and human-based)
could be based on the relation between them and the users.

The *social vector* is a tool in ASCM, which is applied to optimize the allocation of multiple crowdsensing tasks to groups of physical elements (e.g., people, cloud servers). To model the structures of the crowdsensing task context, user’ context, and crowd group context, *social vector* first defines four types of attributes in each context: *continuous*, *discrete*, *Boolean* or *string*.

**Continuous attributes:** It includes the information of a user’s mobile device like computation power (e.g., the number of floating-point operations that the device can perform per second), communication capacity (e.g., bandwidth available for communicating of the devices), and remaining battery time, which are obtained via a general API in the mobile device.

**Discrete attributes:** It includes the number of similar tasks executed, the number of remaining tasks, and the number of reused resources for a particular crowdsensing task, which are updated and recorded in each mobile device individually whenever a task is completed.

**Boolean attributes:** It includes whether a mobile user has related knowledge or prefers to help in a particular task, which are input by users after the deployment of a mobile crowdsensing application on mobile devices.

**String attributes:** It includes the records of a user’s historical location (e.g., the names of the places she has been visited), kind of the vehicle the user own, and her social attributes (e.g., gender, age, cultural background, and professions).

In essence, for a crowdsensing task, an *m*-dimensional *social vector* to model the structure of it can be defined as:
\[ \overline{V_T} = [a_1, a_2, ..., a_m] \]  

(3.1)

which has \( m \) attributes, it means this crowdsensing task needs \( m \) kinds knowledge/service help to finish it, or say this task can be divided up to \( m \) sub-tasks. For example, for a crowdsensing task in the application scenarios of collaborative route planning to POIs in VSNs as mentioned in Subsection 1.1.1, it could be divided to two sub-tasks like \( a_1 \) - finding the names of the POIs, \( a_2 \) - finding the best routes to the POIs.

Also, suppose that there are \( M \) mobile users, which are \( \{U_1, U_2, ..., U_M\} \), and \( N \) multiple mobile crowdsensing tasks in VSNs, which are \( \{T_1, T_2, ..., T_N\} \). A straightforward but ineffective way is to assign the tasks to the mobile users randomly. A better approach is to assign each task to a group of mobile users with the most appropriate resources and knowledge to carry out the tasks (i.e., based on their attributes as defined in the social vectors). Assume that there is a mobile user \( S \) has the most appropriate resources and knowledge to execute the task \( T_i \). We can define an ideal social vector

\[ \overline{V_{T_i, S}} = [\overline{V_{T_i, S}(1)}, \overline{V_{T_i, S}(2)}, ..., \overline{V_{T_i, S}(m)}] \]  

(3.2)

Besides, assume that there is a group \( G \) formed by a group of user \( U_j \), where \( j = 1, 2, ..., L_i \) and \( \sum_{i=0}^{N} L_i = M \). The group social vector between the group \( G \) and the task \( T_i \) is defined by

\[ \overline{V_{T_i, G}} = \frac{1}{L_i} \left[ \sum_{j=1}^{L_i} \overline{V_{T_i, U_j}(1)}, \sum_{j=1}^{L_i} \overline{V_{T_i, U_j}(2)}, ..., \sum_{j=1}^{L_i} \overline{V_{T_i, U_j}(m)} \right] \]  

(3.3)

where \( \sum_{j=1}^{L_i} \overline{V_{T_i, U_j}(x)} = a_x \). Note that for each group, the group social vector gives the centroid of the group. A requestor of crowdsensing seeks to find a group of \( L_i \) mobile users \( U_1, U_2, ..., U_{L_i} \) help to finish a crowdsensing task, such that if the \( \overline{V_{T_i, G}} \) is the closest to the ideal social vector \( \overline{V_{T_i, S}} \), then it means a good crowd group for this crowdsensing task is found. Referring
to Figure 3.3, for purposes of illustration, we define a simple *social vector* with two attributes $a_1$ and $a_2$.

3.2.1.2 Calculation of distances between attributes

As discussed above, to find the good crowd groups for crowdsensing tasks, we need to calculate the distances between the attributes in each context, so as to find a best *group social vector* which is the closest to the *idea social vector*. For example, the distance between the *group social vector* $\overrightarrow{V_{T_i,G}}$ and *idea social vector* $\overrightarrow{V_{T_i,S}}$ for a crowdsensing task with $m$ attributes can be defined as:

$$|\overrightarrow{V_{T_i,G}} - \overrightarrow{V_{T_i,S}}| = \sqrt{a_1^2 + a_2^2 + \ldots + a_m^2}$$  \hspace{1cm} (3.4)

where $a_m = \left(\frac{1}{|U_i|} \sum_{j=1}^{l_i} \overrightarrow{V_{T_i,U_j}}(m) - \overrightarrow{V_{T_i,S}}(m)\right)$. Obviously, if the data type of the items in the attributes is continue, distributed or Booleans, we can directly use Euclidean distance to calculate the distance between the items in each attribute. If the type of the items in the attributes is string, we need to calculate the semantic distance between them, e.g., the distance between two words *limo* and *sedan*. 
An ontology is a specification of a conceptualization that describes the concepts and relationships existing in a domain. A number of ontology-based approaches for context-aware mobile service have been proposed [87]. Instead, here we propose a novel ontology-based semantic distance computing method as a lightweight and generic solution for mobile devices, which can make use of different ontology to calculate the semantic distance of two items.

To calculate the semantic distance between two string items (words) in the attributes of social vector, the ontology-based semantic distance computing method executes the query-resource matchmaking component, whose core task is to compute the similarity between items. In this method, the semantic similarity of two concepts (words) is mainly decided by impacts from the following three aspects: the distance (the length of the path) between two
words; the depth of two words and the depth of their most specific common parent in the common ontology; and whether the direction of the path between the two words is changed.

Figure 3.4 shows an example, and more details about this calculation method and the algorithm to find the most specific common parent is introduced in Subsection 4.4.3 of Chapter 4. First, the distance between two words refers to how many steps it takes from one word to reach the other in the ontology; e.g., there are five steps from the word “Izzue” to the word “Gucci handbags”. The more steps between two words, the larger is the distance between them, and the lower is their similarity. Second, considering the depths of two words in the ontology satisfies our intuitive understanding of an ontology’s classification tree: the classification tree is a progressive refinement work; when two classes are deeper in the classification tree, the relationship between them becomes closer, so that the distance is smaller and the similarity is higher. For example, the words “HK fashion” and “HK shirt” are closer than the relationship between the words “Fashion” and “Handbags”. Furthermore, two words have a higher similarity when they are closer to their most specific common parent in the taxonomy tree. Finally, if the direction of the path between two words changes, the two words must not belong to the same tree branch and their similarity should be degraded by assigning a lower score. For example, the path between the word “HK fashion” and “HK shirt” changes direction, while the path between “HK fashion” and “clothing” does not.

3.2.1.3 Optimizing the forming of crowd group

Based on the above two processes, we adopt two intelligent computing techniques: Genetic Algorithm (GA) [88] and K-means [89] clustering as optimization methods in social vector for the forming of crowd groups, when allocating multiple crowdsensing tasks to multiple crowdsensing participants in run-time. Note that GA provides more optimized
solutions but requires more processing, while K-means clustering provides less optimized solutions using less possessing, which may be more suitable in some time sensitive application scenarios of VSNs.

Figure 3.5: GA-based clustering

For purpose of illustration, here we present a GA-based clustering algorithm for assigning people to different tasks. Inspired by the evolution of living organisms, GAs are intelligent computing techniques for finding optimized solutions. Basically, practical solutions are expressed as “chromosomes”, which can be mixed to generate new chromosomes through a crossover process. Sometimes, a mutation process can also be employed to introduce small changes in chromosomes after a crossover operation. After many generations of crossover and mutation operations, a close-to-optimal solution is obtained. Figure 3.5 gives an example, assume there are ten people. Each of them has an identity number \{0,1,...,9\}. Initially, random solutions are generated to assign the people into groups. Every three groups function like a chromosome. Chromosomes are selected based on their fitness values. The fitness value is defined as the distance between the group social vector and the ideal social vector. Then, \( \alpha \) chromosomes are selected through a random process based on their fitness value. For assigning people to different groups effectively, chromosomes with a higher fitness have a higher probability of being selected (higher fitness value implies smaller distance between the group social vector and the ideal social vector). Note that a chromosome may be selected more than once. After selecting \( \alpha/2 \)
pairs of chromosomes, each of them is mixed in the crossover process. Assume that two parent chromosomes, A and B have been selected. After that, $\beta%$ of A is mixed with $(1- \beta%)$ of B to produce C and $(1- \beta%)$ of B is mixed with $\beta%$ of B to produce D. Some identity numbers may need to be added or removed to ensure the identity numbers of all people are in the chromosomes and without duplication. The best two chromosomes among A, B, C and D survive. Finally, any identity number may be replaced by another with a pre-defined mutation probability. The above steps are repeated 1000 times (i.e., 1000 generations).

![Figure 3.6: K-means based clustering](image)

Besides, considering the computationally extensive nature of GA-based algorithm, we also adopt a K-means based clustering algorithm in social vector. An example is shown in Figure 3.6, it randomly assigns people into three crowd groups so that each group has roughly the same number of people. For each person, it can calculate the distance between his/her social vector and the centroid of the social vector of his/her group. For maintaining the desired number of people in each group, a person who is closest in his/her group is
assigned to the second closest group if the number of people in the original group exceeds the predefined limit. The above process is repeated until no people need to be re-assigned from one group to another. In other words, the process ends when all groups become stable. Since the choice of initial people in group can greatly affect the final result (i.e., the smallest sum of distance between group social vectors and ideal social vectors), the best result of multiple trials of different initial people in groups will be adopted.

3.2.2 Working theory of ASCM in Vita for task allocation in crowdsensing

Here we use two examples to illustrate the overall working theory of ASCM with other components of Vita for the task allocation of mobile crowdsensing applications in VSNs. Assume that there are two roles: A – requestor of mobile crowdsensing; B – mobile users who participate in crowdsensing and help to finish the task. The same user may have multiple roles, such as A and B simultaneously, but since the definitions of different roles are distinct and different processes are provided in order to support these roles, thus we present them separately.

A: Requestor of mobile crowdsensing

As shown in Figure 3.7, a requestor of mobile crowdsensing can choose functions through the user interface of her mobile device, according to her application purpose. After the Task Client in ASCM receives the related application request and description, it first assesses whether this application request can be satisfied through the existing services and/or a collaboration of them on the local mobile device and the cloud platform of Vita. If the assessment is positive, then the Task Client will assess whether this request needs service collaboration, and if it finds that the existing services in the local mobile device can directly meet this request, it will then send the request to the Mobile SOA framework and get the
related result. If the existing services cannot satisfy the request, the Task Client will send the collaboration request to the social vector in ASCM. Then the social vector analyzes and decides the method of service collaboration based on the attributes of this request (note that all required information of using social vectors can be obtained via a general API in mobile devices, recorded in each mobile device individually whenever a task is completed or inputted by users after deploying a crowdsensing application of Vita on mobile devices), and then sends the service invocation request to the cloud platform of Vita through the Mobile SOA framework, so as to get the service collaboration results from the other mobile devices (result-M) via the cloud platform and/or the cloud platform itself (result-C). Furthermore, the Mobile SOA framework will combine both of the results (result-M and result-C) and return the result to the Task Client.

Figure 3.7: Process flow of requestor of mobile crowdsensing

If the Task Client finds that the original application request could not be dealt with at the beginning, it will encapsulate the request as a new human based service by the Transformer of ASCM, create the related task request, and try to get the result through
mobile crowdsensing, more details about this process will be introduced in B below. Finally, the Task Client will combine all of the results from the various process branches and return the final results to the user through the user interface of her mobile device.

**B. Mobile users who participate in crowdsensing and help to finish the task**

In Role A, it is possible that during the mobile users’ runtime, the current available services and collaboration results both in their mobile devices and on the cloud platform are unable to meet the purposes of their crowdsensing applications. In this case, Vita can automatically transfer the related service requirements of the users to standard web services, and then deploy the services to the cloud platform through the mobile SOA framework according to the service requirements. After that, other mobile users can participate in crowdsensing and help to finish such a task.

![Diagram of process flow](image)

Figure 3.8: Process flow of participants provide services in mobile crowdsensing

As shown in Figure 3.8, mobile users who plan to participate in and help to finish some application tasks of crowdsensing in VSNs can first submit their specialties and purposes to the Mobile Client through the user interface of Vita. Then, through the Mobile SOA
framework, the expected description of the tasks will be sent to the ASCM and the cloud platform of Vita, and the cloud platform of Vita will push the information to other mobile users of Vita. As discussed above, with the help of social vector in ASCM, the Mobile SOA framework of Vita in the devices of other mobile users can automatically match the existing tasks and the expected tasks (according to their requirements and descriptions), and then return the results to the cloud platform. After that, the cloud platform of Vita will combine all the suitable tasks as a task list, and then return to the mobile devices of users who plan to participate in crowdsensing. Based on the task list and the related description of the tasks in the user interface of their mobile devices, the users can select the crowdsensing tasks they prefer to participate in. After finishing the tasks, they can input their work through the user interface. The Mobile Client in their mobile devices will then submit the work to the Transformer of ASCM, transfer the work to standard web services, and finish the integration of the mobile crowdsensing tasks in the cloud platform of Vita. Finally, the Mobile Client will return the results of crowdsensing tasks with assessment comments from users (who posted the crowdsensing task before) via the cloud platform to the participants.

3.3 Implementation Strategies

In this section, we discuss and present how the main components of the Vita platform and ASCM can be implemented in practice, so as to facilitate the realization of our ASCM approach in real-world deployment.

3.3.1 Implementation of the cloud platform of Vita

We adopt the Amazon Web Service (AWS) infrastructure services (i.e., EC2 and S3) and a series of open source techniques, such as JBoss jBPM [90], Apache ODE [91], Apache Tomcat, BPEL4People and WS-HumanTask [80] for the implementation of Vita’s cloud
platform. Other IaaS cloud computing platforms that are not AWS based could also be used to implement the Vita cloud platform. It consists of four parts: management interface, process runtime environment, storage service, and deployment environment.

**Management Interface:** The management interface is implemented by integrating the Apache ODE management interface, the JBoss jBPM management interface, and the development environment provision interface. For instance, as shown in Figure 3.9, the implementation of the development environment provision provides the ability to download sources and/or compiled releases of the software packages that are required for setting up the development environment, and related documentation and examples.

**Process runtime environment:** As shown in Figure 3.10, the process run-time environment is implemented using two application servers: (a) the Apache Tomcat server for the setting up of the BPEL running environment - Apache ODE; and (b) JBoss AS (Application Server), on which the jBPM process manager is deployed. Based on these two open source business process runtime environments, BPEL processes and their extension BPEL4People processes can be deployed. Also, we use the process adapter to transform the corresponding BPEL4People part into the BPMN implementation of business processes, which can be integrated with WS-HumanTask. Moreover, as in the mobile environment of Vita, its services are REST-based, while BPEL only supports Simple Object Access Protocol (SOAP) based web services. Thus, the BPEL4People
processes are not supported by Apache ODE. To address this issue, we use a REST-SOAP Adapter. This adapter can receive the SOAP service invocation request, and transform this request into the REST service invocation request.

![Diagram of process runtime environment](image)

**Figure 3.10: Implementation of the process runtime environment**

---

**Storage service:** Based on the AWS S3 infrastructure, the storage service wraps the APIs for all of the data storage requirements from other modules: the data for the WS-HumanTask, the related software packages, examples of documents for the development provisioning, and the mobile SOA environment provisioning.

**Deployment environment:** The deployment environment is composed of three modules. We integrate the Apache ODE deployment environment and the JBoss jBPM deployment environment to form a base for the Management Interface to support BPEL and the BPEL4People development environment. Based on the storage service module, we implemented the mobile SOA environment provision module.

### 3.3.2 Implementation of the ASCM

The current implementation of ASCM in Vita is based on the Android OS and open source techniques. As indicated in Subsection 3.2.1, beyond the social vector and Task Client, which could be developed on Android directly, the major implementation task of ASCM is the Transformer. In order to implement the Transformer, we adopt the open source techniques BPEL4People and WS-HumanTask [80]. Based on WS-HumanTask, we develop
a sub-model inside the *Transformer*, which could describe and transfer the heterogeneous human-based task data to standard data format. Then through the *HumanTask Activity* that is developed based on the BPEL4People, the processed data can be encapsulated to standard web service and deployed to the mobile SOA framework as a human-based service. In addition, if a similar human-based task has been deployed and finished in the cloud platform of Vita, the result can be accessed by the *Human-based Services* through the mobile SOA framework.

### 3.3.3 Implementation of the mobile SOA framework

The overall implementation architecture of mobile SOA framework is shown in Figure 3.11. Efforts to migrate conventional SOA frameworks like Jersey or AXIS to current Android OS were unsuccessful due to the limited Java libraries offered by such systems. Thus we adopt a light weight SOA server. There are already some lightweight web servers available in mobile OS, such as the kWS Android Web Server (supports HTTP 1.0 and static web pages) [92], PAW server for Android (supports PHP plugin) [93], and I-Jetty open source web container based on Android (supports Java Servlet) [94]. Based on the Java service implementation we are using, we use I-Jetty to implement the mobile SOA server, which is at the bottom of the architecture. On top of the mobile SOA server are the Java class libraries, which mainly consist of the encapsulated mobile web services and other Java class libraries. All of the web services based on the REST style are deployed in the mobile SOA framework and supported by the underlying Java class libraries. Finally, external to the mobile SOA are the mobile clients implemented to enable specific applications. Application developers of crowdsensing can develop different kinds of mobile clients based on the specific application purpose. One requirement here is that the implementation of a mobile
client needs to correspond to the resource operation of the underlying REST style based web services, and their mapping relation is implemented through the configuration files in the mobile SOA framework.

![Diagram](image)

**Figure 3.11:** Overall implementation architecture of mobile SOA framework

The process flow of mobile SOA framework is as shown in Figure 3.12. During runtime, the mobile SOA framework will firstly receive the web service request from the mobile client, and then analyze the request Uniform Resource Locator (URL) and the related parameters encapsulated by HyperText Transfer Protocol (HTTP), so as to determine the specific Java class to invoke the corresponding web services based on the configuration files. Finally, after the operation of the related web services, the mobile SOA framework returns the results to the client in the form of REST style data through HTTP. One advantage of this architecture design is that the application developers do not need to be concerned with issues of mapping relation about specific URL to corresponding service resources, but can focus on the development of the crowdsensing application itself.
In addition, the communications between the cloud platform of Vita and its mobile platform employ the standard web service format based on the HTTP protocol and Extensible Markup Language (XML) data format. In addition, although BPEL interactions on the Vita cloud side are SOAP based, and its services in the mobile platform are RESTful Web Services based, the SOAP-REST transformation can be achieved using additional adapters in between, similar to the method described above for the cloud platform.

3.4 Evaluation

In this section, we first present a novel mobile crowdsensing application Smart City developed on the Vita platform, to demonstrate the expressivity of ASCM in the application scenarios of our daily life’s traveling. After that, we conduct quantitative evaluations and comparison to validate the efficiency and effectiveness of ASCM when it is deployed on Vita in real-world scenarios.

3.4.1 Application example

As discussed in Subsection 1.1.1, in a mobile crowdsensing task for collaborative route planning to POIs, we can divide this task to multiple smaller tasks and allocate multiple participants to finish these tasks in an effective manner, e.g., assisting the users (or requesters)
of crowdsensing to get satisfied results. Thus, based on Vita and ASCM, we develop the Smart City, a novel mobile crowdsensing application which can assist users to figure out two specific POIs when they are traveling in a new city: (i) Eating place; and (ii) Shopping tour.

In addition, as shown in Figure 13, Smart City consists of another two generic functions: services and crowdsensing. The function of services is based on the mobile SOA framework and takes advantages of the RESTful Web Service that allows developers to flexibly extend new functions to this application; and the function of crowdsensing is based on the ASCM, social network services of the mobile SOA framework, which enables users to post and/or accept crowdsensing tasks by their mobile devices through social networks.

Figure 3.13: Smart City - context-aware mobile crowdsensing application
Here, we set up an experimental application scenario, so as to demonstrate this functions of *Eating place* and *Shopping tour of Smart City* via crowdsensing under simple but common scenarios. In this case, it consists of ten persons each carrying an Android phone and has installed the *Smart City*. Assume that there are two persons have traveled to Hong Kong from Vancouver, and one person called *Blair* wants to taste some local food in Hong Kong, and another person *Betty* wants to do some shopping in Hong Kong.

For the *Eating place* function used by *Blair*, as shown in the left side of Figure 3.13, *Blair* posts the related crowdsensing request through the *eating* function of the *Smart City* on her phone, as follows: *What is the delicious food in Hong Kong?* Based on *Blair’s* social activity records (i.e., the frequency and type of restaurants she visited) stored in the ASCM, *Blair’s* dining habit could be identified showing that she favors non-spicy food. Subsequently, with the help of ASCM and cloud platform of Vita, it can automatically push the request to the rest persons who have similar dining preferences as *Blair* and have experience about the food and restaurants in these two cities (i.e., three of them had lived in Vancouver before). Then, they can upload their comments about the suitable restaurants to cloud platform of Vita no matter they are staying in such restaurants or not. Note that preference with a different dinning preference will not be presented with *Blair’s* request, making sure that the requests are delivered to an appropriate target group that leads to relevant and accurate query result. Also, based on the location service, the request will be pushed to the persons who are staying in the recommended restaurants, and through the user interface of the *Smart City*, they take a photograph of the food and upload to cloud platform of Vita. After the cloud platform of Vita gets all of the data (i.e., photograph, comments, location information, map services), the ASCM can automatically match and combine the
other relevant data stored in it before through the specific information. The overall answers are then returned to Blair’s phone (shown in the left corner of Figure 3.13 which consists of three appropriate answers). Finally, with the help of context-aware mobile crowdsensing through Smart City, Blair enjoys local flavors in Hong Kong that are preferable to her. She wants to share her experience with others, to help the next new visitors. To do this, using her phone she inputs the information similar to those obtained through mobile crowdsensing before. Such information (comments, photograph with address, etc.) can then be directly shared with other visitors with the same request, or stored on the cloud platform of Vita as a new service.

Simultaneously, as shown in the screen shot in the right corner of Figure 3.13, Betty posts the crowdsensing request: Recommendations of branded clothing in Hong Kong via the shopping tour function. As shown in the Unified Modeling Language (UML) diagrams at the bottom of Figure 3.13, with the help of ontology-based semantic distance computing method in ASCM, it could automatically retrieve the implicit context information about her task in two categories: clothing information, e.g., Izzue fashion and Gucci handbags (the stores which she usually visits); and user information, e.g., female, HK branded preference (through the basic information of her social account), average consumption in fashion $350-$900. After that, her task with the context information is pushed to cloud platform of Vita. On the other hand, the ASCM automatically retrieves the related context information of the participants who accepted to join Betty’s task. After the ASCM confirms the shopping information they provide that match the context of Betty’s task, it uploads such information to the Vita cloud platform. Finally, the Vita platform automatically combines all the shopping information from different participants (i.e., participants A and B uploaded the
brand and discount information; participants C and D uploaded the specific location information; participant E uploaded the photos related to the recommended stores) that fits Betty’s crowdsensing task and returns them to her phone. For instance, COUR CARRE clothing (similar to Izzue) has a 10% discount today, and the location is Floor L3, Shop 03.

3.4.2 System and task performance

We evaluate the system and task performances of Vita in terms of three parameters: time efficiency, energy consumption, and networking overhead in mobile devices, when they finish concurrent crowdsensing tasks, as these parameters have great impacts on the experience of mobile users when they are participating in mobile crowdsensing. The experimental environment is: **Hardware:** Amazon EC2 M1 Medium Instance; 3.75 GiB memory; 2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit); 410 GB instance storage; 32-bit or 64-bit platform; I/O Performance: Moderate; EBS-Optimized Available: No. **Software:** operating system: Ubuntu 12.04.1; Servers: ApacheTomcat 7.0.33 and JBoss AS 7.1.1; BPEL engine BPEL4People environment: ODE1.3.5 and jBPM 5.4. **Experimental devices:** Vehicle - 2005 Toyota Sienna; A variety of Android devices are employed, which are described below for each set of experiments.

A. Time delay, basic battery consumption, and networking overhead of Vita

Two Nexus 4 (with LTE module) were used in driving 2005 Toyota Sienna for testing. Each test last 30 minutes, and total of 5 tests were run, and the average results were calculated. We adopt the WordNet ontology [95] as benchmark, and the computation task of data interpretation is to index and calculate the similarities of concepts on this ontology under the condition of four different size assertions (1000, 1500, 2000, 36000), from which we obtained four sets of data correspondingly. For each data set, we test the time efficiency
of the task in two situations: perform the task on Nexus 4, and perform the task by uploading it to the cloud platform of Vita. The distribution of the task according to Poisson distribution with a rate of $E=5/\text{min}$. In addition, we record the network overhead on Nexus 4 when it uploads the task to Vita.

The experimental results are summarized in Table 3.2. The time delay when performing the task via cloud consists of: (i) response and communication time between the Vita cloud platform and the mobile devices; and (ii) processing time of the task. We find that the response time of the four sets of data are similar, with all averaging about 10.5s, while the process time mostly depends on the size of the data set. From Table 3.2, we can see that Nexus 4 gets a better time efficiency when the size of data set is 1000, 1500 and 2000, while the cloud performs much better when the size of data set is around 36000. Since ASCM supports dynamically performing the computation task across mobile device and cloud platform, thus the maximum time delay is around 13s even in the quite intensive computing

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time delay (msec) - via Vita cloud</td>
<td>Response time</td>
<td>Response time</td>
<td>Response time</td>
<td>Response time</td>
<td>Response time</td>
</tr>
<tr>
<td>Max.: 19386</td>
<td>Max.: 20731</td>
<td>Max.: 19376</td>
<td>Max.: 18735</td>
<td>Max.: 138758.3</td>
<td></td>
</tr>
<tr>
<td>Min.: 9571</td>
<td>Min.: 5954</td>
<td>Min.: 9486</td>
<td>Min.: 8693</td>
<td>Min.: 40617.16</td>
<td></td>
</tr>
<tr>
<td>Average: 10673</td>
<td>Average: 10485</td>
<td>Average: 10954</td>
<td>Average: 10577</td>
<td>Average: 63988.86</td>
<td></td>
</tr>
<tr>
<td>Process time</td>
<td>Max.: 489</td>
<td>Max.: 735</td>
<td>Max.: 2834</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Max.: 40</td>
<td>Min.: 447</td>
<td>Min.: 686</td>
<td>Min.: 2156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min.: 40</td>
<td>Average: 456</td>
<td>Average: 706</td>
<td>Average: 2478</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average: 40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time delay - local computation (msec)</td>
<td>Max.: 2287</td>
<td>Max.: 5011</td>
<td>Max.: 1498025</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Max.: 2081</td>
<td>Min.: 4652</td>
<td>Min.: 7997</td>
<td>Min.: 1294654</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min.: 2213</td>
<td>Average: 4736</td>
<td>Min.: 7196</td>
<td>Average: 136073</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average: 2213</td>
<td>Average: 7445</td>
<td>Average: 7445</td>
<td>Average:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Battery consumption</td>
<td>75mAh/30mins</td>
<td>72mAh/30mins</td>
<td>75mAh/30mins</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Network overhead</td>
<td>1.67MB/150 requests</td>
<td>1.69MB/152 requests</td>
<td>1.64MB/147 requests</td>
<td>1.59MB/143 requests</td>
<td>N/A</td>
</tr>
</tbody>
</table>
situation when the size of data set is 36000. Thus, our experiments demonstrate that our ontology-based semantic distance computing method used ASCM can achieve a reasonable time efficiency, and should be feasible to be applied in the task allocations of mobile crowdsensing applications in VSNs for real-world deployment.

Moreover, we make a simple comparison with the related work Medusa [96], where the corresponding time delay is about 64s, although their runtime environments are different. The main reasons are: (a). The Vita cloud platform (in which the ASCM built in) adopts the RESTful Web Service architecture both in its mobile platform and cloud platform, where all of the data and codes of computation tasks offloaded to the cloud from mobile devices are in the format of standard web services. This enables the data transmissions to be more application-related, making the instance object fields a better match with the corresponding methods, and avoiding unnecessary network overhead; (b). The Medusa adopts Short Message Service (SMS) to deliver the task, which involves delays of about 27s.

**B. Time efficiency and computational overhead of concurrent tasks**

As Vita adopts RESTful Web Services, mobile crowdsensing applications deployed on Vita are based on web services. Also, each mobile device may need to execute multiple tasks simultaneously in the practical application scenarios of crowdsensing in VSNs. Thus, we developed a web service based application on Vita, which calculates the representative benchmark *N-Queens Puzzle* [97] both in mobile devices and the cloud platform of Vita, so as to test the efficiency and computational overhead (battery consumption) of the crowdsensing tasks deployed on Vita. In addition, as the communication overhead of the system itself has been evaluated in the last part, and the additional communication overhead mostly depend on the types of crowdsensing applications (i.e., multimedia related or textual
content), thus in this part we skip this aspect. In the experiment, we use Google Nexus S (Android 4.1.2 version, 1500mAh battery capacity), and consider $N$ from 12 to 16. The number of concurrent tasks follows a Poisson distribution with arrival rates of $E=1$ or 3 tasks per minute, respectively. The time efficiency and battery consumption of each task were recorded over 1 hour periods, then the average values were computed. We considered both tasks completed only in the Nexus S in a driving vehicle (2005 Toyota Sienna) with normal speed or by the server in the cloud platform of Vita when the computational task is allocated to it. The screen was shut off during the runtime of tasks.

![Figure 3.14: Execution time and battery consumption of the N-queens puzzle (E=1)](image1)

![Figure 3.15: Execution time and battery consumption of the N-queens puzzle (E=3)](image2)

The results are shown in Figure 3.14 and Figure 3.15. From the results, we find that
Nexus S cannot finish the tasks when $N=16$ even when $E=1$. When $N<15$, both execution time and battery consumption performances of the tasks completed in Nexus S are better than those of tasks uploaded and completed in the cloud. However, when $N=15$, both the time and battery consumptions become much higher when $E$ is changed from 1 to 3. Considering the computationally intensive nature of the *N-Queens Puzzle* (when $N=15$) and the hardware capacity of Nexus S, this means that the web service-based mobile crowdsensing applications deployed on Vita can be finished with high efficiency and low overheads in many of the popular Android devices when performing most of the common computation tasks in VSNs. Furthermore, as introduced in Subsection 3.2.1, the *social vector* in ASCM can optimize the allocation of crowdsensing task, e.g., when $N\geq 15$, the computation task in this test case could be automatically allocated to Vita cloud, as the distance between the *social vector* of Vita cloud and *idea social vector* of this task is becoming closer than the *social vector* of Nexus S with it.

### 3.4.3 Performance of ASCM

As mentioned in Subsection 3.2.1.3, in the *social vector* of ASCM, we propose to incorporate the two approaches - Genetic Algorithm (GA) and K-means clustering as solutions for finding multiple optimized crowd groups among multiple participants in crowdsensing application tasks. Due to the experimental constraints, e.g., it is not easy to find enough volunteers to do the evaluation multiple times in real world, thus, here we set up a Java simulation program to evaluate the performance of these algorithms. Also, to the best of our knowledge, there is no research results published in the literature about how to assign multiple participants to multiple crowdsensing groups yet, thus we make comparisons between our approaches and when forming the crowd groups randomly. The experimental mobile devices are Google Nexus 10 (Android 4.2.1 version, battery capacity 9000mAh).
Each experiment involves 15 participants, each of whom has i social attributes indicating whether the participant has knowledge $K_i$. Each social attribute has a value between 0.0 and 1.0, generated by a random number generator. We assign these participants to K groups (each group corresponds to a task). A social attribute $a_i$ indicates whether a participant P has knowledge $K_i$. In other words, if P has $K_i$, $a_i = 1$; otherwise $a_i = 0$. The social vector between a participant P and a set of knowledge K is defined by

$$\overline{V}_{P,K} = [a_1, a_2, \ldots, a_m]$$

(3.5)

where m is an integer. Assume that there is a task $T_i$ for crowdsensing which is formed by a group of participants $P_j$, where $j = 1, 2, \ldots, L_i$, and $\sum_{i=0}^{N} L_i = M$ (M means the number of mobile users, and N means the number of tasks). The social vector between a group for crowdsensing and a set of knowledge K is defined by

$$\overline{V}_{T_i,K} = \frac{1}{L} [\sum_{j=1}^{L} \overline{V}_{P_j,K}(1), \sum_{j=1}^{L} \overline{V}_{P_j,K}(2), \ldots, \sum_{j=1}^{L} \overline{V}_{P_j,K}(L)]$$

(3.6)

Good crowd groups for crowdsensing tasks are found if $|\overline{V}_{T_i,K}|$ is as large as possible for all i. For GA, the simulation process ends after 1000 generations. As for K-means clustering, the simulation process is repeated until no participant need to be re-assigned from one crowd group to another. We evaluate the effectiveness of GA and K-means clustering, and randomly forming the crowd groups under different required numbers of group (note, here we set the default number of each task’s attributes is 11). Figure 3.16 shows the results and comparison of GA, K-means clustering and random forming of crowd groups. Longer distance between the participants and the centroids of groups implies that every participant in the group has knowledge in a wider range. It can be seen that the average distance between participants and the centroids of the groups decreases when the number of tasks increases. It is because when the number of tasks increases, the probability that participants can be
arranged into a farthest group is decreased. Besides, GA achieves better performance than K-means clustering because it can arrange participants to groups more intelligently, and both the performances of GA and K-means clustering are better than random forming of crowd groups. With GA, the average distance between participants and the centroids of groups is about 0.9 when the required number of groups is 5. With K-means clustering, the average distance between participants and the centroids of groups is about 0.8 when the required number of groups is 5. In contrast, the average distance between participants and the centroid of groups is 0.63 when forming the crowd groups randomly and the required number of groups is 5. In means that no matter adopt GA or K-means in ASCM for task allocation of mobile crowdsensing, the performance is at least 26.98% better than randomly forming the crowd groups in finishing multiple tasks.

Figure 3.16: Evaluation results of GA, K-means clustering and random forming of crowd groups (different no. of tasks)

We further evaluate the effectiveness of GA and K-means clustering under different number of attributes (note, here we set the default number of task is 7). Figure 3.17 shows the results and comparison of GA, K-means clustering, and random forming of crowd groups. Longer distance between the participants and the centroids of their group implies that every participant in the group has different knowledge. It can be seen that the average distance
between participants and the centroids of the group increases when the number of attributes increases. Besides, GA achieves better performance because it can assign participants to different groups more intelligently. With GA, the average distance between participants and community centroids is longer than 1.2 when the number of social attributes is 20. K-means clustering performs worse than GA. With K-means clustering, the average distance between participants and the centroids of group is 1.1 when the number of attributes is 20. With random forming of crowd groups, the average distance between participants and the centroids of group is 0.76 when the number of attributes is 20, much worse than GA and K-means clustering. Thus, it means our ASCM solution performs at least 44.74% better than the common multiple tasks allocation method when forming the crowd groups randomly in this test case.

![Figure 3.17: Evaluation results of GA, K-means clustering and random forming of crowd groups (different no. of attributes)](image)

Note that GA can achieve better performance (i.e., longer distance between the participants and the centroids of their groups) than K-means clustering but the computation overhead and response time of GA is much higher than K-means clustering. Thus GA should be employed only if the accuracy of assigning participants to different groups outweighs the response time. Also, both the response time of K-means clustering and random forming of
crowd groups is pretty low, lower than 100ms in all of our test cases. Thus, in most cases, K-means clustering should be employed because of the limited processing power of the system and requirements of time efficiency for the crowdsensing applications in VSNs.

3.5 Related Work

There have been several research works about mobile distributed systems for crowdsensing. The different works are differentiated by: (i) Targeting some specific application scenarios, e.g., using smart phone to provide feedbacks to users about their transportation behaviors [98], monitoring personal heart information [99], and predicting bus arrival times [100]; (ii) Using specific techniques to construct standalone crowdsensing system to support multiple mobile crowdsensing applications, e.g., using Twitter [101], and using an in-node hardware abstraction layer and overlay management protocol to allow multiple applications sharing sensing data across different mobile nodes [102]; and (iii) Supporting efficient development of different mobile crowdsensing applications; e.g., the work in [103] aims to enable developers to write server side programs in lieu of distributed programs, so as to ease the development of crowdsensing applications on smartphones.

Partly inspired by but different from these works, Vita provides a general approach at the system and architecture design level by leveraging the advantages of a number of techniques, which salient parts are integrated and orchestrated into a flexible and generic platform that could be used to deploy, examine and evaluate different context-aware solutions for mobile crowdsensing applications. Also, the ASCM designed in Vita supports the quantification of many to many social relationships, and effective allocations of multiple crowdsensing tasks with multiple participants. In the following we contrast Vita with two existing works that are close to our work.
Medusa [96] is a programming framework that provides a programming language and a runtime system for efficiently developing mobile crowdsensing applications, which enables reduction of the tasks to smaller pieces compared to standalone systems for crowdsensing applications. It employs a distributed system architecture to coordinate the tasks between cloud servers and smart phones. Compared to Medusa, Vita has several advantages: (i) The ASCM designed in Vita supports the quantification of many to many relationships between each context in crowdsensing. Thus, ASCM simultaneously supports dynamic balancing of the allocation of computation tasks between mobile devices and the cloud platform that enables the task could be finished efficiently, and effective allocation of multiple crowdsensing tasks with multiple participants. (ii) The Amazon AMT adopted by Medusa in the execution of crowdsensing tasks can only be used inside the USA. Moreover, AMT is for commercial use and is not open source. It does not support the development of customized crowdsensing mechanisms and platforms by third parties, while Vita adopts open source techniques and industrial standards such as BPEL4People, and leverages the advantages of social networking services to set up the cloud platform, which can therefore be more universally used, and supports flexible extension and customized redevelopment.

On the other hand, the current weaknesses of Vita comparing to Medusa are mainly in two aspects: (i) Vita lacks a mature incentive mechanism to encourage people to participate in crowdsensing; (ii) The security of diverse crowdsensing applications and data on Vita is still a concern, as the current implementation of Vita does not include any security mechanism. However, as Vita fully adopts an open architecture, thus the application developers of Vita can flexibly design different customized mechanisms to address these issues according to their specific requirements. Further, we shall extend Vita to include
incentive mechanisms and improve security and privacy measures in the future.

ThinkAir [104] is a software framework that supports the migration of computation tasks of smartphone applications to the cloud through mobile code offloading, so as to achieve dynamic computational resource allocation and parallel execution between the smartphone and the cloud computing platform. Different from ThinkAir, Vita adopts the RESTful Web Service architecture both in its mobile platform and cloud platform. In Vita, all of the data and codes of computation tasks offloaded to the cloud from mobile devices are in the format of standard web services. This enables the data transmissions to be more application-related, making the instance object fields a better match with the corresponding methods, and avoiding unnecessary network overhead. Also, Vita supports the reuse of web services. As web services are also written as software codes, web service reuse implies code reuse. However, more than code reuse that usually takes place at development or compilation time, web service reuse further enables sharing of available services among multiple mobile applications during their runtime. This means that rather than balancing the computation tasks between mobile devices and the cloud platform at the coding stage, Vita also enables this balancing to occur dynamically at run time, and mobile users can further act as web service providers by sharing the services available in their devices with other users directly or via the cloud platform.

3.6 Summary

In this chapter, we have proposed the ASCM, a novel application-oriented service collaboration model to approach the automatic allocation of multiple mobile crowdsensing tasks with multiple participants, so as to enable the tasks to be finished in an efficient and effective manner. To the best of our knowledge, ASCM is the first work that investigates
how to quantify the social relationships between participants, crowdsensing tasks and crowd groups in crowdsensing, and supports automatic allocations of multiple crowdsensing tasks with multiple crowdsensing participants across mobile devices and cloud platform efficiently and effectively during mobile devices run-time. Furthermore, we design and develop a generic RESTful Web Service-based mobile cloud platform Vita, to verify the feasibility of ASCM, and explore how to facilitate the realization of ASCM in VSNs for real-world deployment. Also, the Vita platform provides a generic model to deploy, examine and evaluate different context-aware solutions for mobile crowdsensing applications. Based on the works of ASCM and Vita platform, we develop a novel context-aware mobile crowdsensing application – Smart City to demonstrate the expressivity and usefulness of ASCM in our daily lives’ traveling. Our practical experiments have demonstrated the efficiency and effectiveness of ASCM when it is deployed on Vita for multiple crowdsensing tasks in common situations. Also, the experimental results verify ASCM performances better in the allocation of multiple crowdsensing tasks with multiple participants, compared to existing solutions for crowdsensing, which form multiple crowd groups randomly in run-time.
Chapter 4: Agent-based Multi-layer Framework with Context-aware Semantic Service for Crowdsensing Applications in VANET-based VSNs

4.1 Introduction

As introduced in Section 2.2, VSN systems are built on top of vehicular networks that provide connectivity between users and devices participating in the VSN as well as the Internet at-large. While cellular networks can provide such connectivity, the cost may be too high and the latency too large. Instead, in-vehicle and road-side users who are close-by (e.g., traveling on the same street or stretch of highway) can use their mobile devices to inexpensively establish a VANET without the need for accessing the cellular infrastructure. However, as we outlined in Subsection 1.1.2 and Section 2.2, there are two research challenges have not been effectively addressed by the existing solutions yet:

- A framework for the development of robust and reliable crowdsensing applications that self-adapt to the dynamic nature of VANETs is needed but not yet available.

- In users’ dynamic contexts of VSNs, it is challenging for mobile crowdsensing applications to autonomously match appropriate service and information with different users (requesters and participants) in crowdsensing, while a systematic approach is lacking.

Consequently, it is of interest to investigate and develop a software platform that uses the high-level application programming approach to facilitate the development and deployment of a diverse range of self-adaptive mobile crowdsensing applications in VANET-based VSNs. This platform should incorporate services that enable crowdsensing applications to self-adapt to dynamic network connectivity and users’ dynamic contexts within VSNs during runtime. To the best of our knowledge, such a platform does not exist.
currently. This work presented in this chapter fills the gaps identified above by presenting an effective software platform for ubiquitous mobile crowdsensing applications in VSNs.

The **primary contribution** of this chapter is the proposal, design and implementation of S-Aframe, which is the first mobile software platform that supports real-world development and deployment of VSN-based mobile crowdsensing applications on Android devices. S-Aframe hides the complexity of handling changing network connectivity, and varying basic user services requirements of VSNs, by providing a high-level multi-agent programming model with extensibility support. Also, S-Aframe provides a rich set of framework services with a standard Java API format. Therefore, S-Aframe provides a modeling paradigm to facilitate the creation and deployment of different self-adaptive mobile crowdsensing applications in VANET-based VSNs.

The **second contribution** of this chapter is the proposal of a new solution called context-aware semantic service (CSS) that integrates software agent and semantic techniques in S-Aframe to effectively manage and utilize various context information for different crowdsensing applications in VSNs. CSS can enable agent based mobile crowdsensing applications developed on S-Aframe to intelligently determine what and how services and information should be delivered to users in crowdsensing, and autonomously adapt the behaviors of these services to users’ dynamic contexts during run-time, e.g., automatically assist drivers to analyze and accept/reject rideshare requests, as they can hardly use mobile devices while driving.

The rest of the chapter is organized as follows. Section 4.2 reviews related research on VSNs. Section 4.3 introduces the programming model and features of S-Aframe. Section 4.4 presents the design and implementation approaches of S-Aframe, and discusses how S-
Aframe satisfies the key requirements for the execution of crowdsensing applications in VSNs. Section 4.5 evaluates S-Aframe through a set of practical experiments. Section 4.6 presents *smart ride*, a practical crowdsensing application of VSNs developed based on S-Aframe. Section 4.7 summarize the chapter.

### 4.2 Related Works

In this section, we review the related work in the literature, which inspire the design of S-Aframe with CSS, and discuss the difference between them and our approach. Following the classification schemes of self-adaptive systems provided by [105], we summarize them as shown in Table 4.1.

**Table 4.1: Summary of comparisons with related works**

<table>
<thead>
<tr>
<th>Project</th>
<th>Object</th>
<th>Realization approach</th>
<th>Temporal</th>
<th>Interaction support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmbientTalk [20]</td>
<td>application</td>
<td>Development support: Internal adaptation by language - AmbientTalk</td>
<td>Reactive adaptation and continuous monitoring</td>
<td>Human involvement: No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run-time support: handling network failures of dynamic networks</td>
<td></td>
<td>Interoperability support: No</td>
</tr>
<tr>
<td>Agilla [24]</td>
<td>application</td>
<td>Development support: External adaptation by middleware; language - nesC</td>
<td>Reactive adaptation and continuous monitoring</td>
<td>Human involvement: No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run-time support: Agents can dynamically respond to a changing wireless sensor network</td>
<td></td>
<td>Interoperability support: Yes</td>
</tr>
<tr>
<td>MobiSN [30]</td>
<td>application</td>
<td>Development support: No</td>
<td>Reactive adaptation</td>
<td>Human involvement: Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run-time support: Auto semantic-based social grouping</td>
<td></td>
<td>Interoperability support: No</td>
</tr>
<tr>
<td>RoadSpeak [25]</td>
<td>application</td>
<td>Development support: Specific Java APIs</td>
<td>Reactive adaptation and continuous monitoring</td>
<td>Human involvement: Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run-time support: Auto location &amp; time based social grouping, dynamic voice chat matching</td>
<td></td>
<td>Interoperability support: No</td>
</tr>
<tr>
<td>SCDSCF [28]</td>
<td>application</td>
<td>Development support: No</td>
<td>Reactive adaptation and continuous monitoring</td>
<td>Human involvement: No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run-time support: Application composition and semantic-based user context acquisition</td>
<td></td>
<td>Interoperability support: Yes</td>
</tr>
<tr>
<td>S-Aframe</td>
<td>application</td>
<td>Development support: External adaptation by framework; language -Java with standard API format</td>
<td>Reactive adaptation and adaptive monitoring</td>
<td>Human involvement: Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run-time support: Applications can self-adapt to changing network connectivity status and users’ dynamic situations of VSNs during mobile devices’ runtime</td>
<td></td>
<td>Interoperability support: Yes</td>
</tr>
</tbody>
</table>

RoadSpeak [25] is the first framework proposed for VSNs, which allows commuters to
automatically join voice chat groups on roadways. Unlike traditional social networks, RoadSpeak considers, in addition to the interests of users, the time intervals and locations in its definition of the VSN profile when user groups are formed. RoadSpeak partially supports extensibility. It provides a number of Java APIs to application developers, based on which developers can extend RoadSpeak clients to provide enhanced functionality. Nevertheless, RoadSpeak relies on client-server interactions. In a VANET environment, it is difficult to provide a stable server among the vehicles all the time. In addition, RoadSpeak only provides a voice chat service. Its extension support is merely for the grouping of membership in this service, and not for application developers to extend it to provide other services and functions for VSNs. Thus, it can hardly fulfill the diverse and ubiquitous service requirements of different users of crowdsensing in a VSN.

Recently, several semantic based service systems over dynamic networks have been proposed [28, 30]. Fujii et al. proposed a semantic-based context-aware dynamic service composition framework (SCDSCF) [28] using the semantics of components and user contexts over the World Wide Web. SCDSCF consists of a strict abstract model for modeling semantics, a supporting middleware, and a service composition mechanism. However, it models the elements in each component, including the semantics, contexts as well as the user, in a strict manner, which results in significant efforts for programmers to follow for further application development. It also causes much redundancy in information representations that may affect its efficiency. The framework may offer a generic approach for the Web, but is not necessarily suitable in the context of VANETs. MobiSN [30] is a framework proposed for social network construction with mobile phones, which incorporates a semantic-based matching method that works with user profiles. The
framework only addresses a specific application domain, i.e., social grouping, and does not provide a generic approach for a range of applications. MobiSN also lacks support for interoperability among distributed elements and does not consider adaptation of applications to changing contexts in dynamic environments; thus it could hardly be used for VSNs.

AmbientTalk [20] is an experimental programming language for developing mobile peer-to-peer applications. A feature of its programming paradigm is that it accounts for the possibility of network failures in its programming model. In addition, it employs a purely event-driven concurrency framework based on actors. In AmbientTalk, actor executions can be concurrent with asynchronous actor method invocations; thus AmbientTalk is very suitable for highly dynamic networks. However, AmbientTalk is a completely new language, which means that programmers have to spend considerable time to become familiar with it before using it to develop applications for VSN systems. Furthermore, AmbientTalk does not provide any key services to handle the unique challenges of VSN applications, thus handicapping the application development of VSNs. In the implementation of S-Aframe, we encapsulate AmbientTalk as new Java libraries by making use of its symbiotic programming mechanism, and integrate new generic services into its multi-layer framework with an agent-based programming model. Consequently, S-Aframe extends AmbientTalk by retaining its advantages, while further providing a modeling paradigm with CSS to facilitate the development and deployment of context-aware mobile crowdsensing applications for VSNs using Java.

4.3 Model

The programming model of S-Aframe is shown in Figure 4.1. It is designed to facilitate the development and deployment of context-aware mobile crowdsensing applications on
mobile devices over VANET-based VSNs. A software agent can be seen as a composition of software and data, which is able to function autonomously. Mobile agents are software agents that act on behalf of their creators and can function without a continuous active connection; e.g., after they have migrated to a node, even if the connection is broken, they can autonomously proceed to process the data, and return the results and/or migrate to the next node when the connection resumes. Thus, mobile agents are well-suited for executing applications in the dynamic network environment of VANETs. Moreover, mobile agents can be adopted to realize savings in terms of network traffic load by transmitting the agent code instead of raw data over VANETs. Consequently, an agent based programming model is adopted in S-Aframe as shown in Figure 4.1.

![Figure 4.1: Programming model of S-Aframe](image)

### 4.3.1 Scalability

Conventional agent systems are usually designed for specific applications. In systems such as Agilla [24], mobile agents contain all the relevant codes when they want to accomplish certain tasks, although some of the codes may never be used, resulting in wasted network resources. On the other hand, this approach requires developers to develop a completely new mobile agent whenever a new application of crowdsensing is to be
implemented; this is so even when the new application shares common features with existing applications and use similar application services. Consequently, developers have to spend a significant amount of time developing different applications for VSNs, resulting in a low efficiency in application development. Thus, in S-Aframe, we adopt a novel type of agents - resident agents as service agents to provide a reusable set of application services to mobile agents. Also, using resident agents to provide application services to mobile agents via agent communications not only enables more flexible operations of the mobile agents, but also provides scalable options to different mobile devices (e.g., on-demand deployment of resident agents and services) as the hardware constraints of them are heterogeneous. Note that the former agent-based solutions in mobile computing [29] usually provide services to mobile agents through a hosting middleware/general agent.

In S-Aframe, based on the programming model, an application developer could develop resident agents, mobile agents, and applications. Resident agents provide application services on mobile devices of VSN systems. Such application services provide local resources for the crowdsensing application to visiting mobile agents. To program resident agents, the programmer can invoke, configure and extend a common set of application services provided by S-Aframe. Mobile agents are created by applications and they can automatically migrate around VSNs with their states and execution results. They can dynamically use different application services provided by resident agents on local nodes to accomplish specific application tasks. An application, as the owner of a mobile agent, can send the mobile agent to execute application services in an underlying VANET and retrieve it back to the mobile node running the crowdsensing application.
4.3.2 Adaptation to dynamic network connectivity

Since the network connectivity of the underlying VANET is dynamic and frequently changes, each mobile node can join or leave the network anytime and anywhere, which may cause mobile crowdsensing application failures. For example, without knowing the failures of targeting nodes when mobile agents execute tasks around VANETs, they may get lost in such networks and fail to bring the expected results back to the users, while the lack of sufficient services and resource in a new mobile node joining a VSN may also impact the correctness of the execution results of mobile agents for crowdsensing applications. Thus, S-Aframe provides self-healing and self-configuring capabilities [23] over dynamic network connections as key framework services to crowdsensing applications. In S-Aframe, for crowdsensing applications to self-heal, it enables the resident agents deployed on every mobile node to continue monitoring the status of a VANET (e.g., which nodes have just become unavailable in a VSN), and provide the information to the mobile agents to help them to adapt to node or link failures. Also, for self-configuration, S-Aframe provides a service that can dynamically and automatically deploy the core functions and services to a new mobile node when it joins a VSN upon VANETs, and if a mobile device does not have the necessary resources or services, mobile agents can also automatically transfer the necessary resources or services between mobile nodes.

4.3.3 Adaptation to users’ dynamic contexts

In a VANET-based VSN system, because of the opportunism of user connections, the changing contexts of the users may also result in users’ dynamic contexts. However, traditionally, the descriptive information of the service requester is compared to that of the service provider, and their similarity is measured using traditional service matching by
simple string-matching (e.g., location based [24], identities based [27]) methods. Such an approach cannot work well as it is not realistic to require service requesters and service providers to use exactly the same contexts (e.g., words about their destinations in a rideshare application) in open and dynamically changing environments of VSNs. Therefore, semantic techniques and context information models are incorporated in CSS as a key framework service of S-Aframe, to enable mobile crowdsensing applications to dynamically and intelligently adapt to different practical VSN scenarios, e.g., matching users’ requirements and accessible services, automatically finding services and information to meet users’ requirements, recommending nearby VSN users with similar destinations/preferences, and so on. More specifically, in order to adapt to users’ dynamic contexts of VSNs, CSS can enable the resident agents of crowdsensing applications to monitor any change in user contexts while the mobile agents are executing the requested service. Upon detecting any changes, CSS can intelligently compose a new service instance that better suits the user’s new contexts and provide it to the mobile agents through the resident agents. In order to adapt to different users, in addition to learning users’ preferences in different contexts from historical information, S-Aframe also allows users to explicitly specify rules for services they prefer in a specific context.

4.3.4 Impact and cost

Considering the limited resource of mobile devices (i.e., computing power, storage capacity, and battery energy), unstable networking connections of VANETs, and time efficiency requirements of most crowdsensing applications in transportation scenarios, the impacts and costs of mobile crowdsensing applications on mobile devices are always a concern. Thus, we adopt weak adaptation [24] as the primary option in S-Aframe, so as to
decrease the resource requirements on mobile devices and ensure that crowdsensing applications developed on S-Aframe have considerable time efficiency in finishing tasks. Also, in order to reduce networking overhead of VSN systems, S-Aframe divides data into two types: shared and non-shared. The shared data contain only the basic context information (e.g., ID name and available services) of the local device, which will be shared with the VSN system by the resident agents. On the other hand, non-shared data contain the agent codes, application specific services, existing data in local mobile devices, and execution results of mobile agents, etc. In addition, when developers develop mobile agents, they can decide whether the mobile agents should store the processing results locally in the mobile devices, and what data mobile agents should carry over to the next devices to which they migrate.

4.3.5 Security and privacy

Security and privacy are common concerns for mobile crowdsensing applications over VANET-based VSNs. Firstly, a malicious user can masquerade as a legitimate user by intercepting and replaying his/her identity to other mobile devices in VSNs. The dynamic security framework proposed in [106] addresses this problem by minimizing the impacts of malicious behaviors in VSNs. Using this framework, a model was designed to derive the probabilities of admitting and trusting a malicious node in VSNs. Furthermore, trust is another important aspect for VSN applications, since it enables entities to cope with uncertainty and un-controllability caused by the free will of others in VSNs. The vehicular network trust model proposed in [107] integrates cryptography-based entity trust, which provides security protections on data origin and integrity, and social trust, which provides a level of belief in the transmitted data. A trust architecture that also models situation-aware trust to address several important privacy issues in VSNs is presented in [108]. In addition,
authentication and access control are vital for security and privacy of VSN applications given the open environment and the possible presence of threats where adversaries can monitor and expose sensitive data of other participants. There are several existing solutions for addressing these issues. For instance, a combined authentication-based multilevel access control to preserve privacy in collaborative environments is proposed in [109], while a lightweight mutual authentication mechanism for network mobility is presented in [110]. In [111], several complementary security mechanisms of public key infrastructure (PKI) for VANETs are presented. Even though the current version of S-Aframe does not incorporate the security mechanisms mentioned above, these mechanisms can be integrated into S-Aframe as a framework service to achieve different levels of security.

Above all, the success criteria of the design of S-Aframe and CSS are summarized in Table 4.2 as following.

Table 4.2: Summary of success criteria of the design of S-Aframe and CSS

<table>
<thead>
<tr>
<th>Category</th>
<th>Success criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>The mobile crowdsensing applications developed on S-Aframe should have high success rate in finishing their tasks under considerable rate of disconnection frequency, e.g., each node’s connection status change every 10s - 30s.</td>
</tr>
<tr>
<td>Effectiveness of CSS</td>
<td>In users’ dynamic context of VSNs, the mobile crowdsensing application developed on S-Aframe and CSS should have the capability to automatically match appropriate services and information with different users. For example, in a rideshare application scenario, a crowdsensing application should be able to return considerable number of appropriate results (i.e., potential drivers) to the users when they are using CSS.</td>
</tr>
<tr>
<td>Time efficiency</td>
<td>The mobile crowdsensing application developed on S-Aframe should have reasonable time efficiency in finishing their tasks upon VANETs, e.g., the migration time of mobile agent between two nodes should be less than 7s (as it normally needs about 7s to setup up the WiFi communication between mobile devices inside vehicles).</td>
</tr>
<tr>
<td>Overhead</td>
<td>The overhead of mobile crowdsensing applications developed on S-Aframe should be reasonable for popular mobile devices, e.g., the RAM usage should be lower than 200MB – 300MB.</td>
</tr>
</tbody>
</table>
4.4 Design and Implementation of S-Aframe

As proposed in Section 4.3, S-Aframe is designed to provide a systematic approach to facilitate the creation and deployment of context-aware mobile crowdsensing applications in VSNs, particularly applications that self-adapt to dynamically changing VANET environments and users’ dynamic contexts during their executions in VSNs. Thus, in this section, we present the key design and corresponding implementation strategies of S-Aframe, and discuss how they meet the requirements of crowdsensing applications in VSNs.

4.4.1 Overall architecture of S-Aframe

We adopt a hierarchical architecture for S-Aframe, so as to provide a light-weight but scalable framework and facilitate agent collaborations and resource reuse for multiple crowdsensing applications on mobile devices. As shown in Figure 4.2, the architecture of S-Aframe has four layers from the bottom up: the framework services layer, resident agent layer, mobile agent layer, and owner application layer.

![Figure 4.2: Architecture of S-Aframe](image-url)
4.4.1.1 Architecture layers

**Framework service layer:** This layer initializes the run-time environments of S-Aframe, extracts the context information from the VSN, and provides the core functions and generic services as framework services to the upper layer. The framework services are only accessed by the resident agent layer but not by other layers. In order to meet the unique requirements of crowdsensing applications in VSNs beyond basic services like time service and ID name service, this layer consists of two core functions as shown in Figure 4.2: network management services, and CSS. They enable crowdsensing applications of VSNs to become context-aware in adapting to dynamic network status and dynamic users’ contexts in VSNs simultaneously. More details about these two functions will be presented in Section 4.4.2 and Section 4.4.3.

**Resident agent layer:** The resident agents provide all application services of S-Aframe on each node to visiting mobile agents. Resident agents can be deployed automatically on-demand to any mobile node that participates in the same VSN. Once resident agents are deployed to a mobile node, they will remain in it to provide various application services simultaneously as long as the node is a part of the VSN. Mobile agents can directly use all the services that the resident agents provide. Application services provided by the resident agent layer can be built on the services provided by the framework service layer. Framework services only implement basic and generic functions and services that are required by most crowdsensing applications of VSNs. Application developers of S-Aframe can develop new application services and deploy them in the resident agent layer. Thus, the services provided by this layer contain two parts: (i) framework services provided by the framework service layer and (ii) application specific services developed by application developers. Framework
services could be reused by every crowdsensing application of VSNs, while application specific services could be reused by a specific type of crowdsensing application on the mobile devices during run-time.

**Mobile agent layer:** The mobile agents run on top of resident agents and are used to execute different application services provided by resident agents. They do not contain any application services in their codes. All the services a mobile agent needs come from the resident agents, such as the services for specific applications, migration service, etc. A mobile agent only contains basic information, such as its migration mode, processing scheme for executing results, as well as computation and communication results. Also, mobile agents are used for transferring necessary resources or application services to some mobile devices when they do not contain such resources or services. Moreover, the data of task execution result gathered from a mobile agent could be shared among multiple crowdsensing applications of VSN in one mobile device.

**Owner application layer:** The applications are owners of mobile agents. An application provides the user interface to its users of crowdsensing who use mobile devices in their vehicles. Through the user interface, the user can select the developed functions he/she prefers, and the owner application of S-Aframe could automatically initiate a mobile agent to execute the services of S-Aframe to accomplish the corresponding function in the VSN system, or release multiple mobile agents to accomplish different crowdsensing tasks simultaneously. S-Aframe supports multiple mobile agents with multiple owner applications working at the same time. In addition, the owner applications can monitor the status of mobile agents they release with the help of resident agents distributed on each node.
4.4.1.2 Architecture implementation

The overall implementation flow of S-Aframe is shown in Figure 4.3. The technical details of S-Aframe, the list of Java APIs, as well as the source codes of a prototype are available in the project of our website [112].

The current version of S-Aframe has been implemented based on the Android system. The execution environment and framework services of S-Aframe are developed by implementing a new layer - mobile S-Aframe AmbientTalk library, on top of the AmbientTalk mobile libraries and class libraries of Android. In this layer, we encapsulate the AmbientTalk virtual machine, and its related basic networking APIs as a new library of Android. On top of
this layer, we developed the S-Aframe Java class library, and designed a mechanism which can invoke the method `evalAndPrint(String script, PrintStream output)` in the original class library of AmbientTalk to exactly map the class attributes between these two layers. Based on the APIs that we implemented, we developed the framework services of S-Aframe, such as the network management services and CSS. Thus, the agent-based applications of S-Aframe developed in Java on Android could automatically invoke the APIs in the mobile S-Aframe AmbientTalk library layer.

Then, there are two main implementation tasks of the resident agent layer: (i) configure the framework services as operations and provide them to the upper layers (mobile agent and owner application layers); (ii) configure the application specific services implemented by application developers using Java, and provide these services to the upper layers as well. The mobile agents and owner applications can then access the Java APIs of S-Aframe through accessing the services provided by the resident agent layer. Finally, based on the above, S-Aframe provides the programming model of resident agent, mobile agent and owner application to application developers, who can use Java to program them according to the specification of the programming model and Java APIs.

4.4.2 Dynamic network environment

As mentioned above, we implemented the *mobile S-Aframe AmbientTalk library* as extended class libraries of Java, which can leverage the advantages of AmbientTalk in our framework, e.g., functions for transparent multi-hop routing in the network layer. Thus, in S-Aframe, we only need to consider two situations when mobile agents are executing the crowdsensing applications while the network environment of VSN changes:

*First situation:* Mobile nodes become disconnected when mobile agents are executing
crowdsensing applications in a VSN system.

Second situation: New mobile nodes join a VSN system when mobile agents are executing crowdsensing applications.

In S-Aframe, we develop a novel network status service in its framework service layer, which makes use of the network APIs provided by AmbientTalk, where every node can listen and determine how many nodes are currently available in the VANET. In addition, we adopt a scheme that enables resident agents to share all the IDs of currently connected nodes, as well as available service lists of mobile devices. By using this service, the framework service layer can inform the upper layer about the real-time networking status of the VSN, such as the current list of node IDs and available services. At the same time, considering the diverse VSN scenarios and specific performance concerns of different crowdsensing applications, S-Aframe does not put any constraint on the application behaviors like specific migration schemes of mobile agents. Instead, to ease the developers’ efforts, S-Aframe provides three generic migration strategies for application developers to develop mobile agents: i) migrating among all available mobile nodes; ii) migrating among a subset of mobile nodes; or iii) only migrating to one mobile node (a special case is that the mobile agent does not migrate to any device but executes the tasks of a crowdsensing application on the local mobile device). Thus, application developers can flexibly select a migration mode, define a migration list and identify primary service requirements (e.g., a specific group of potential drivers who meet the basis requirements in a rideshare application) when they are developing mobile agents to accomplish crowdsensing applications in VSNs. Moreover, developers can design an algorithm for the migration sequences of mobile agents according to their specific situations.
For the first situation, there are two main issues: (a) disconnection of the target mobile nodes to which mobile agents are migrating; and (b) disconnection of the mobile nodes that are currently hosting some visiting mobile agents. In S-Aframe, a mobile agent can obtain the latest ID name list of currently connected nodes from the resident agent when it migrates to a new node. The mobile agent can also store the list of nodes that it has visited before. Thus, the mobile agent can compare its migration list with the latest ID names list and history list every time it migrates to a new node, in order to automatically decide the next appropriate migrating target. In addition, because the migration time of a mobile agent from one node to another is relatively short (normally less than 3s), the probability that the next visiting target node will become disconnected during a mobile agent’s migration is correspondingly low.

Moreover, if a node currently visited by a mobile agent becomes disconnected, it can automatically continue executing the task once the host node is reconnected since in S-Aframe, a mobile agent can store its task history and execution results. Also, the mobile agent’s owner application will wait for some predefined amount of time, after which a new mobile agent will be released to continue execution of the crowdsensing application. Consequently, when nodes become disconnected, mobile agents in a VSN system can self-adapt as they are executing their crowdsensing applications upon VANETs. Also, unlike former solutions for agent-based applications on mobile networks, which specified the behaviors of mobile agents [13], or completely ignored the agent migration patterns [24], S-Aframe not only can handle nodes failures in VANETs for different VSN applications, but also enable developers to flexibly develop different customized crowdsensing applications of VSNs.

On the other hand, network connectivity will change and new service requirements will
arise when new nodes join and leave the VSN system. As mentioned above, mobile agents can obtain the latest network connectivity status through resident agents when they migrate to a new node and automatically decide the destinations. Therefore, it will not impact mobile agents’ migration when new nodes join and leave the VSN system and when network connectivity changes. The main issue for the second situation is the new service requirements. We assume that all nodes have the initial executing environment of S-Aframe when they join the VSN system. Also, we design a new deployment service in S-Aframe, whereby when a new node joins the VSN system, it will automatically broadcast a request to other nodes. Once an existing mobile node of the system receives the message, it can automatically deploy the framework service and resident agent to the new node. Based on the framework service, the new node can obtain the up-to-date status of the VSN system, such as the latest ID name list of the users and latest list of available services as previously mentioned. Therefore, when a mobile agent migrates to a new mobile node that has just joined the VSN system and does not contain the necessary services or resources for the mobile agent’s task, the mobile agent can automatically transfer the services and resources from other nodes to it through the local resident agent, and thus help resident agents to extend application services. Meanwhile, a crowdsensing user of the new node can release a mobile agent to collect the necessary services and resources that it requires. Thus, different from existing mobile agent platforms such as JADE [22] and SPRINGS [23], which adopt a centralized component to track and update the services whenever mobile agents migrate to new nodes, the S-Aframe approach does not even need to be aware of every single agent involved in the VANETs, because in S-Aframe mobile agents can collaborate with resident agents in every node to self-update the services during their migrations.
4.4.3 Context-aware semantic service (CSS)

As shown in Figure 4.2, the design of CSS mainly consists of two parts: semantic-based models, and semantic-based matching algorithm.

4.4.3.1 Semantic-based models

A. Application specific service of S-Aframe and its semantics

An application specific service of S-Aframe is defined as a process unit encapsulating certain business logic and functions, which elucidates what this service does, how it achieves what it does, and how it can be invoked. The traditional service in a service-oriented architecture consists of several information sections: description information, interface information and implementation information. The description information section includes the name, service function, category, and properties of the service. The interface information section specifies a service function via inputs and outputs, and the implementation information contains technical specifications that realize the service function. Each input or output is modeled as a pair of name and data type.

For example, a findPath service with an input parameter “to” with data type “String” is represented as “to:String”. Although the name of the input parameter may imply that it is a destination address, such semantic information cannot be understood automatically by computers or software agents. In S-Aframe, we follow such a standard service model and adopt these key categories to define all types of information available in the application specific services of S-Aframe, so as to standardize its service interactions. However, we also explicitly incorporate a semantic information section to pave the way for semantics understanding and semantic-based service matching of crowdsensing applications in VSNs. An application specific service of S-Aframe is illustrated in Figure 4.4 (a).
The semantic information section adopts an ontology-based conceptual graph to model semantic information of an application specific service with labeled links. An ontology is a specification of a conceptualization that describes the concepts and relationships existing in a domain. A labeled link connects an item in a service and a concept in the ontology with labels equivalentTo or isA to model that the item is equivalent to the concept or is an instance of the concept, respectively. Semantics of service functions, service properties, name, inputs, and other information can be annotated in this way. The modeling of service semantics is illustrated in Figure 4.4(b). For instance, modeling the semantics of the findPath service mentioned previously includes specifying the input “to” as equivalentTo the concept Destination. Another example, 535 Robson Street isA Address, means that the item “535 Robson Street” is an instance of the concept Address. Note that in S-Aframe, elements in a service may be associated with different ontologies and on the other hand, different application specific services may share the same (i.e., common) ontology.

**B. Context information and their semantics**

There exist various kinds of context information originating from multiple information sources at each node on the VSNs, such as social contacts, user history, vehicular onboard
diagnostic (OBD) data and so on. It is hard to enumerate all the context information completely, but it is feasible to classify the core context into several main categories based on the key elements in a VSN: vehicle (Cv), environment (Ce), people (Cp), mobile device (Cm) and network (Cn). Thus, the core context information C can be defined as a set $C = \{Cv, Ce, Cp, Cm, Cn\}$, which can be constructed in the shape of an information tree, as shown in Figure 4.5(a). Different from the notion of context in [82], which refers to the different meanings of a word in different situations, the context information herein is regarded as a node profile, such as a user profile of a mobile device in VSNs. Its semantics are also modeled with one or more ontology-based conceptual graphs. Figure 4.5(b) shows an example of context information in two different nodes linked to a common ontology, from which we can infer that vehicle1 is also a car based on the subclass relationship between sedan and car. Application developers can also follow these specifications to define various new application specific services and contexts on top of S-Aframe.

![Figure 4.5. Semantic-based model of context information.](image)

(a) Context Information Model. (b) Context information with semantic
C. Modeling domain logics and user-specified preferences in logic rules

Given access to context information, CSS models user preferences using logic rules on how to interpret existing context. A user-specified preference in CSS is modeled as a set of conditions and consequences, which specifies that when the conditions are met (e.g., facts like “there is a traffic jam on the street ahead”), its consequences should be taken into consideration (e.g., “do not drive on the street”). Each condition and consequence is modeled as a box consisting of a predicate (e.g., hasTrafficJam) and one or more variables (e.g., X). Each rule has a link labeled implies between the grouped conditions and the consequence. For example, Figure 4.6 shows a rule “if street(X) and hasTrafficJam(X), then notDriveOn(X)”. Such rules are used by CSS with a rule-based inference engine to support the customization of services to individual user profiles and even infer new contexts. CSS also allows application developers or domain experts to represent domain logics in logic rules. For instance, in a carpooling service, if the number of available seats in a car is greater than or equal to one, then the car is available for carpooling (i.e., “if isCar(X), hasVacancy(X,Y), greaterthan(Y,0), then availableCarpooling(X)”).

4.4.3.2 Semantic-based matching algorithm

Semantic-based matching of CSS is realized through determination of the extent of mutual satisfaction with respect to the interpreted user request and major functional information of the service, i.e., service name, service functions, service inputs and outputs,
and service properties. Semantic matching between a user request \( r \) and a service \( s_i \) is calculated as the weighted average similarity:

\[
Cc_s(r, s_i) = \frac{\text{Sim}_{Cc_s}(r, s_i)}{W_n + W_f + N_1 W_l + N_2 W_o + N_3 W_p}
\]  

\[
\text{Sim}_{Cc_s}(r, s_i) = \text{Sim}(r, s_{i\text{name}}) W_n + \text{Sim}(r, s_{i\text{function}}) W_f + \sum_{j=1}^{N_1} \text{Sim}(r, s_{i\text{input},j}) W_l + \\
\sum_{k=1}^{N_2} \text{Sim}(r, s_{i\text{output},k}) W_o + \sum_{t=1}^{N_3} \text{Sim}(r, s_{i\text{property},t}) W_p
\]  

\[
N_1 = \min(N_{r\text{input}}, N_{s_{i\text{input}}})
\]  

\[
N_2 = \min(N_{r\text{output}}, N_{s_{i\text{output}}})
\]  

\[
N_3 = \min(N_{r\text{property}}, N_{s_{i\text{property}}})
\]

where \( \text{Sim}(r, s_i) \) is the semantic similarity between \( r \) and \( s_i \), \( w_n, w_f, w_l, w_o \), and \( w_p \) are the weights for the name, function, input, output, and property, respectively. \( N_{s_{i\text{input}}}, N_{s_{i\text{output}}}, \) and \( N_{s_{i\text{property}}} \) are the number of inputs, outputs, and properties in service \( s_i \), respectively, and \( N_{r\text{input}}, N_{r\text{output}}, N_{r\text{property}} \) are the numbers of inputs, outputs, and properties in \( r \), respectively. In general, we expect that the service name and the service function match well with the user request before we check whether the inputs, outputs and properties match. Thus, two more constraints may be associated with equation (1): \( \text{Sim}(r, s_{i\text{name}}) > \lambda \) and \( \text{Sim}(r, s_{i\text{function}}) > \lambda \), \( \lambda \in [0,1] \).

Given the context model with semantics in CSS, it is easy to find the corresponding concepts for the two items in the common conceptual graph. For example, for a context model that specifies “santana isA Car”, we find the corresponding concept “Car” for the term “santana”. Similarly, we find the corresponding concept for the semantic label “equivalentTo”. Therefore, computing the semantic similarity between an item in the user and an item in the service can be projected to computing semantic similarity of two concepts.
In a conceptual graph of the common ontology, three aspects impact the semantic similarity of two concepts (elements of the graph): the distance or length of the path between the two elements, the depths of the two elements and the depth of their most specific common parent in the common ontology, and whether the direction of the path between the two elements is changed. The semantic similarity is calculated as:

\[
Sim(I_1, I_2) = 2^C(k+p+d) \frac{\text{depth}_{\text{common\_ancestor}}}{\text{depth}_{I_1} + \text{depth}_{I_2}}
\]

(4.6)

where \(\text{common\_ancestor}\) is the most specific common parent item; \(\text{depth}\) represents the depth of an item in the ontology; \(k\) defines the length difference of the two paths between the most specific common parent item and the two items; \(p\) defines the path length between two items; \(d\) defines the changes of the directions over the path; and \(c\) is a constant between 0 and 1.

**Figure 4.7: An example of ontology taxonomy**

Figure 4.7 shows some part of the taxonomy in the common ontology. First, the distance between two elements refers to how many steps it takes to reach from one element to another in an ontology. We can see that the element "Car" goes through four steps to reach the element "Truck". The more steps between two elements, the larger the distance, and the lower the similarity. In this equation, we use the length difference of the two paths between the most specific common parent item and the two items (defined by \(k\)) and the path length between two items (defined by \(p\)) to define the distance. We can show that \(Sim(I_1, I_2)\) decreases when \(k\) and
p increases as c is less than 1. Second, considering the depth of two elements in the ontology satisfies our intuitive understanding of ontology’s classification tree: the classification tree is a progressive refinement work; when two classes are deeper in the classification tree, the relationship between them becomes closer, so that the distance is smaller and the similarity is accordingly higher. For example, the elements "Car" and "Minivan" are closer than the relationship between the element "PassengerVehicle" and "CargoVehicle". Furthermore, two elements have a higher similarity when they are closer to their most specific common parent in the taxonomy tree. Finally, if the direction of the path between two elements changes, the two elements must not belong to the same tree branch and their similarity should be degraded by assigning a lower score. For example, the path direction between the element "Car" and "Minivan" changes, while the direction of the path between "Car" and "Vehicle" does not. We can show that the definition of Sim(I1, I2) satisfies all these features.

We can see that the range for the semantic similarity is [0, 1]. When two items are irrelevant, there is no common parent, then depthcommon_ancestor is 0 and thus their similarity degree is 0. When two items are identical or synonym or equivalent to each other, we have k=p=d=0 and depthcommon_ancestor = depthI1 = depthI2, thus the semantic similarity is 1. Let c be 1/2, the similarity degree of car and minivan is 2*(1/2)^((0+2+1))/3/(4+4)=3/32.

A. Semantic matching mechanism in S-Aframe

In S-Aframe, the resident agent encapsulates the CSS service and provides it to mobile agents and owner applications in the upper layers. New application specific services developed by application developers can be built on top of policies specified by the CSS and communicate with it through the resident agent. When a new mobile device joins the VSN as
a new node, it accepts its user’s service requirements through the user interface of the owner application, and releases a new mobile agent carrying the user requirements and context information. The new mobile agent then navigates the network, calls the CSS through the resident agent on the mobile node it visits, and carries execution results back to its owner. The semantic matching mechanism of CSS in S-Aframe is shown in Figure 4.8, which consists of the semantic interpretation procedure, semantic matching procedure, context monitoring procedure, context interpretation procedure, and rule inference engine.

![CSS mechanism](image)

Figure 4.8: CSS mechanism

Upon receiving a user requirement from the resident agent, the semantic interpretation procedure interprets the user requirement into a semantic conceptual graph if it is a service request, or interprets it into one or more logic rules if it is a user-specified preference. The context monitoring procedure monitors the changes of context in a given VSN through the resident agent and invokes the context interpretation engine to interpret the context information into a semantic conceptual graph upon detecting changes. The rule inference
engine is invoked by the context interpretation procedure and the semantic interpretation procedure, takes the detected context information, user-specified rules, a list of available application services and application rules associated with the services available, and performs rule inference. Only services that match all the logic rules can pass the rule inference engine and become available service candidates for the semantic matching procedure. The semantic matching procedure takes the semantic conceptual graph representing a service request, the semantic conceptual graph representing existing context information, and a list of available application services as well as their associated ontologies, and performs the semantic matching function. The semantic matching procedure then returns one or more application services, each of which is associated with a matching degree in line with the predefined threshold, representing the similarity between the service and the user request. The result is handed to the resident agent, and the mobile agent in turn carries the results back to its owner. CSS in the owner node further chooses the service with the highest similarity matching degree and may automatically execute the service to fulfill the user request.

Compared to other existing service systems over dynamic networks [24, 28, 30], the proposed CSS has the following advantages. First, in terms of the range of context information, CSS proposes a comprehensive context model covering the major parties in a VSN. Second, compared to the semantic representation model in [28], CSS adopts a much more straightforward formula, i.e., labeled links, to establish relationships between services and their semantics, as well as between a context and its semantics, which is more lightweight and scalable for resource-constrained mobile devices. The framework in [30] also applies an ontology-based matching mechanism; however, the similarity measurement may return results contrary to common sense or remain unsolvable in some cases. For example, based on
their measurement, the similarity between two concepts \( Ca \) and \( Cb \) is 0 if they are both direct subclasses of the root concept. Furthermore, to improve interoperability and self-adaptiveness, CSS models not only the semantics of application specific services and context information with labeled links and ontologies, but also the semantics of application domain logics and user-specified preferences information with logic rules. Compared to the framework in \[30\], CSS also allows application developers and end users to further specify rules on how to interpret context information and model application domain logics. Finally, CSS utilizes a semantic matching mechanism on top of ontology reasoning and rule inference to match user requests and services with respect to specific context information so as to improve its adaptability over mobile dynamic networks.

### 4.4.3.3 Implementation of CSS in S-Aframe

The current version of CSS in S-Aframe is implemented in Java and built upon several existing technologies including Jena2 [113] and Drools [114]. The Ontology Base consists of ontology-based conceptual graphs that can be easily described in various formats, such as RDF [115] or OWL [116]. Although S-Aframe assumes that domain experts or application developers design and provide such ontologies, they can turn to numerous existing ontologies, such as WordNet [95]. Our current implementation of semantic and context interpretation procedure employs RDF for ontology representation and Jena2 is integrated for parsing RDF. Since CSS is a framework service and does not interact with application specific services directly, the context monitoring procedure relies on resident agents of crowdsensing applications to monitor the change of context information in VSNs.

Figure 4.9 shows the flow chart of the semantic matching function in the semantic matching procedure. It is an iterative procedure based on similarity computation as it should
go through each item in two sets formed by a service request and application service. The output is a list of services available in the local VSN with corresponding similarity degrees. In addition, in calculating similarity, inference procedures such as retrieving the most specific parent of two items (shown in Algorithm 4.1), an item’s depth in the ontology, the path length between two items, the number of direction changes, and concept projection are implemented. In addition, logic rules in CSS are based on Horn Logic [117] for the inference reasoning. There exist several rule engines for rule inference such as Drools and ALCAS [118]. We integrate Drools, an Object-Oriented Rule Engine for Java in our current implementation.

Figure 4.9: Flow chart of the CSS
ALGORITHM 4.1. Pseudo Code of Finding the Most Specific Parent

```java
public long FindLCA(HierarchicalWordData[] words, out distance, out lcaDepth, out depth1, out depth2)
{
    long LCA = -1; lcaDepth = -1; depth1 = -1; depth2 = -1; distance = MaxValue; i=-1;
    1. while (++i < 1 && LCA == -1) {
        2. IDictionaryEnumerator trackEnum = words[1 - i].Distance.GetEnumerator();
        3. if (trackEnum == null) return -1;
        4. while (trackEnum.MoveNext()){
            5. commonAncestor = trackEnum.Key;
            6. if (words[i].Distance.ContainsKey(commonAncestor))
            7. { dis_1 = words[i].GetDistance (commonAncestor);
                 8. dis_2 = words[1 - i].GetDistance(commonAncestor);
                 9. len = dis_1 + dis_2 - 1;
                 10. if (distance > len)
                     11. { lcaDepth_1 = words[i].GetDepth(commonAncestor);
                         12. lcaDepth_2 = words[1 - i].GetDepth(commonAncestor);
        }
    }
}
```

4.5 Evaluations of S-Aframe

In this section, we evaluate S-Aframe in four aspects that are of particular concerns for crowdsensing users of VSNs in the real-world: (i) reliability of the developed agent-based mobile crowdsensing applications under dynamically changing network connectivity; (ii) time efficiency of tasks completed by mobile agents across multiple mobile devices in ad-hoc mode; (iii) communication and computation overhead, and memory requirements of S-Aframe on mobile devices; and (iv) effectiveness of semantic-based matching of CSS.

In order for S-Aframe to be evaluated under representative VSN scenarios, the experimental setting is as follows: five users are employed, each carrying an Android device equipped with 802.11n WiFi module. The devices include three Android phones and two Android tablets, all of which are running Android version 4.0 or above and S-Aframe (source
code available at [112]). These are common mobile devices that commuters may carry. We use one of the Android devices to act as a WiFi hotspot to which others are connected. As shown in Figure 4.10, each user carries a mobile device denoted as node $M_x$ (where $x$ is the index of each user) and moves in an area of about 150×200 m$^2$. The inter-node distance ranges from 90m to 250m with an average of approximately 150m. These distances are much longer than the recommended safe distance between vehicles in dense traffic [119], but may be realistic under sparse traffic. There are nine predefined positions, and each user may run crowdsensing application created on S-Aframe to initiate mobile agents to accomplish the crowdsensing tasks from others wirelessly when he/she moves from one position to another at normal walking speed. This allows the evaluation of S-Aframe to account for the possible impacts of noise, bad channels, and link failures in real-world environments. Also, we use software configuration methods to let the mobile devices being disconnected/reconnected frequently, so as to simulate the high dynamic feature of VANETs, e.g., vehicles move at high speeds that results in the wireless links of mobile devise on-board vehicles become unreliable and have short lifetimes.

![Figure 4.10: The map area of the experiment](image)
4.5.1 Reliability

As discussed in Section 4.1, the dynamic network connectivity is an inherent challenge for VSN applications. The success rate of crowdsensing task executions in the VSN under such dynamism is a suitable measure of reliability.

Due to the experimental constraints, e.g., it is difficult to test the reliability of S-Aframe in multiple vehicles under the same condition, we evaluate the tasks execution success rate of S-Aframe under simulated conditions of mobile devices being disconnected. In a real vehicular scenario, the speed difference between two vehicles travelling in the same direction normally will not exceed 60km/h (16.67m/s), and the sum of the speed of two vehicles in opposite directions normally will not exceed 180km/h (50m/s). Given that the current popular mobile devices equipped with 802.11n WiFi module normally have an ideal WiFi communication range up to 500 meters at a data rate of 15.5 Mbps or higher when there is no obstacle [120], the duration of connectivity between two mobile devices in two different encountering vehicles will be in the range of 10s (500/50) to 30s (500/16.67) or even longer if the vehicles are traveling in the same direction with no excessive speed differential. Therefore, in this experiment, we configure the simulated conditions of mobile devices being disconnected in constant time intervals (T) of 30s, 20s, and 10s, or in exponentially distributed time with rate parameter $E = 1/30s$, $1/20s$, and $1/10s$ correspondingly. We use two test cases to evaluate this. In the first case, all users initiate a mobile agent from their mobile devices simultaneously to finish the same crowdsensing task (collect the location information of the other four mobile devices). In the second case, all users initiate two mobile agents (one for each task) from their mobile devices simultaneously to finish two different tasks (one is to collect the location information and another is to collect
the meta-data of a specific photo from the other four mobile devices). Each case is repeated four times, from which the overall average results are calculated. In addition, during the experiments, we observed that beyond the effect of disconnections by software configuration mentioned above, mobile agents failed to finish their tasks on some occasions due to noise or bad channel conditions, and we also took these cases into account in the results presented.

The results of the first case are shown in Figure 4.11. We found that S-Aframe achieves similar success rates under a fixed or random time between disconnections. With disconnection in constant time, the success rates are comparable for the cases of $T = 30s$ and $T = 20s$. In addition, when $T = 10s$, the success rates degrade faster than those under the two lower disconnection frequencies. With exponentially distributed times between disconnections, the success rates are similar when $E = 1/30s$ and $E = 1/20s$, and degrade faster when $E = 1/10s$. As discussed above, since in real vehicular scenarios, the durations of connectivity between two mobile devices in two encountering vehicles are normally more than 10s, the probability that they get disconnected within 10s is relatively low. Therefore, the crowdsensing applications created on S-Aframe can exhibit a high reliability in their task execution in vehicular environments under the considered rates of disconnection frequencies.

![Figure 4.11: Tasks execution success rate-first case](image-url)

(a) Constant time (b) Exponentially distributed time
The results of the second case are shown in Figure 4.12. Compared to the first case, under the same disconnection frequencies, the success rates of the second case have degraded a little. It is mainly because in the current version of S-Aframe, its communication is based on AmbientTalk, which is still an experimental language, and its use of multi-cast heartbeats (e.g., UDP packets) to detect disconnected peers may not be very reliable. Thus, the success rates of the S-Aframe tasks may degrade when the communication environments become more complicated. However, the worst success rate in the second case is still more than 50% in the atypical condition when T = 10s or E = 1/10s.

![Graphs](image)

(a) Constant time (b) Exponentially distributed time

Figure 4.12: Tasks execution success rate-second case

### 4.5.2 Task execution latency

Similar to the experimental setting in Section 4.5.1, we use four test cases to evaluate the execution time efficiency of S-Aframe under the condition of one or multiple owner applications with one or multiple mobile agents in each Android device, which does not go offline in this set of experiments. In the first case, one owner application releases one or two mobile agents to finish the related crowdsensing tasks by using two different types of services: a) original framework services incorporated in S-Aframe; or b) application specific services extended by developers. In the second case, two owner applications release two mobile agents to finish the
same tasks by using the same services simultaneously. In the third case, one owner application releases a mobile agent to finish a task by using an application specific service, and another owner application releases a mobile agent to finish a task by using only the framework service. In the fourth case, two owner applications each releases two different mobile agents to finish two different tasks at the same time.

A summary of all test scenarios in this experiment is shown in Table 4.3. Although S-Aframe supports one mobile agent invoking multiple application services of S-Aframe to accomplish a task, in this experiment, we choose to evaluate under the condition that one owner application in each Android device launches one or two mobile agents, with each agent corresponding to a specific task and finishing the task by invoking one service, so as to better demonstrate the impact of the number of mobile agents and type of application service on the task execution latency of S-Aframe. Moreover, when each owner application has two mobile agents working independently at the same time, we only provide the total time for all mobile agents to finish their tasks in all target nodes, as the task completion times of individual mobile agents are quite close. In addition, we can monitor the status of mobile agents through the user interface of S-Aframe as shown in Figure 4.13. The results in Figure

<table>
<thead>
<tr>
<th>Category</th>
<th>Numbers</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case 1</strong></td>
<td>Owner applications: 1&lt;br&gt;Types of mobile agents: 1/2&lt;br&gt;Mobile agents: 1/2</td>
<td>To assess the impact of different type of services on task execution latency, and the relative impact between type of services on task execution latency when invoking them concurrently</td>
</tr>
<tr>
<td><strong>Case 2</strong></td>
<td>Owner applications: 2&lt;br&gt;Types of mobile agents: 1&lt;br&gt;Mobile agents: 2</td>
<td>To assess the impact of owner applications on task execution latency when mobile agents invoking the same type of services</td>
</tr>
<tr>
<td><strong>Case 3</strong></td>
<td>Owner applications: 2&lt;br&gt;Types of mobile agents: 2&lt;br&gt;Mobile agents: 2</td>
<td>To assess the impact of owner applications on task execution latency when mobile agents invoking different type of services</td>
</tr>
<tr>
<td><strong>Case 4</strong></td>
<td>Owner applications: 2&lt;br&gt;Types of mobile agents: 2&lt;br&gt;Mobile agents: 4</td>
<td>To further assess the impacts on task execution latency when the above three cases are co-existing</td>
</tr>
</tbody>
</table>
4.14-4.17 are shown with their 92.5% confidence intervals calculated over the successful rounds (the error margins are difficult to see because they are so small, which are only about ±250.69ms for all cases).

From Figure 4.14, it can be observed that the task execution time latency using the framework service and application specific service are comparable. The reason is that after the application specific service is extended by the application developer, it becomes a component of the resident agent layer, and thus mobile agents can directly invoke this service. Meanwhile, as the situations in all mobile nodes are similar, the execution time increases linearly with the number of target nodes. In addition, we find that the execution time of one owner application with two tasks working independently at the same time is similar to that of two tasks working independently at different times. This is because even when two different tasks are working simultaneously, they are in different threads, and thus they do not affect the efficiency of each other.
From the results shown in Figure 4.15 we find that if two owner applications release multiple mobile agents to execute the same task independently at the same time, the task execution time latency is less than the case when there is only one owner application, in particular when the number of targets increases. Through monitoring the status of mobile agents (as shown in Figure 4.13), we find that the execution time contains two main parts: migration time (about 2.74s from one mobile device to another) of mobile agents; and processing time (about 4s) for a mobile agent to finish its task in a new mobile device, which consists of initialization time (<1s), and the time for processing the task in parallel with fetching the latest information about the network status for migration (about 3s). In the case of multiple owner applications with one or multiple mobile agents, when an earlier agent finishes the task and shares the results with mobile agents that arrive later, time is saved from having to duplicate the tasks. Moreover, from Figure 4.16, we find that the situation discussed above will not happen if two owner applications release different mobile agents to finish different tasks simultaneously. Therefore, in this case, their execution time latency is found to be similar to that in the case of only one owner application, since they cannot reduce
the processing time of tasks.

Figure 4.15: Time efficiency (same applications) of multiple owner application

Figure 4.16: Time efficiency (different applications) of multiple owner applications

Figure 4.17: Time efficiency of multiple owners with multiple mobile agents
On the other hand, as shown in Figure 4.17, with multiple owner applications having multiple mobile agents working in multiple related crowdsensing tasks simultaneously, the time efficiency of a single task is usually improved. This implies that the task execution latency of S-Aframe should not increase even if the communication environment becomes more complicated. In our practical experiment, the internode distance is about 150m, and the time for a mobile agent to migrate from one node to another is found to be about 2.74s, or at a speed of 197km/h, which is fast enough to support tasks in many mobile crowdsensing applications in VSN scenarios, including emergency tasks in traffic accident scenarios.

In addition, as with Agilla [24], we adopted the micro-benchmarks and Java virtual machine (JVM) to perform on a network with 25 nodes, to test the migration of S-Aframe under simulations. We observe that the average one-hop latency, averaged over 100 simulations, is about 146ms, which is 30.48% lower than that of Agilla (about 210ms) under the same setting. Even though S-Aframe and Agilla are implemented on different platforms, a key reason for the better latency performance of S-Aframe is that it uses resident agents to communicate with mobile agents, unlike Agilla which relies on remote shared memory (tuple space) access. This can simplify the operation of the mobile agent on each node and hence decrease the time latency.

4.5.3 Overhead

As the communication overhead of each Android device is similar under the same condition, we compute only the average communication overhead (by invoking the network APIs provided by Android) under the condition that each Android device initiates one mobile agent, while other conditions are the same as the experimental setting in Section 4.5.2. As shown in Figure 4.18, the communication overhead consists of two parts: foreground and
background, where the latter is mainly due to the resident agent and related networking services of S-Aframe and the former is due to the mobile agent when it is visiting different devices in the underlying VANET.

As discussed in Section 4.5.1, the resident agents need to transmit multicast heartbeats at regular intervals to detect any disconnection from peer nodes, thus giving rise to uniform background traffic. This is supported by our observation that the background overhead increases linearly with the number of online nodes, each transmitting 20KB every 5 minutes. Also, since the uniform background traffic is due to the design of S-Aframe and independent of the application, no matter which kind of crowdsensing applications are developed on S-Aframe, the background traffic characteristics should remain similar. On the other hand, the foreground overhead depends on the migration rate (or number of nodes visited per unit time) of a mobile agent. Also, we can foresee that in real-world scenarios of VSNs, since the execution times of mobile agents depend on different purposes of specific crowdsensing applications, which usually follow the exponential distribution, the migration rates of mobile agents also follow the same distribution accordingly. Thus, the foreground traffic of S-Aframe can be modeled by a Poisson process. Approximately 12KB is incurred for every
move between two nodes for one mobile agent.

Moreover, we monitored the battery consumption and memory requirement of an Android device (Google Nexus 10 with 9000mAh battery, 2GB memory capacity) running S-Aframe in two scenarios: one in which the Android device with S-Aframe provides application services to visiting mobile agents but do not release any mobile agent; another in which the Android device with S-Aframe constantly releases mobile agents to execute the application tasks. Each of the two test scenarios lasted 30 minutes, and was repeated 10 times. It was found that the average battery consumption and memory requirements due to running S-Aframe were relatively low, consuming only 2% of the battery capacity and 17.2MB of memory in the first scenario, and 2.6% of battery capacity and 19.6MB memory in the second scenario. Also, we compare the battery consumption and memory requirements between S-Aframe and AmbientTalk in the first scenario. We observe that the battery consumption due to running AmbientTalk is comparable to that with S-Aframe (~2%), while its memory requirement is 15.6MB or ~10% lower than S-Aframe. This is mainly because S-Aframe needs to run more basic services and multiple resident agents while AmbientTalk does not. However, considering that most mobile devices have 1GB or more memory and are recharged daily by their users at home, workplace, or in vehicle, the battery consumption and memory requirement should not limit the adoption of S-Aframe in most VSN application scenarios.

Furthermore, as the set of performance results from the experiments presented in this section was limited to five users in a specific environment, these results by no means provide a thorough validation of S-Aframe under all possible conditions, and therefore performance comparisons with other situations should be made with caution. For example, increasing the
number and density of users, obstacles and weather conditions in other VSN scenarios may also significantly impact the reliability and task execution latency of S-Aframe. The main intention here is to investigate the feasibility of S-Aframe under simple but common scenarios, from which better designs can be realized and applied to different VSNs scenarios for real-world deployment in the future. For instance, we found that from the background traffic studies, while increasing the frequency of multicast heartbeats to detect disconnected peers may increase the task reliability of S-Aframe, it may also significantly increase the cost (i.e., network overhead and tasks execution latency) of crowdsensing applications of VSNs. Thus, we could investigate a scheme to estimate the disconnection frequency of VANETs, and consider it jointly with the specific application requirements (i.e., high requirements for reliability or not) for dynamic switching of the frequency of multicast heartbeats in mobile devices during run-time, so as to achieve an appropriate trade-off between the task reliability of S-Aframe and the cost of crowdsensing applications in different VSN scenarios.

4.5.4 Effectiveness of CSS

Due to the experimental constraints, it is difficult to find a large number of users and devices for a real-world evaluation of the semantic matching performance of CSS. Thus, we instead evaluate the results of simulating VSNs containing different numbers of nodes ranging from 50 to 1000, distributed randomly over an area of 150×200 m², in which each node moves at a predetermined speed and can automatically broadcast a request to other nodes in the network. The faster the nodes move, the more dynamic the network is. Each node in the network provides a list of application services (e.g., returning the information of its vehicle), and has its own context information such as its vehicle information and current
position that is updated from time to time. All vehicle information in the context for each node is automatically generated by exemplifying from the Onto_Vehicle ontology. Each node will execute the semantic-based similarity measure matching through CSS, independently from other nodes, when it receives the request broadcasted by the source node. A mobile agent is then released by the source node to access the network. Once the mobile agent reaches a node in the network, it collects the query results.

![Figure 4.19: Effectiveness of the semantic-based matching](image)

To evaluate the effectiveness of the semantic-based similarity measure, we compare two different matching approaches in the process of finding nodes with similar vehicles: our semantic-based similarity measure matching and keyword matching. In the latter approach, when a mobile agent reaches a node, it recalls the service on the node to return the vehicle information in its context, compares the information with the exact word(s) in the service request that it carries, and determines whether the two have common words or are even identical. It does not take into account the specific meaning of the request and the information from the node. In our semantic-based similarity measure matching approach, the semantic meaning of the vehicle information is acquired by projecting the information
onto the ontology, and the similarity between vehicle information of two nodes is returned by recalling the CSS service. We set 0.95 as the value of parameter $c$ in Equation (4.6) and chose two similarity thresholds: 0.75 and 0.5, for the evaluation.

As shown in Figure 4.19, the results confirm that our approach with a Semantic-High threshold (=0.75) returns more matched results than keyword-based approach as adopted by schemes such as Agilla [24]. Also, the semantic-based matching algorithm in our CSS takes into account the meaning of the request and measures the similarity on the semantic level. As a consequence, even if the service requested is quite different from that provided on the syntax level, they may still be closely related semantically. This important feature can be applied to realize intelligent service discovery in various crowdsensing applications of VSNs.

Figure 4.19 also shows that there are more results with the Semantic-Low threshold (=0.5), compared to the higher threshold. That is, with other settings being the same, the lower the threshold is, the more results CSS returns. When the priority in some application scenarios of VSNs is to find a match but the current threshold returns none, such a feature would be important as it guides us to lower the criteria to get more returned results.

In addition, the alternative approach MobiSN [30] is implemented on J2ME but not Android platform, and MobiSN is only designed for the semantic similarity computing in the creation of social grouping in MSNs and not opens APIs to developers for other purpose. Thus, it makes direct and fair comparison between S-Aframe and MobiSN at times difficult if not impossible, and we do a qualitative comparison between them in the application example presented in next section.
4.6 Application Example

For a rideshare application deployed over Inter-VANETs linked by Internet connections [121], its major tasks are to automatically, intelligently and efficiently help the users to search for the appropriate people (drivers or passengers) who are available to them, and provide them most relevant context information (i.e., price for this rideshare, and the planned destination of individual user). Finally, based on the context information, this application should help the users figure out the most suitable driver/passenger for them in an easy and convenient manner. However, as the users who involved in a rideshare application are usually opportunistic and undefined, the common collaborative applications which assume the participants from specific/named groups may not work. Also, as the users’ surrounding contexts are changing in transportation environment, hence a rideshare application needs to provide each user multiple potential candidates with latest information. In addition, in case the signal of cellular networks are not good for Internet access, a rideshare application also should be able to work over VANETs, so as to make sure the latest rideshare information can be updated to each user in time. Consequently, based on S-Aframe, we develop *smart ride*, a novel crowdsensing application to demonstrate the usefulness of S-Aframe and CSS in rideshare as mentioned above.

For this application, we first developed a resident agent to provide all the necessary application services at a node to a visiting mobile agent. The resident agent invokes the CSS, which semantically matches the requirements carried by a mobile agent released from a service requestor with the locally available services. If suitable services are found, the mobile agent can return the correct context information and execution results to its owner. Also, beyond generic services provided by S-Aframe, some services like position and map services
which can invoke the GPS sensor were developed and deployed as application specific services. Second, we developed the mobile agent and its owner application for this application. The task of the mobile agent is to automatically collect the context information according to the requirements of the users in the underlying VANET by sequentially visiting each one of them. The owner application creates and sends mobile agents to execute the task in the underlying VANET while provides an interface to the user.

![Smart Ride User Interface](image)

**Figure 4.20: Screen-shots of the smart ride user interface**

The user interfaces of smart ride are shown in Figure 4.20. For demonstration, two drivers (driving 2005 Toyota Sienna vehicles) and two potential passengers, each carrying a Google Nexus 4 in which the *smart ride* application is installed, were deployed to form a VSN. In addition, we observed that the average time to form a VSN among these four nodes was about 8s through a WiFi hotspot. One of the potential passengers named *Shirley* used her Nexus 4 to register with the VSN system and searched for the best-matching driver for ridesharing. Once registered as shown in Figure 4.20 (a), Shirley input her requirements for
ridesharing via the user interface. This resulted in smart ride automatically releasing a specific mobile agent to execute this service in the VSN. As mentioned earlier, upon visited by a mobile agent, the resident agent can provide it with the necessary application services and CSS. Thus, the mobile agent automatically and intelligently migrated between these phones, executed its task to discover the appropriate drivers, and returned the results to Shirley. The average time that Shirley obtains the results is about 18s. In this demonstration, the two drivers were driving at a speed of about 30km/h, and the network was relatively stable. Thus the mobile agent could finish the task with a high success rate (it only failed once in ten repetitions), and the time latency of the task was mainly due to mobile agent’s migration and running CSS but not network status update. The migration totally took 12s and running CSS required about 5.1s (1.7s on each node which consists of 1s initial time and 0.7s process time). Furthermore, we found that semantic matching used about 53% of the time (0.37s) running semantic matching CSS in each node, and the remaining time on semantic interpretation, context interpretation, and rule inference. Since the time spent on semantic matching includes several parts such as retrieving the most specific parent of two concepts, and a concept’s depth in the ontology, thus the time varies with the size of ontology used for computing semantic similarity of a pair of service items. The current experimental results suggest that, in user applications, small ontologies are preferred to huge ontologies.

Finally, Figure 4.20(b) shows the results displayed on Shirley’s phone, which reports that two drivers (Victor and Terry) are found to match the requirements of her request. Figure 4.20(c) shows the result displayed on Victor’s Nexus 4. It may automatically accept Shirley’s request since it has a high degree of matching. Figure 4.20(c) also shows the option for ridesharing drivers to similarly post information about their trips and rideshare requirements.
In addition, the ridesharing driver could select the “Auto” mode, which will automatically post his rideshare information in a specific location/time when he is driving (similar to opportunistic sensing).

We further illustrate the matching procedure between a service requester (Shirley) and a service provider (Victor). First, several logic rules are defined for the application, including the two rules given in Section 4.4.3. There are several ontologies necessary for this example, including the domain ontology for service function, which provides different descriptions of the service function in the domain of rideshare and their relationships, as shown in Figure 4.21.

![Domain ontology of service function descriptions for rideshare](image)

**Figure 4.21: Domain ontology of service function descriptions for rideshare**

We reuse the ontology shown in Figure 4.7 for the vehicle ontology (*Onto_Vehicle*) and the WordNet ontology as the source of the common ontology for items to be matched. Another domain ontology is the road structure ontology (*Onto_Road*), a fragment of which is shown in Figure 4.22.

![Fragment of the road information ontology](image)

**Figure 4.22: Fragment of the road information ontology**
The context information such as the vehicle information which could be derived from
the OBD sensor embedded in vehicles, users’ personal information, and social information
for rideshare is shown in the Figure 4.23.

Next, we exemplify the service request from Shirley. Figure 4.24 shows a fragment of
the rideshare requesting service carried by a mobile agent from Shirley’s mobile phone. CSS
infers that Pacific Center is an instance of Robson Street.

Figure 4.25 shows a service fragment from service provider Victor. The attribute value
“3” indicates that Victor has three seats available in his Limo. Thus CSS derives from the
rule inference engine that Victor can provide the rideshare service. Furthermore, CSS infers
that Limo is an instance of Sedan and his destination is an instance of Robson Street.
Then CSS executes semantic matching and infers that the similarity between Robson Street and Robson Street is 1 and the similarity between Sedan and PassengerVehicle is 3/64. Furthermore, the distance matching degree between a service requestor and a service provider is a decreasing function of the distance between them, for example, \( f(x) = \cos(x \pi / 2D) \), where \( D \) is a preset parameter defining the maximum distance that the service provider can be considered as a candidate. Table 4.4 summarizes the similarity score for each matching item and the overall result at Time 1, which shows a 0.81 similarity matching between Shirley’s requirements and Victor’s context information, a 0.65 between Shirley and Terry and thus the service provided by Victor is returned. If keyword-based method is used in this example, Victor cannot provide rideshare service to Shirley as Victor’s service descriptions (e.g., destination) do not match Shirley’s requirements. Our solution suggests two potential service providers, Victor and Terry and thus achieves a higher recall rate. Moreover, because the participants in the example are moving all the time, at Time 2, the mobile agent detects that Terry is a better choice than Victor. The semantic based method in MobiSN [30] does not consider any restriction specified by the context information and may return a driver whose vehicle has already reached its full capacity as the service provider. Comparatively, our solution offers a higher accuracy as it further filters service providers through inferences with logic rules.

<table>
<thead>
<tr>
<th>Matching Items</th>
<th>Weight</th>
<th>Similarity (Shirley, Victor)</th>
<th>Similarity (Shirley, Terry)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time 1</td>
<td>Time 2</td>
</tr>
<tr>
<td>Service Function</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>VehicleType</td>
<td>0.15</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Gender</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Distance</td>
<td></td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>Distance Matching</td>
<td>0.15</td>
<td>0.7</td>
<td>0.38</td>
</tr>
<tr>
<td>Destination</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Overall</td>
<td>1</td>
<td>0.81</td>
<td>0.76</td>
</tr>
</tbody>
</table>
4.7 Summary

In this chapter, we have presented S-Aframe, an agent based multi-layer framework with CSS that is able to utilize context information and provide a high level software platform for VANET-based mobile crowdsensing applications. S-Aframe hides the complexity of dealing with changing network connectivity and varying user services requirements in VSNs, by providing a programming model with extensibility support. S-Aframe also provides a rich set of framework services to support application development using Java with standard API format. Therefore, S-Aframe provides a modeling paradigm which can facilitate the creation and deployment of different VANET-based mobile crowdsensing applications for VSNs. Furthermore, S-Aframe supports dynamic agent collaborations and service matching during run-time of mobile devices. Thus, more autonomous, intelligent and adaptive crowdsensing applications can be developed for the dynamically changing environments of VSNs.

To the best of our knowledge, S-Aframe is the first mobile platform that supports the creation and deployment of context-aware mobile crowdsensing applications for VANET-based VSNs. We have extensively evaluated S-Aframe through a set of practical experiments on Android devices, which showed that the crowdsensing applications created on S-Aframe can perform well under dynamically changing connectivity, and achieve good time efficiency with low computation and communication overhead. Furthermore, we have designed and implemented a ridesharing mobile crowdsensing application based on S-Aframe to demonstrate its practical feasibility for real-world VSNs.
Chapter 5: A Crowdsensing-based Situation-aware Music Recommendation Application for Drivers

5.1 Introduction

According to the statistics published by the World Health Organization (WHO) [2], 1.2 million people die and 50 million people are injured or disabled on roads every year. Statistics in Europe show that approximately 10-20% of all traffic accidents are due to diminished vigilance of drivers, e.g., fatigue or negative emotion. Previous research has demonstrated that not only does listening to suitable music while driving not impair driving performance, but it could lead to an improved mood and a more relaxed body state, e.g., decreasing the high breath rates of drivers when they are fatigued [122].

However, to the best of our knowledge, a mobile music recommendation application that specifically designed for drivers is not yet available. Thus, we design and develop SAfeDJ, a novel crowdsensing-based situation-aware music recommendation application for drivers. The development of SAfeDJ aims to further demonstrate how the solutions in our platform introduced in previous chapters support the creation of context-aware mobile crowdsensing applications, and facilitate the realization of such applications in real-world deployment in VSNs.

The rest of the chapter is organized as follows. Section 5.2 reviews the background of related techniques and concepts of SAfeDJ. Section 5.3 presents the key designs of SAfeDJ with implementation strategies, and discusses how the solutions designed in our crowdsensing platform support the development of SAfeDJ. Section 5.4 demonstrates the mobile application of SAfeDJ, and evaluates the performance of SAfeDJ through a set of practical experiments and user studies. Section 5.5 summarizes this chapter.
5.2 Background and Related Work

In general, the current mobile music recommendation applications mainly follow three approaches [123]: (i) Collaborative filtering (CF); (ii) Content-based; and (iii) Hybrid. CF is based on the assumption that if two users have similar music preferences, then the songs preferred by one user will also be recommended to the other. A problem with the current CF approach is that it could not recommend new songs to users if such songs have not previously received any comments from any user [124]. In contrast, the content-based approach recommends songs to user by calculating the similarity between songs. For example, if a user likes a particular song, then other similar songs are recommended to the user. However, driving situations may also have significant impacts on the choice of preferable songs, e.g., a song may be preferable to a user when he is driving on an open road but not when he is stuck in a traffic congestion. Also, both the current CF and content-based, and even the hybrid approach (which combines CF and content-based) would be difficult to predict and recommend songs to individual drivers when they encounter new driving situations.

Based on our crowdsensing platform, we develop SAfeDJ, a novel crowdsensing-based situation-aware music recommendation application for drivers. Different from the hybrid approach, SAfeDJ further leverages the computing and sensing capacity (i.e., front camera) of smartphone and the sensors like OBD in cars to automatically analyze the impact of music to drivers in different situations. Also, similar to opportunistic sensing, SAfeDJ enables automatic sharing of such data (i.e., the drivers’ mood-fatigue changing history with information of recommended song and location) in an interactive manner. Furthermore, SAfeDJ uses the ASCM and CSS in our crowdsensing platform to orchestrate the interactive data from different participants in crowdsensing, so as to achieve intelligent
recommendations of preferable music to individual drivers in their new driving situations. For instance, SAfeDJ recommends music to drivers considering both their listening behaviors and current driving situations.

In addition, cloud computing is emerging as the dominant computing environment to enable the sharing of resources such as infrastructures, platforms, and value-added services and applications [125]. Today, linking traditional mobile services to the cloud is becoming increasingly important, as doing so can scale up information processing capabilities and achieve striking results with high efficiency through parallel and distributed computing techniques [126]. Thus, the cloud side of SAfeDJ is built on the Vita cloud platform we presented in Chapter 3 mainly for two purposes: (i) working with the smartphones to process and identify the attributes of new songs and their corresponding music-moods, to achieve higher efficiency of music recommendation and decrease the battery consumption of smartphones, and (ii) working as a central coordinating platform to aggregate and store the sensing data from diverse drivers, to facilitate the orchestration of such data from different participants in crowdsensing, and hence improving the intelligence of the crowdsensing-based music recommendation for individual drivers.

5.3 Overall Design of SAfeDJ

The system architecture of SAfeDJ is shown in Figure 5.1, in which we can observe the structure and key components of SAfeDJ. Correspondingly, Figure 5.2 shows the overall workflow of SAfeDJ. In general, SAfeDJ mainly has two processes: (i). Music-mood mapping (MM) and context-music mood mapping (CMM); (ii). Music Recommendation and satisfaction calculation of songs.

The MM methods are designed to process and identify the attributes of songs and their
corresponding music-moods (which consists of six dimensions of moods as [lively, energetic, sentimental, groovy, noise, peaceful], or simply, [L,E,S,G,N,P]) for smartphones in an effective manner, more details about it could be found from the Appendix A. As discussed at the beginning of this chapter, SAfeDJ is developed to further demonstrate how the solutions in our platform introduced in previous chapters support the creation of context-aware mobile crowdsensing applications. Thus, in this part, we focus on the introduction of the CMM, and explaining how the implementation of it for situation-aware music recommendation is supported by our crowdsensing platform.

Figure 5.1: System architecture of SAfeDJ
5.3.1 Context model

To implement the **CMM**, we first need to design a context model for smartphones to automatically collect and contextualize data about drivers from multiple sensors in real-time. This model is shown by the green boxes in Figure 5.1 as input data for each car (or driver). Referring ASCM presented in Chapter 3, we first define three major categories: **social context (SC)**, **traffic context (TC)**, and **human mood-fatigue context (HMC)**, in which SC and TC are corresponding to string attributes and HMC is corresponding to continue attribute in ASCM. The above contexts can be obtained from different kinds of sensing data, and then used as key contexts to support the predictions of drivers’ music preferences in different driving situations while driving as presented in **CMM** later.

**SC**: From our online surveys [127], we observe that the probabilities of two drivers favor the similar style of music in the same driving context are higher when their SC (i.e., gender) are closer. Thus, in our current design of SAfeDJ, the data about SC is obtained from the user profiles of Facebook, which includes the cultural background, gender, age, and driving experience of a driver. Also, it includes the interaction history of a driver with friends in his social circle. In addition, we plan to integrate the data from the professional social networking music websites like SoundCloud and Deezer, so as to further analyze the users’
music interest pattern in the future evolution of SAfeDJ.

**TC:** It expresses information about the traffic condition that a driver is situated in. This includes the location, weather, road surface condition, traffic congestion, etc. The location data can be obtained automatically by a smartphone using its onboard GPS receiver, while the weather and traffic related information can be obtained from the Internet accordingly. The data about TC can also be acquired by aggregating traffic condition data reported by individual drivers.

Figure 5.3: Structure of the data used in the driver’s mood-fatigue detector

**HMC:** We design a mood-fatigue detector in HMC, which can collect and analyze multiple sensing data on a smartphone to infer the real-time mood-fatigue degrees of the driver. As shown in Figure 5.3, the mood-fatigue detector mainly considers the data: (i) engine rpm (RPM), throttle input (Tth) and driving speed (Sp) from OBD-II; (ii) heart rate (Hr) and in-vehicle temperature (T) from wearable sensors; (iii) facial expression and blinking frequency of the driver’s eye-lid (EI) from the front camera of smartphones; and (iv) road congestion situation from online traffic service (e.g., Google map). Based on the above sensing data, we develop a process to analyze the mood-fatigue degrees in the detector (more technical details and evaluations about this detector could be found from Appendix B). In this process, the status of a driver’s mood are classified into six dimensions: aggressiveness, patience, neutral, happiness, sadness, and anger, and then the degrees of the six dimensional moods and fatigue are inferred and quantified as float data (from 0 to 1.0).
Finally, after all the data about SC, TC and HMC have been collected and contextualized, we design a mechanism that enables the smartphone of each driver to automatically synchronize such contexts to the crowd history dataset (see Figure 5.1) in the cloud side of SAfeDJ 5s before the end of each song being played. An example about the user history table (UHT) of such dataset is shown in Table 5.1. In addition, since the cloud side of SAfeDJ is built on the Vita cloud platform presented in Chapter 3, thus the communications between mobile devices and SAfeDJ cloud is based on HTTP protocol as well.

Table 5.1: User history table (UHT) in the crowd history dataset

<table>
<thead>
<tr>
<th>Hist._id</th>
<th>user_id</th>
<th>fatigue_deg.</th>
<th>aggressive_deg.</th>
<th>neutral_deg.</th>
<th>happy_deg.</th>
<th>sad_deg.</th>
<th>angry_deg.</th>
<th>patience_deg.</th>
<th>longitude</th>
<th>latitude</th>
<th>music_scores-S,L,E,S,G,N,P</th>
<th>time_stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23673</td>
<td>0.8</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.6</td>
<td>122.085</td>
<td>37.42</td>
<td>[0.71,0,0,1,0,0,0,0,0,0]</td>
<td>16:31:3</td>
</tr>
<tr>
<td>2</td>
<td>14655</td>
<td>0.2</td>
<td>0.8</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>120.666</td>
<td>38.52</td>
<td>[1,0,1,0,0,0,0,0,0,0]</td>
<td>16:32:3</td>
</tr>
<tr>
<td>3</td>
<td>17887</td>
<td>0.6</td>
<td>0.6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
<td>123.785</td>
<td>37.12</td>
<td>[0,0,1,0,0,0,0,0,0,0]</td>
<td>16:33:3</td>
</tr>
<tr>
<td>4</td>
<td>23673</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
<td>0.9</td>
<td>0</td>
<td>0.7</td>
<td>0.7</td>
<td>122.133</td>
<td>37.33</td>
<td>[1,0,2,0,1,-0.2,0,0,0,0]</td>
<td>16:35:3</td>
</tr>
</tbody>
</table>

5.3.2 Context-music mood mapping (CMM) and music recommendation

There are many different possible types of human moods and traffic conditions in a driver’s daily life. Therefore it would be difficult for individual smartphones to predict and recommend suitable music to a driver in new driving situations that have not been encountered before. For instance, there may be records that some songs were preferable to a driver driving in mountains with a happy mood, while there is no music record corresponding to situations when this driver is driving in similar mountains but with a sad mood, or driving on bumpy roads for the first time but with a similar happy mood. In Subsection 5.3.1, we have presented the context model designed for smartphones, which enables an in-car
smartphone to automatically collect and contextualize the data of an individual driver's SC, TC and HMC, and synchronize the processed data to the cloud side of SAFeDJ in real-time.

In particular, the HMC is derived from multiple sensors (i.e., front camera of a smartphone, OBD, wearable sensor). However, unlike participants of common collaborative applications which are from a specific/named group, the users of SAFeDJ who use their smartphones to contribute the data are anonymous, it may not easy to quantify the context similarity between them. Also, it still needs a way to combine the data contributed from multiple drivers effectively to recommend suitable music to individual drivers, as the data only from one driver may not sufficient enough for reference. Thus, in this part, we further develop the **CMM** in SAFeDJ. The implementation of CMM is based on ASCM (Chapter 3) and CSS (Chapter 4) designed in our crowdsensing platform. In general, CSS supports the calculation of context similarity between multiple drivers in a precise manner; and ASCM supports the selection of representatives, so as to maximize the summed up context of them for music recommendation. Therefore, CMM can effectively utilize the crowd data from different anonymous drivers constitute mobile crowdsensing, and enable different in-car smartphones to help each other and recommend preferable music to individual drivers in their new driving situations.

Figure 5.4: Example of similarity computing in CMM
The main idea of CMM is to use statistical methods to explore each type (SC, TC, HMC) of context’s impact on music choice in driving situations, and then use similarity computing methods presented in CSS to calculate the context similarities between multiple drivers to achieve situation-aware music delivery. For example, as shown in Figure 5.4, to recommend a song to Driver A while he is driving, we can calculate the context similarity between Driver A and Driver X (from crowd history dataset shown in Figure 5.1) in terms of their SC, TC and HMC. If the overall similarity between them is high, it means the probability that Driver A and Driver X prefer the same style of music in the similar driving situations is high. Then the satisfaction score of music recorded with the specific driving situations from Driver X could be used as a reference. Likewise, the similarities between other drivers and Driver A could be calculated, and then based on ASCM, we can use an effective way to use their satisfaction scores of music together with Driver X’s score as overall references for music recommendation to Driver A in his new driving situations.

5.3.2.1 Data statistics and similarity computing

Corresponding to the semantic-based matching algorithm of CSS presented in Chapter 4, in CMM, the context similarity between two drivers like Driver A and Driver X is initially defined as $\text{sim}_{\text{ini}}(D_A, D_X)$ in equation (5.1), where $\text{Sim}(D_{A_S}, D_{X_S})$ is the SC similarity, $\text{Sim}(D_{A_T}, D_{X_T})$ is the TC similarity, and $\text{Sim}(D_{A_HM}, D_{X_HM})$ is the HMC similarity, and $W_s$, $W_t$, $W_{hm}$ are the weights for ST, TC, and HMC, respectively. $N_{DA_{S\text{input}}}$, $N_{DA_{T\text{input}}}$, $N_{DA_{H_{\text{input}}}}$ are the number of items about SC, TC, and HMC from Driver A, respectively, and $N_{DX_{S\text{input}}}$, $N_{DX_{T\text{input}}}$, $N_{DX_{H_{\text{input}}}}$ are the number of items about SC, TC, and HMC, respectively, which could be provided in the historic record of Driver X. Finally, the overall context similarity $\text{sim}(D_A, D_X)$ between these two drivers is calculated as the weighted average similarity shown in equation
(5.1).

\[
sim_{ini}(D_A, D_X) = \sum_{j=1}^{N_1} \text{Sim}(D_{A,s}, D_{X,s})W_s + \sum_{j=1}^{N_2} \text{Sim}(D_{A,t}, D_{X,t})W_t + \sum_{j=1}^{N_3} \text{Sim}(D_{A,hm}, D_{X,hm})W_{hm}
\]

\[
N_1 = \min(N_{D_A,s\text{input}}, N_{D_X,s\text{input}}) \quad \text{(5.2)}
\]

\[
N_2 = \min(N_{D_A,t\text{input}}, N_{D_X,t\text{input}}) \quad \text{(5.3)}
\]

\[
N_3 = \min(N_{D_A,hm\text{input}}, N_{D_X,hm\text{input}}) \quad \text{(5.4)}
\]

\[
sim(D_A, D_X) = \frac{\text{Sim}_{ini}(D_A, D_X)}{N_1W_s+N_2W_t+N_3W_{hm}} \quad \text{(5.5)}
\]

To define the values of \(W_s\), \(W_t\), \(W_{hm}\), we need to explore and quantify the impact of \(SC\), \(TC\), and \(HMC\) on music choice while driving. Thus, we set up an online survey [127], in which 505 participants with driver licenses were involved globally. Based on the data collected from this survey, we adopt the professional data statistics software \textit{Stata 13} [128] to analyze the data. In addition, we also use these data as the initial dataset of the crowd history dataset in SAfeDJ cloud, so as to avoid the potential cold-start problem (i.e., no user records of SAfeDJ at the very beginning) [129]. The experimental setting and outputs of \textit{Stata 13} are shown in Table 5.2. From this table, we can observe that with the stated 95% confidence intervals, the absolute values of the averages of coefficients of \(SC\), \(TC\) and \(HMC\) are 0.347, 0.254, and 0.956, respectively; and the corresponding standard deviations are 0.266, 0.314, and 0.312. The higher absolute value of coefficient means the higher impact of the corresponding type of context on music choice. Thus, from these results, and \(W_s+W_t+W_{hm}=1.0\), \(0.374+0.254+0.956=1.584\), we can define: \(W_s=0.374/1.584=0.223\), \(W_t=0.254/1.584=0.163\), \(W_{hm}=0.956/1.584=0.614\). Note, as we have discussed in Subsection 5.3.1, that \(TC\) may impact the mood-fatigue of drivers and hence indirectly impact their preference of music, while here we only define the direct impact of \(TC\) to drivers’ music
choice.

Table 5.2: Statistics about the impact of context to music choice in driving situations

<table>
<thead>
<tr>
<th>Items</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>[95% Confidence Interval]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social context (SC)</td>
<td>-0.347</td>
<td>0.266</td>
<td>-0.703, 0.314</td>
</tr>
<tr>
<td>Traffic context (TC)</td>
<td>0.254</td>
<td>0.314</td>
<td>-0.672, 0.871</td>
</tr>
<tr>
<td>Human mood-fatigue context (HMC)</td>
<td>-0.956</td>
<td>0.312</td>
<td>-1.726, -0.188</td>
</tr>
</tbody>
</table>

Experimental settings: Statistics software: Stata 13; Logistic regression; Number of obs = 505; LR chi2(4) = 9.27; Prob > chi2= 0.0547; Log likelihood = -167.08999; Pseudo R2 = 0.0270

During the development of SAfeDJ, we observe that the data about SC (i.e., gender, cultural background) and TC (i.e., road condition) are usually string data. While as introduced in Subsection 5.3.1, the data about HMC have been normalized as decimals. Thus, we design two similarity computing methods: 1). Ontology-based similarity computing to calculate the similarities of SC - \( Sim(D_{A_S}, D_{XS}) \), and TC - \( Sim(D_{A_t}, D_{Xt}) \), respectively; and 2). Euclidean similarity computing to calculate the similarity of HMC - \( Sim(D_{A_hm}, D_{Xhm}) \).

1) Ontology-based similarity computing

Normally, the data about SC and TC are string data and they usually have implicit contexts. It would be difficult for the traditional keyword based matching (e.g., location based in [130]) to explicitly compare the similarity between two contexts of SC and TC. For example, regarding the SC, directly comparing the two words "Canada" and "UK" as two drivers' cultural backgrounds would return a similarity of 0. Similarly, for TC, directly comparing the two places "Hana Road" and "Hamakua Coast", which are totally different words and locations, would also return a similarity of 0, while their views and road conditions are actually similar. Thus, we adopt the ontology-based similarity computing method designed in CSS (see Subsection 4.4.3.2) to calculate the similarities of SC and TC.
between drivers. In the research community, there already exist several popular ontologies, e.g., WordNet [95] with a large lexical database of English, FOAF [131, 132] that describes the social relations between people, and geo.owl [133] that defines geographic concepts. Considering the high efficiency requirements in the driving situations, we first adopt labeled links to establish relationships between a context and its semantics, so as to enable multiple collected data of SC and TC to be used to compute the context similarity. Then, we can reuse the equation (4.6) presented in Subsection 4.4.3.2 of CSS, which can make use of the above and other existing ontologies to calculate the similarities between two context items of SC and TC in an effective and efficient manner.

\[ Sim(l_1, l_2) = 2C^{k+p+d}\text{depth}_{\text{common}\_\text{ancestor}}/(\text{depth}_{l_1} + \text{depth}_{l_2}) \]  

(5.6)

where common_ancestor is the most specific common parent item, and it can be calculated via the Algorithm 4.1 presented in Subsection 4.4.3.3. After the similarity of each item about SC and TC has been calculated, we can get the \( \sum_{j=1}^{N_1} Sim(D_{A_s}, D_{X_s}) \) and \( \sum_{j=1}^{N_2} Sim(D_{A_t}, D_{X_t}) \) based on equations (5.1), (5.2) and (5.3).

2) Euclidean similarity computing

In Subsection 5.3.2, the contexts of HMC have been defined in seven dimensions (one dimension of fatigue and six dimensions of moods): fatigue-degree, aggressive-degree, neutral-degree, happy-degree, sad-degree, angry-degree, patience-degree; and each dimension is normalized as decimals between 0 and 1. Thus, we just need to calculate the Euclidean distance of each dimension between two drivers’ HMC shown in equation (5.7), and then sum up the results in equation (5.8) to get the \( \sum_{j=1}^{N_3} Sim(D_{A_{hm}}, D_{X_{hm}}) \).

\[ Sim(D_{A_{hm}}, D_{X_{hm}}) = 1 - \sqrt{(D_{A_{hm}} - D_{X_{hm}})^2} \]  

(5.7)
\[ \sum_{j=1}^{N} \text{Sim}(D_{A, hm}, D_{X, hm}) = \sum_{j=1}^{7} \left( 1 - \frac{1}{7} \sqrt{(D_{A, hm, j} - D_{X, hm, j})^2} \right) \] (5.8)

### 5.3.2.2 Context vector

After calculating the similarities between Driver A and all the other drivers, we could select the most suitable drivers with specific driving situations recorded in the crowd history dataset in SAfeDJ cloud as context representatives (which could be considered as an example of crowdsensing participants) for music recommendation to Driver A. To avoid the possible random and subjective factors of single driver that may impact the choice of music, we need to select more than one representative. Also, to make sure the music could be delivered in time in highly dynamic driving scenarios, we need to limit the number of representatives. Thus, in our design, we select the most similar driving situation of Driver A himself (i.e., recorded a few minutes ago in time \( t \)) and marked as \( D_A \), plus the records of the other ten drivers as representatives. Based on the social vector of ASCM we introduced in Chapter 3, to make sure that the summed up contexts of the other ten representatives could be most similar to Driver A’s current context, we can define a group context vector in CMM:

\[ \text{sim}(D_A, D_{sum}) = [\sum_{x=1}^{10} \text{Sim}(D_{A, s, x}, D_{X, s})W_s, \sum_{x=1}^{10} \text{Sim}(D_{A, t, x}, D_{X, t})W_t, \sum_{x=1}^{10} \text{Sim}(D_{A, hm, x}, D_{X, hm})W_{hm}] \] (5.9)

in which the \( \text{Sim}(D_A, D_{sum}) \) means the similarity between Driver A and the summed up context of the other ten representatives, and it should be as large as possible. Also, \( |\sum_{x=1}^{10} \text{Sim}(D_{A, s, x}, D_{X, s})| \leq 1.0, |\sum_{x=1}^{10} \text{Sim}(D_{A, t, x}, D_{X, t})| \leq 1.0, |\sum_{x=1}^{10} \text{Sim}(D_{A, hm, x}, D_{X, hm})| \leq 1.0, \) since the highest similarity is 1.0 as defined in equation (5.5). The process to choose the ten representatives for music recommendation is shown in Algorithm 5.1, in which it computes the similarities of all context items and compares the constitutes of them, so as to select the maximum \( \text{Sim}(D_A, D_{sum}) \).
**ALGORITHM 5.1.** Finding the Ten Most Suitable Context Representatives

**Input:** Similarity items of each context representative candidates.

**Output:** Ten context representatives

1. For each x, compute all the similarity terms of driver x with driver A, and sum up all similarity items as sim(x)
2. Sort sim(x) in descending order
3. Mark sim(0)
4. From i = 1 on, observe individual terms in sim(x), if the constitution of sim(x) is similar to sim(x-1), then throw away sim(x), else, mark sim(x)
5. Repeat 3 until there are 10 sim(*) are marked

Once the $D_{At}$ and the other ten context representatives are selected, since each of them has the corresponding history satisfaction scores of music - $S_x$ (has six dimensions [L,E,S,G,N,P]) along with the driver’s specific driving situations (see Table 5.1 in Subsection 5.3.1), then we can get the overall music recommendation score - $S_{AR}$ for Driver A through:

$$S_{AR} = (S_{At} * mSim(D_A,D_{At}) + \sum_{x=1}^{10} S_x * Sim(D_A,D_x))/(mSim(D_A,D_{At}) + \sum_{x=1}^{10} Sim(D_A,D_x))$$  \(5.10\)

In equation (5.10), we can observe that the higher overall similarity $Sim(D_A,D_X)$ between the representative $X$ and Driver $A$, then the higher weight of $S_x$ in $S_{AR}$. Also, usually the history of Driver $A$ himself has more reference value than others, thus we put a constant value $m$ in this equation and set its default value as 2, and we plan to let the users of SAfeDJ configure m to suit their needs. In addition, if there is a disconnection between the smartphone and the cloud side of SAfeDJ while driving, the context representative will consist of only $D_{At}$, then $S_{AR}=S_{At}$. 

132
5.3.2.3 Music recommendation and satisfaction calculation of songs

As introduced at the beginning of Section 5.3, each song belonging to a driver using SAfeDJ could be processed and calculated in the six dimensions to get the music-mood - $S_{songi}([L,E,S,G,N,P])$. Thus, we just need to compare $S_{AR}$ and each $S_{songi}$ in the MMD (referring Table 5.1 in Subsection 5.3.1.2) of Driver $A$, then the song with the minimum $|S_{songi} - S_{AR}|$ will be selected and recommended to Driver $A$.

**ALGORITHM 5.2. Satisfaction Calculation of Songs and Music Scores Update**

**Input:** start time of the song, end time of the song, $S_{songi}([L,E,S,G,N,P])$ in the MMD, UHT-mood-fatigue history data within the time range of the song, and the history satisfaction scores of music - $S_x[L,E,S,G,N,P]$

**Output:** a score [-1,1] indicating the level of satisfaction the user towards the song, and updated $S_x[L,E,S,G,N,P]$

**Positiveness definition:** happy +2, neutral +1, patient +1, not aggressive +1, sad -1, angry -2, not patient -1, aggressive -1 (with ‘+’ indicates positive and ‘-’ indicates negative)

1. range = [start of song : end of song] > [0:n-1]; for k in [0:n-1]
   
   $p_k = \text{happy}*2 + \text{neutral}*1 + \text{patient}*1 + \text{not aggressive}*1 + \text{sad}*(-1) + \text{angry}*(-2) + \text{not patient}*(-1) + \text{aggressive}*(-1)$ (all of point k)

   for k in [0:n-1]

   Compute $d = (p_{(k+1)} - p_k) - (p_k - p_{(k-1)})$

   If $d \geq 0$, satisfaction += 1, else, satisfaction -=1; normalized satisfaction = satisfaction / n

2. Initialize: threshold $T = x$ (x default value is 0.4)

   If satisfaction $\geq 0$

   satisfaction = satisfaction$*T$

   updated $S_x = \text{satisfaction} * S_{songi} + S_x$

   Else updated $S_x = \text{satisfaction} * S_{songi} + S_x$

   Limit updated $S_x$ to [-1,1]

Furthermore, as discussed in Subsection 5.3.1, we could use the smartphones to detect and record the real-time mood-fatigue data of drivers and store such data in the UHT (Table
5.1 in Subsection 5.3.1). Thus, we could adopt Algorithm 5.2 for the smartphones to automatically calculate the satisfaction scores of each song the driver is listening (to obtain the changing trend of a driver’s moods during the time while he is listening to the song), and synchronize such scores to the cloud side of SAfeDJ to generate/update the historic satisfaction score of the music - $S_x$ in real-time, i.e., data shown in the right side of Table 5.1.

### 5.4 Experiments

![Figure 5.5: Overhead of SAfeDJ in smartphones](image)

We evaluated the basic overhead of SAfeDJ running in popular smartphones in four aspects: (i) CPU usage; (ii) RAM usage; (iii) Basic data consumption; and (iv) Battery consumption, so as to demonstrate the feasibility of the SAfeDJ application in real-world deployment in VSNs. The experimental environment of the cloud side of SAfeDJ is as
follows: **Hardware** - Amazon EC2 M3 2xlarge Instance; 30 GB memory; 8v EC2 Compute Unit; 2*80 GB SSD storage; Moderate I/O Performance; EBS-Optimized Available: No. **Software** - operating system: Windows Server 2008. **Experimental device** – Google Nexus 5. **Vehicle** - 2005 Toyota Sienna. Also, each test lasted for 60 minutes and was repeated ten times, and then the average was calculated. The results of the four tests are summarized in Figure 5.5.

The CPU usage is around 19% and RAM usage is approximate 175MB when running SAfeDJ in Nexus 5 without uploading *MM* for local music analysis. The CPU and RAM usages increase to around 45% (56% in peak period) and 240MB (320MB in peak period), respectively, when running SAfeDJ and uploading *MM* for local music analysis. Considering that the hardware capacity of smartphones is increasing and drivers can hardly operate other mobile applications while they are driving, thus the overhead of SAfeDJ should be acceptable for many popular smartphones.

Without uploading *MM* for music analysis, the data consumption of SAfeDJ is mostly spent in the transmission of music files; hence we only test the basic data consumption of SAfeDJ when finishing local music analysis in two situations. In the first situation, we recorded the data consumption over a stable WiFi network in non-driving scenarios. In the second situation, we recorded the data consumption when using LTE network and driving the 2005 Toyota Sienna in downtown Vancouver at normal speed. Figure 5.5(c) shows that the basic data consumption of SAfeDJ is very low in both situations, only 9.59KB at the beginning for initial account synchronization (i.e., SC of a driver) between the cloud side of SAfeDJ and Nexus 5. Then, the data consumption increases to around 31KB after running SAfeDJ in a Nexus 5 for 60 minutes, which is mainly spent in the synchronization of *HMC*,
TC (i.e., location) and satisfaction score of music, and receiving the music recommendation results from the cloud side of SAfeDJ. Also, we can observe that the results in these two situations are quite close, mainly because SAfeDJ only synchronizes the context with and receives the recommendation results from the cloud side of SAfeDJ 5s before the end of each song as mentioned in Subsection 5.3.1. Thus, the likelihood of additional data consumption resulted by communication signaling is quite low in most common driving situations.

In addition, the battery consumption in the Nexus 5 due to running SAfeDJ is about 16% per hour without uploading MM or 34% when uploading MM. Since the smartphones could usually be charged onboard vehicles, the battery consumption should not be a serious concern for running SAfeDJ onboard vehicles. Furthermore, we observe that the CMM in the cloud side of SAfeDJ can finish the similarity computing for music recommendation and return the results to the Nexus 5 within 3s in all our test cases, which is fast enough for music delivery in most driving scenarios. Therefore, the above results have demonstrated the feasibility of the SAfeDJ application built on our crowdsensing platform in real-world deployment in the future.

5.5 Summary

In this chapter, we have presented SAfeDJ, a novel crowdsensing-based situation-aware music recommendation application for drivers. The development of SAfeDJ further demonstrates how our platform supports the creations of context-aware mobile crowdsensing application. For example, the context vector of CMM in SAfeDJ is built on ASCM presented in Chapter 3, which helps to make sure the summed up contexts of the representatives (which can be considered as a special case of crowdsensing participants in
ASCM) could be maximized. Also, the semantic based matching algorithm of CSS presented in Chapter 4 enables the CMM of SAfeDJ to quantify the similarity of SC and TC between drivers in a more precise manner in the run-time (or execution time). Furthermore, our practical experiments demonstrates the feasibility of SAfeDJ in real-world deployment in VSNs.
Chapter 6: Concluding Remarks and Possible Future Directions

In this chapter, we conclude the dissertation by highlighting the contributions of each chapter and summarizing the results. Also, we present a number of possible directions for further research.

6.1 Summary of Accomplished Work

In this thesis, we have proposed a crowdsensing platform to support the building of context-aware mobile crowdsensing applications in VSNs. We have summarized the system architecture of MSNs, and discussed the unique features and system architecture of VSNs compared with MSNs in Chapter 2. More specifically, our work have mainly focused on two aspects in the overall workflow of mobile crowdsensing in VSNs: task allocation and task execution. We have proposed a novel ASCM to approach the automatic allocation of multiple mobile crowdsensing tasks with multiple participants in an efficient and effective manner in Chapter 3. For the execution of crowdsensing applications, Chapter 4 has introduced S-Aframe, an agent-based multi-layer framework with CSS which provides systematic approaches that enable the self-adaptivity of different mobile crowdsensing applications in high dynamic environment of VANET-based VSNs. In Chapter 5, we have designed and developed SAfeDJ, a situation-aware music recommendation application for drivers as an example, to further demonstrate how our platform supports the creation of context-ware mobile crowdsensing applications and facilitate the real-world deployment of them in VSNs.

- In Chapter 2, to form a good foundation for our research targets, we have first conducted a comprehensive survey on the system architecture of MSNs from the perspective of mobile software system and mobile computing, which is still lacking
in the literature. We have studied the system architecture of MSNs through three architectural views from different stakeholders’ perspectives: physical view from the aspects of system engineers of MSNs, development view from the aspects of MSN application developers, and logical view from the aspects of end users of MSNs. After that, we have discussed the unique features and system architecture of VSNs compared with MSNs, and analyzed the specific challenges of mobile applications in VSNs.

- In Chapter 3, we have proposed ASCM, a novel application-oriented service collaboration model to approach the automatic allocation of multiple mobile crowdsensing tasks with multiple participants, so as to enable the tasks to be finished in an efficient and effective manner. To the best of our knowledge, ASCM is the first work that investigates how to quantify the relationships between multiple participants, crowdsensing tasks and crowd groups in crowdsensing, while supporting automatic allocations of multiple crowdsensing tasks with multiple crowdsensing participants across mobile devices and cloud platform efficiently and effectively during run-time. Furthermore, we have designed and developed a generic RESTful Web Service-based mobile cloud platform called Vita, to verify the feasibility of ASCM and explore how to facilitate the realization of ASCM in VSNs for real-world deployment. Also, the Vita platform provides a generic model to deploy, examine and evaluate different context-aware solutions for mobile crowdsensing applications. Based on the works of ASCM and Vita platform, we have developed a novel context-aware mobile crowdsensing application – *Smart City* to demonstrate the expressivity and usefulness of ASCM in our daily lives’ traveling. Our practical experiments have demonstrated the efficiency and effectiveness of ASCM when it is deployed on Vita.
for multiple crowdsensing tasks in common situations. Also, the experimental results have verified ASCM performs better in the allocation of multiple crowdsensing tasks with multiple participants, compared to the existing solutions for crowdsensing, which form multiple crowd groups randomly in run-time.

- In Chapter 4, we have presented S-Aframe, an agent based multi-layer framework with context-aware semantic service (CSS) that is able to utilize context information and provides a high level software platform for VANET-based mobile crowdsensing applications. S-Aframe hides the complexity of dealing with changing network connectivity and varying user services requirements in VSNs, by providing a programming model with extensibility support. S-Aframe also provides a rich set of framework services to support application development using Java with standard API format. Therefore, S-Aframe provides a modeling paradigm which can facilitate the creation and deployment of different mobile crowdsensing applications to self-adapt to the dynamic network connectivity status in VANET-based VSNs. Also, we have designed a new model CSS that integrates software agent and semantic techniques in S-Aframe to effectively manage and utilize various context information for different crowdsensing applications in VSNs. CSS can enable agent based mobile crowdsensing applications developed on S-Aframe to intelligently determine what and how services and information should be delivered to users in crowdsensing, and autonomously adapt the behaviors of these services to users' dynamic contexts during run-time. We have extensively evaluated S-Aframe through practical experiments on Android devices, which showed that the crowdsensing applications created on S-Aframe can perform well under dynamically changing connectivity, and achieve
good time efficiency with low computation and communication overhead. Furthermore, we have designed and implemented a ridesharing mobile crowdsensing application based on S-Aframe to demonstrate its practical feasibility and usefulness for the real-world VSNs.

- In Chapter 5, we have designed and developed SAfeDJ, a crowdsensing-based situation-aware music recommendation application for drivers. The development of SAfeDJ has further demonstrated how the solutions designed in our platform support the creation of context-aware mobile crowdsensing applications, and facilitate the realization of such applications in real-world deployment in VSNs. To the best of our knowledge, SAfeDJ is the first crowdsensing-based music recommendation application that specifically designed for drivers. This application enables different smartphones to collaboratively recommend preferable music to individual drivers under driving situations that the drivers may or may not have experienced before, hence diminishing fatigue and negative emotion of drivers. Also, our practical experiments demonstrates the feasibility of SAfeDJ in real-world deployment in VSNs.

In conclusion, our accomplished works in this thesis have provided a crowdsensing platform to address the research challenges in the overall workflow of crowdsensing in terms of task allocation and task execution. This platform supports the creation of different context-awareness mobile crowdsensing applications in VSNs, and facilitates the real-world deployment of such applications in VSNs in the future. In particular, the development of SAfeDJ (situation-aware music recommendation system for drivers) that has been introduced in Chapter 5 was based on our platform for task allocation and task execution of
crowdsensing proposed in previous chapters. For example, based on ASCM presented in Chapter 3, we have designed the context vector in CMM of SAfeDJ to make sure the summed up contexts of the representatives (which can be considered as a special case of crowdsensing participants in ASCM) could be maximized. Also, based on the semantic based matching algorithm presented in CSS in Chapter 4, we have applied ontology-based similarity computing method in CMM that enables CMM to quantify the similarity of $SC$ and $TC$ between drivers in a more precise manner in the run-time (or execution time) of SAfeDJ. In addition, the cloud side of SAfeDJ is built on the Vita cloud platform presented in Chapter 3.

6.2 Future Directions

In the following, some possible directions for extending our work presented in the dissertation are provided.

1. Security and privacy of mobile crowdsensing applications in VSNs: security and privacy are always common concerns for mobile crowdsensing applications over VSNs. Even though our current work does not take these challenges into account, they should be addressed in the future. For example, as presented in Chapter 4, authentication and access control are vital for security and privacy of VANET-based crowdsensing applications given the open environment and the possible presence of threats where adversaries can monitor and expose sensitive data of other participants. We can adopt a combined authentication-based multilevel access control to preserve privacy in collaborative environments, or integrate complementary security mechanisms of PKI for VANETs. Also, for the SAfeDJ presented in Chapter 5, it is necessary to preserve the privacy of the
users, e.g., in terms of their geographical locations and personal music preference. Different approaches have been proposed to provide privacy-preservation by transforming the data, e.g., adding noise [134]. In the context of location privacy, it ensures that the user’s location is indistinguishable from at least K-1 other users. To achieve K-anonymous location privacy, one common approach is to incorporate a trust server that is responsible for removing the user’s ID and selecting an anonymizing spatial region containing the user and at least K-1 other users [135]. We can explore how to apply these cloaking and anonymization techniques to provide privacy preservation on the geographical location and music interests of users in SAfeDJ in the future.

2. **Impact of alternative traffic models and schemes for vertical handover:** as discussed in Chapter 4, the foreground traffic characteristics of S-Aframe may depend on the type of deployed crowdsensing applications. For the applications involving real-time mobile group communications, the foreground traffic generated may exhibit characteristics similar to the mobile instant messaging (MIM) [136] that follows an exponential distributions as suggested by 3GPP [137]. However, a more recent study involving an analysis of MIM traffic from 7 million users in a large-scale cellular network suggests that a lognormal distribution may be a better fit for the inter-arrival time distribution of MIM user messages [138]. Thus, further investigation of S-Aframe can be done on the impact of alternative traffic models for the foreground traffic, such as the Joint ON/OFF model proposed in [138]. In addition, the underlying network connectivity of S-Aframe is based on pure VANETs, which are highly dynamic
networks that may not be always available. If network connectivity is required but a VANET is unavailable, one may need to switch to a cellular network or a nearby WiFi access point as a fallback. In this case, a vertical handover scheme could be designed in the future, to enable S-Aframe to dynamically switch between VANETs and cellular/WiFi networks.

3. **Cross-platform testbed to validate different context-aware solutions for mobile crowdsensing applications:** mobile crowdsensing is an emerging topic and only appear in the research communities in the recent years, a number of new research work has been proposed to facilitate context-aware crowdsensing in mobility environments (i.e., transportations) by using mobile devices. However, throughout our research work, we find that the existing solutions for context-aware crowdsensing are usually implemented on different kinds of platforms, e.g., our solutions are implemented on Android platform, while others may be implemented on iOS or J2ME etc. Thus, it makes direct and fair comparison between our solutions and the alternative approaches at times difficult if not impossible. Also, to the best of our knowledge, there is no standard baseline for the evaluations of different mobile crowdsensing solutions yet. As introduced in Chapter 3, we have designed and developed a generic RESTful Web Service-based mobile cloud platform called Vita to verify the feasibility and investigate how to facilitate the realization of our ASCM approach for VSNs in real-world deployment. Also, Vita could potentially be used as a generic model to evaluate different context-aware solutions for mobile crowdsensing applications. Nevertheless, the current implementations of Vita is still based on
the Android platform and the IaaS cloud - Amazon EC2, and could not run on other platforms yet. Thus, it would be valuable to explore how to extend Vita to other platforms (i.e., iOS) as a cross-platform testbed in the future.
Bibliography


[33] A. Karam, and Nader Mohamed, "Middleware for Mobile Social Networks: A


2011, pp. 1-6.


and Applications (WMCSA), 2004, pp. 41-50.


[137] 3GPP, “GERAN study on mobile data applications V0.4.0,” Tech. Rep., 2013


Appendix A

Music-mood Mapping (MM)

To recommend suitable music that fits the drivers’ real-time moods, we need to identify the corresponding music-moods that each song expresses. The main idea of MM is to process each musical piece inside a user’s smartphone. For each piece of music, MM extracts a six-dimensional musical feature, classifies it to one of the existing clusters generated by some well selected representative pieces of music, and calculates a mood label based on the music representatives in that cluster. The final output of MM is a dictionary with song titles as keys and their mood labels as values. MM consists of two processes: (i) Music feature extraction, and (ii) Music mood labeling.

A.1 Music feature extraction

Here we define a six-dimensional music representation as shown in Figure A1. It is based on music representations of well-known music genre classification works [149]. These six dimensions are zero-crossing rate, unit power, low energy rate, tempo, spectral centroid, and tonal type. The first four are temporal features and the last two are spectral features. Accordingly, each piece of music can be represented as a point in the six-dimensional space.

Figure A1: Six-dimensional music representation
The feature extraction starts from reading the digitized musical piece, which can be in any popular format (such as wav, aiff, aac, mp3, etc.), followed by combining the two channels into one, normalizing it and storing it into an array. With this monaural music array and the audio sampling rate, the zero crossing rate, unit power, and low energy rate could be calculated as follows [149]:

Let x=music array,

\[
\text{zero crossing rate} = \frac{1}{N} \sum_{n=1}^{N} |\text{sign}(x[n]) - \text{sign}(x[n - 1])| \tag{A.1}
\]

\[
\text{unit power} = \frac{1}{N} \sum_{n=1}^{N} x[n]^2 \tag{A.2}
\]

\[
\text{low energy rate} = \frac{\# \text{ of point } k \text{ with } x[k]^2 < \text{unit power}}{\text{length of } x} \tag{A.3}
\]

\textbf{Tempo}: Based on a well-established tempo tracking algorithm as described in [149], we design and implement a new music tempo module. The algorithm of this module goes in several steps. First we take a down-sampled copy of the absolute value of the music array with down-sample rate 32:

\[
y = \text{downsample}(|x|, 32) \tag{A.4}
\]

Then we compute the autocorrelation of the above output, and limit its minimum value to 0:

\[
y'[k] = \max(\sum_n y[n]y[n - k], 0) \tag{A.5}
\]

Then we take an approximately 4 seconds section of the above output:

\[
y'' = y'[\frac{fs}{32 \times 4} : \frac{4 \times fs}{32}] \tag{A.6}
\]

where 32 is the down-sample rate. The above output is fed to a 50th-order one dimensional median filter for smoothing. We then calculate a 5th order polynomial fitting of the above output, and subtract the post-fit data from the pre-fit data, followed by a normalization.
process. The resulting data should contain clear peaks whose positions indicate the tempo of the original musical piece. Then we apply a peak picking algorithm to the data, with minimum peak height of 0.5 and minimum peak distance $fs / (32 \times 4)$ as constraints. With a scaling factor to transform the current scale back to the beats per minute (bpm) scale, the first peak position in the bpm scale is the tempo in question.

**Spectral centroid:** The original music array in time domain needs to be transformed into frequency domain before calculation. The transformation is performed by applying discrete Fourier transform to the array:

$$X_k = \sum_{n=0}^{N} x_n e^{-\frac{i2\pi kn}{N}}, k \in Z \text{(integers)}, \quad (A.7)$$

where $N = 2^t, t \in Z, such that 2^{t-1} < \text{(length of } x) \leq 2^t$

Then we take the single side amplitude spectrum by taking the absolute value of the first half of $X$.

$$X' = |X[0:\frac{N}{2}]| \quad (A.8)$$

Also, the original linear spaced number scale is updated to linear frequency scale:

$$f_k = \frac{fs \times k}{N}, k \in [0, \frac{N}{2}] \quad (A.9)$$

With these, the spectral centroid can be calculated as:

$$\text{spectral centroid} = \frac{\sum_k X'_k f_k}{\sum_k X'_k}, k \in [0, \frac{N}{2}] \quad (A.10)$$

**Tonal type:** The last feature tonal type indicates the overall color of a musical piece. Generally there are 3 tonal types, namely, major, minor, and none [150]. To calculate tonal type, we first transform the amplitude spectrum into a chromagram, which is a representation of the accumulated strength of every musical pitch classes within a given music piece signal [151]. To get a chromagram, the frequency bins of amplitude spectrum below 20Hz is cut,
and then all the bins’ frequency dimensions are mapped to the midi number dimension,

\[
M_k = \text{round}(12 \times \log_2 \left( \frac{f_k}{440} \right) + 69)
\]  
(A.11)

where 69 is the midi number of the ‘A’ above middle ‘C’ (frequency = 440 Hz). This ‘A’ note is provided as the standard reference for converting frequency number to midi number [152]. The midi numbers are then mapped to midi pitch classes (MPCs) simply by taking modulo 12.

\[
\text{MPC}_k = \text{mod}(M_k, 12)
\]  
(A.12)

The 12 bin chromagram is finalized by summing up values of all MPCs with the same pitch class. After we have a 12 bin chromatagram, we proceed to calculate tonality by applying a tonal gravity idea, which is based on the idea describe in [151]. If the tonal gravity is at pitch x, and tonal type is major, then the sum of the chromatagram at bins located at x major scale will have the maximum value among all scales. The same is also true for the tonal gravity of minor tonal type.

\[
\text{major}[k] = \text{chroma}[k] + \text{chroma}[(k + 2)\%12] + \text{chroma}[(k + 4)\%12] + \text{chroma}[(k + 5)\%12] + \text{chroma}[(k + 7)\%12] + \text{chroma}[(k + 9)\%12] + \text{chroma}[(k + 11)\%12]
\]  
(A.13)

\[
\text{minor}[k] = \text{chroma}[k] + \text{chroma}[(k + 2)\%12] + \text{chroma}[(k + 3)\%12] + \text{chroma}[(k + 5)\%12] + \text{chroma}[(k + 7)\%12] + \text{chroma}[(k + 8)\%12] + \text{chroma}[(k + 10)\%12]
\]  
(A.14)

Note that major [(k+3) \% 12] = minor [k], where “\%” is the modulo operator. We compare the chroma bin value maximum entry of both major and minor, take (b+3) \% 12 and b for example, if chroma [(b+3) \% 12] > chroma [b], the tonality will be a major at (b+3) \% 12, if chroma [(b+3) \% 12] < chroma [b], the tonality will be minor at b, otherwise, there will be no tonal gravity nor tonal type.
A.2 Music mood labeling

Algorithm A.1 describes the process of music mood labeling.

**Algorithm A.1. Music Mood Labeling**

**Input:** Musical pieces.

**Output:** A [song title, mood label, music feature] dictionary of the musical pieces.

**Initialize:** static references = 21 clusters centered at 21 music representatives’ feature points in 6D space, all of the 21 points are initialized with mood labels; distance threshold DT, for each musical piece in this music library: p = this musical piece; fe = music feature extract(p)

1. **if** distance of p and every references(static+dynamic) is larger than DT
   create a new cluster centered at p, and assign mood label of nearest point to p
2. **else**
   cluster p to the nearest cluster
   *update the cluster centroid to be the mean of every points in this cluster*
   *assign mood label to p using the mean of mood labels in this cluster*
3. **async task:** dynamic references = a new music point whose mood label is confirmed by crowd history

**Static reference:** Our final goal is to assign a mood label to each musical piece going through this music mood mapping process, but the big question is where the “mood” comes from. Just as can be seen in a lot of online music channels, human tends to affiliate mood labels to music, such as “groovy”, “sad”, “happy”, “lively”, “energetic”, etc. These are evidences showing that music has not only “musical” dimensions, but also “mood” dimensions. Also, there are underlying connections between these two sets of dimensions [153]. In order to establish a reference point that could serve as the ground truth of mood representation of music, we first set up also a six-dimensional space representing music mood based on previous music mood researches [153]. The representation is [lively, energetic, sentimental, groovy, noise, peaceful], or simply, [L,E,S,G,N,P], each with a value ranging from 0 to 1. Then we manually chose 21 representative songs, each based on our
own humanized judgment on its mood being dominated by only one dimension or only two
dimensions (totally 21 possibilities). After that, we involved 224 participants from the
general public in our online survey [154], and they were asked to vote on each song the
dominant moods, given the [L,E,S,G,N,P] multiple selections. After voting, a six-
dimensional mood label is generated at each entry, showing the percentage of positive votes
for the song’s [L,E,S,G,N,P] label respectively. Note that we do not assume that the voting
results will be as the same as what we initially think these songs might be, but our effort is
only to try to make sure that these 21 songs can represent as many as 21 different musical
classes. Thus these 21 songs serve as a static reference, which refers to 21 static music
clusters, each of which contains one point in the six dimensional music feature space that is
attached with a confirmed mood label in the six-dimensional mood feature space.

**Music clustering:** When a piece of music is input to this process, it is first reduced to
a 6-dimensional music feature using the process described in Subsection 5.3.1.1, and then
this feature is compared to the centroids of the existing clusters. The centroid with smallest
Euclidean distance wins and this new input is grouped into that cluster.

**Dynamic reference:** Considering that the 21 songs used in the static reference may not
represent all the possible styles of different music precisely, we also adopt the scheme of
dynamic reference. For example, if there is a song title in crowd history with a high
satisfaction score (e.g., >0.8), and it has already been recommended 10 times, it is considered
as a dynamic reference point, which can be added to the reference pool together with its
mood label in crowd history. If it is already in the reference pool, only the mood label is
updated. Once this dynamic reference is added, it is also being clustered as described in the
above subsection.
A New cluster: Similar to dynamic reference, a new cluster is created with center at this point and the mood label of its nearest point is assigned to it, when a new input is too far away from (e.g., distance of overall music features more than the smallest distance between the static references) all the static references. Also, this point is counted as one of the static reference points.

Mood calculation: The mood label of a new input is assigned as the mean of all the reference points’ mood labels of the cluster it belongs to. Simultaneously, all the mood labels in that cluster will be refreshed whenever a new reference point is added to an old cluster.

Music mood dictionary – final output

Table A.1: Music mood dictionary (MMD)

<table>
<thead>
<tr>
<th>user_id</th>
<th>Song Title</th>
<th>$S_{song\cdot music_mood}$ [L,E,S,G,N,P]</th>
<th>music_features</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>American Idiot - Green Day</td>
<td>[0.434783, 0.913043, 0.086957, 0.347826, 0.173913, 0]</td>
<td>[0.152975, 0.110374, 0.698956, 0.575, 0.165831, 1]</td>
</tr>
<tr>
<td>0</td>
<td>Better That We Break-Maroon</td>
<td>[0.304348, 0, 0.565217, 0.043478, 0, 0.347826]</td>
<td>[0.157671, 0.093225, 0.715329, 0.49375, 0.201159, 1]</td>
</tr>
<tr>
<td>0</td>
<td>Bring Me to Life-Evanescence</td>
<td>[0.086957, 0.304348, 0.608696, 0.173913, 0.304348, 0.173913]</td>
<td>[0.143611, 0.09379, 0.711815, 0.78125, 0.167227, -1]</td>
</tr>
<tr>
<td>0</td>
<td>Dancing Queen-ABBA</td>
<td>[0.695652, 0.521739, 0.130435, 0.521739, 0.086957, 0.130435]</td>
<td>[0.163954, 0.064364, 0.703567, 0.61875, 0.163235, 1]</td>
</tr>
<tr>
<td>0</td>
<td>Don’t Know Why-Norah Jones</td>
<td>[0.086957, 0, 0.521739, 0.086957, 0, 0.782609]</td>
<td>[0.075943, 0.021805, 0.790892, 0.36875, 0.109837, 1]</td>
</tr>
</tbody>
</table>

After all the above steps, a musical piece is mapped to a mood label. Thus the final output of this process applying to a smartphone running SAfeDJ is a music mood dictionary
(MMD) with song titles as keys and their mood labels - $S_{song}$ as values as shown in Table A.1.

Furthermore, considering the intensive computing overhead of $MM$ on smartphones, we deploy the $MM$ both on smartphone and SAfeDJ cloud. Then the cloud could assist smartphones to run $MM$ and directly return the results (i.e., music dictionary shown in Table A.1) to the smartphones, so as to improve the efficiency and decrease battery consumption of smartphones when running SAfeDJ. Some experimental results about this will be presented later in A.3.

A.3 Accuracy and efficiency of MM

Table A.2: Case study about the accuracy of MM

<table>
<thead>
<tr>
<th>Name of Song</th>
<th>Lively(L)</th>
<th>Energetic(E)</th>
<th>Sad(S)</th>
<th>Groovy(G)</th>
<th>Noisy(N)</th>
<th>Peaceful(P)</th>
<th>Overall Distance (Max:2.45)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>[0.348-0.439] = 0.091</td>
<td>[0-0.041] = 0.041</td>
<td>[0.391-0.366] = 0.085</td>
<td>[0.044-0.055] = 0.011</td>
<td>[0-0] = 0</td>
<td>[0.522-0.918] = 0.396</td>
<td>0.418</td>
</tr>
<tr>
<td>Haoting</td>
<td>[0.696-0.776] = 0.080</td>
<td>[0.522-0.306] = 0.216</td>
<td>[0.130-0.041] = 0.089</td>
<td>[0.522-0.384] = 0.138</td>
<td>[0.087-0] = 0.087</td>
<td>[0.130-0.459] = 0.329</td>
<td>0.443</td>
</tr>
<tr>
<td>Manhattan</td>
<td>[0.696-0.449] = 0.247</td>
<td>[0.522-0.225] = 0.297</td>
<td>[0.130-0.031] = 0.099</td>
<td>[0.522-0.505] = 0.227</td>
<td>[0.087-0.020] = 0.067</td>
<td>[0.130-0.541] = 0.411</td>
<td>0.577</td>
</tr>
<tr>
<td>WannaBe</td>
<td>[0.261-0.459] = 0.198</td>
<td>[0.870-0.776] = 0.094</td>
<td>[0.435-0.236] = 0.149</td>
<td>[0.435-0.455] = 0.020</td>
<td>[0.044-0.286] = 0.242</td>
<td>[0.044-0.225] = 0.181</td>
<td>0.403</td>
</tr>
<tr>
<td>Nizaiganka</td>
<td>[0.609-0.532] = 0.077</td>
<td>[0.043-0.165] = 0.122</td>
<td>[0.304-0.190] = 0.114</td>
<td>[0.478-0.354] = 0.124</td>
<td>[0-0.101] = 0.101</td>
<td>[0.261-0.139] = 0.122</td>
<td>0.273</td>
</tr>
<tr>
<td>Huahuo</td>
<td>[0.609-0.190] = 0.419</td>
<td>[0.043-0.139] = 0.096</td>
<td>[0.304-0.494] = 0.190</td>
<td>[0.478-0.203] = 0.275</td>
<td>[0-0.013] = 0.013</td>
<td>[0.261-0.418] = 0.157</td>
<td>0.567</td>
</tr>
<tr>
<td>AndLove her</td>
<td>[0.348-0.076] = 0.272</td>
<td>[0-0.051] = 0.051</td>
<td>[0.391-0.544] = 0.153</td>
<td>[0.043-0.089] = 0.046</td>
<td>[0-0.013] = 0.013</td>
<td>[0.522-0.671] = 0.149</td>
<td>0.353</td>
</tr>
<tr>
<td>FuzzUniverse</td>
<td>[0.217-0.430] = 0.213</td>
<td>[0.435-0.722] = 0.287</td>
<td>[0-0.013] = 0.013</td>
<td>[0.304-0.228] = 0.076</td>
<td>[0.215-0.652] = 0.437</td>
<td>[0-0] = 0</td>
<td>0.570</td>
</tr>
<tr>
<td>Neverwak eup</td>
<td>[0.217-0.190] = 0.027</td>
<td>[0.435-0.316] = 0.119</td>
<td>[0-0.063] = 0.063</td>
<td>[0.304-0.076] = 0.228</td>
<td>[0.652-0.747] = 0.095</td>
<td>[0-0] = 0</td>
<td>0.247</td>
</tr>
<tr>
<td>Waltzfor debby</td>
<td>[0.0-0.089] = 0.089</td>
<td>[0-0.013] = 0.013</td>
<td>[0.696-0.392] = 0.304</td>
<td>[0.043-0.101] = 0.058</td>
<td>[0-0.038] = 0.038</td>
<td>[0.783-0.671] = 0.112</td>
<td>0.343</td>
</tr>
</tbody>
</table>

We randomly picked 10 songs which cover different styles of music and involved 183 volunteers globally to listen to and vote on each song with respect to the dominant moods, and then calculate the voting for the percentage of each song’s [L,E,S,G,N,P]. Simultaneously, we used our $MM$ methods to calculate the [L,E,S,G,N,P] of each song and compared the results...
with the respective results from the online survey. The closer (smaller distance) are the results from these two methods, the higher is the accuracy of our MM methods in the identification of music-mood in real-world. In addition, each song’s music-mood dimension has a maximum distance value 1.0, and the corresponding maximum value of the overall distance of each song is 2.45.

The experimental results are summarized in Table A.2, which shows that regarding each song’s music-mood dimension, the distance is generally around 0.1 between our MM methods and the results from online voting, and the maximum distance is 0.437 in the N dimension of the song FuzzUniverse, while the overall distance is normally around 0.247~0.577 for each song, and 0.419 in average for the 10 songs. Therefore, the results demonstrate that the music-mood dimensions calculated by our MM methods are close to the results from people’s real feeling about these songs in the real-world. Furthermore, since our MM methods support self-update of the music clustering with music moods, the accuracy will be improved when there are more and more users of SAfeDJ in the future.

![Figure A.3: Time efficiency of MM](image-url)
Moreover, we conducted experiments about the time efficiency of MM to analyze music in smartphones. We used 180 songs in mp3 format in our tests, with each song having two bit rates (128kbps and 320kbps). Based on the type of song’s bit rate, we divided the tests into two cases, and recorded the average time for music analysis (overall time divides 180) in each case. Also, there are three situations in each case: (i) Operating MM in SAfeDJ cloud to analyze the music under LTE network under normal driving situations (e.g., around 40~60km/h driving speed in urban areas); (ii) Operating MM in SAfeDJ cloud to analyze the music under stable WiFi network in non-driving situations; and (iii) Only operating MM in smartphones to analyze the music. The experimental environment of SAfeDJ cloud was as follows: Hardware - Amazon EC2 M3 2xlarge Instance; 30 GB memory; 8v EC2 Compute Unit; 2*80 GB SSD storage; Moderate I/O Performance; EBS-Optimized Available: No. Software - operating system: Windows Server 2008. Experimental device - Samsung Galaxy S4. Vehicle - 2005 Toyota Sienna.

The results about the time efficiency of MM is shown in Figure A.3. We find that the average processing time in SAfeDJ cloud is about 26.9s on each mp3 song with bit rate of 128kbps and about 4MB file size, while 33.61s when the mp3 song has a bit rate of 320kbps and about 10MB file size. Since the communication time is mainly made up of the time for uploading the mp3 file from Galaxy S4 to SAfeDJ cloud for processing, this time mostly depends on the size of the mp3 file and network condition (WiFi or LTE). Also, we observe that the uploading speed of the mp3 file may decrease occasionally due to adverse channel conditions and latency of handovers encountered during our driving tests. In most cases, however, the uploading speed of SAfeDJ over an LTE network is very stable in normal driving situations, and the maximum delay is 39.06s in our experiments. This is because
LTE is the most recent generation of cellular networking technology with a high performance and support for terminal mobility at a speed up to 350km/h [155]. In addition, a lot of research works have addressed vertical handovers in heterogeneous wireless networks that support seamless connectivity for smartphones and other terminals in vehicular networks [156], and different mechanisms can be integrated in SAfeDJ to better support vertical handovers in the future. On the other hand, Figure A.3 shows that the average processing time of $MM$ in Galaxy S4 are 104.64s and 138.43s, respectively, for the two bit rates. Since a song normally plays for more than 3 minutes, thus the probability that $MM$ can process a new mp3 song in time while a driver is driving and using SAfeDJ should be high.
Appendix B

Mood-fatigue Detector

B.1 Implementation of the mood-fatigue detector

The working flow of the mood-fatigue detector presented in Subsection 5.3.1 is shown in Figure B1. Firstly, to determine the aggressiveness and patience of a driver, an OBD-II scanner is used to obtain driving data from the car, e.g., the RPM, Tht, and Sp; and these data are complimented by the data of Hr from a heart rate sensor. For example, driving at a high speed and braking hard may indicate aggressive driving behavior. Also, we can infer that a driver is likely aggressive in driving if the average RPM of vehicle is more than 4000 and the throttle input degree of the pedal is 100 percent. Similarly, the history of heart rate and driving behavior could be used to infer the driver’s calmness or degree of being annoyed.

Figure B1: Working flow of the mood-fatigue detector

While to detect the remaining four dimensions (neutral, happiness, sadness, anger) of a driver’s mood in a precise manner, we adopt facial expression to detect the driver’s emotion in real-time, which is achieved by utilizing the front camera of a smartphone. Then as shown
in Figure B2, we apply the cascade classifier to locate the face of the driver in the pictures and then use the face recognizer in the OpenCV-2.4.9 library [157] to identify the mood of the driver. The recognizer is trained by the eigenface algorithm [158] to increase the accuracy on mood detection.

![Figure B2: Facial expressions and eye-lid detector](image)

On the other hand, we also invoke the front camera of smartphones and integrate the $EI$ detection function to analyze the fatigue degrees of drivers. We use the $EI$ distance of the driver’s eyes to analyze the degree of drowsiness. First, we locate the eyes in the picture and measure the distance between the upper and lower eye-lids. We observe the driver’s average eye-lid distance under neutral conditions, and use it as a reference to calculate the relative percentage on how wide the driver opens his eyes. From our initial testing, the drivers are often tired if the relative percentage is less than 50 percent. In addition, the data about in-vehicle temperature ($T$) and road congestion situation could also be used as complements to infer the fatigue degrees of drivers in a more precise manner.

**B.2 Effectiveness of mood-fatigue detector**

To evaluate the effectiveness of the mood-fatigue detector, we involved ten volunteers (five males and five females) on campus to conduct a set of tests. At the same location under normal light, each subject was seated in front of a smartphone (Google Nexus 5) running
SAFeDJ with a five-degree angle to simulate the normal driving position. Then the subject was requested to do five tests, each lasting for 300s, to evaluate the effectiveness of detections for: **fatigue** and four moods - **neutral**, **sad**, **happy**, and **angry**, respectively.

To test the accuracy of fatigue detection, each subject was asked to pretend that he/she was tired, and express his or her common reactions under such status, like closed or squint eyes frequently. Similarly, in the tests for detections of the four moods, each subject was asked to express the four types of moods in terms of how they may appear as their daily facial expressions. As mentioned in Subsection 5.3.1, each item of the mood-fatigue could be quantified as a degree which value ranges from 0 to 1.0. For example, in every ten-second period, if the fatigue status of a subject is detected 7 times and the non-fatigue status is detected 7 times, then fatigue degree of him/her has a value of $7/(7+7)=0.5$ in this period. This means that the higher value of the fatigue degree detected in our test, the higher is the effectiveness of our mood-fatigue detector. Also, as the results may be impacted by the sequences of the five tests, e.g., people’s faces may get tired or stiffed after expressing a certain mood for a long time, we assigned a random sequence of these tests to each subject to avoid such impacts. Finally, after all the data about the ten people are recorded in each test, the average values of them were calculated.

The experimental results of the five tests are shown in Figure B3. We can observe that the average detected value of the fatigue degree is around 0.8, which means most of the fatigue expressions (i.e., tired eyes) from the people could be detected by our mood-fatigue detector. Regarding the detections of the four moods, the average value of the neutral mood is around 0.9 indicating a high effectiveness of detection, since this is the most common mood status of people. However, for the detections of happy and sad moods, the detected
average values of them are both around 0.6, as the facial expressions of such moods are similar to neutral expression for some people. As the most diversified expression, the detected average value of angry is around 0.5 in the corresponding test. Moreover, we make a simple comparison of our mood-fatigue detector with a similar approach found in the literature [159]. Under similar situations, the effectiveness of their approach using smartphones to detect the moods of drivers is around 0.9 for neutral and happy, 0.3 for sad, and 0.4 for angry, respectively. Even though our approach and theirs are running on different Android devices, a key reason for the better detection of negative moods in our approach is that we lower the threshold in the determination of drivers’ negative moods in our algorithm.

![Figure B3: Experimental results about the detections of mood-fatigue degree](image)

Furthermore, it is worth noting that as presented in Subjection 5.3.1, beyond using the front camera of the smartphone to detect the mood-fatigue of drivers like the above tests, our solution also supports using the data from OBD and heart rate sensors simultaneously to facilitate detection of the mood-fatigue of drivers while driving. For instance, the OBD could
be used to detect hard accelerations of drivers and infer the aggressiveness and mood-fatigue degrees of them. Also, as demonstrated in the related work [160], the effectiveness of prediction of drivers’ mood-fatigue degree can even increase to 94% when multiple wearable sensors are used simultaneously with smartphones. Due to the experimental constraints of these tests, e.g., it would be unsafe to require the volunteers to do multiple hard accelerations while driving, we have not evaluated these approaches in practice, but only used front cameras of smartphones for mood-fatigue detections in our tests. It means that in the real-world scenarios during driving, our solution for the mood-fatigue detection could have a higher effectiveness when data from the OBD and heart rate sensors are also incorporated in the detection process.
Appendix C

Case study of SAfeDJ

To validate the SAfeDJ application can deliver preferable music to drivers, we can use the mood-fatigue detector implemented in the context model (see Appendix B) of SAfeDJ to record the mood-fatigue history of drivers. The lower recorded fatigue degree and negative mood degree of drivers by using SAfeDJ than using the smartphone’s default music player in similar driving situations, implies SAfeDJ can deliver preferable music to drivers. Also, it means our crowdsensing platform enables the SAfeDJ application to achieve the function – situation-aware music recommendation for drivers well. In addition, since it would be difficult to involve multiple drivers to use SAfeDJ and drive in the same road and traffic conditions in experimental tests, thus similar to the former research works in driving behavior analysis [161, 162], we adopt a driving simulator as shown in Figure C1(a) to conduct this case study.

Figure C1: Experimental driving scenarios

---

1 The experimental work reported in this appendix was undertaken at East China Normal University, Shanghai, China under the supervision of Prof. Jidi Zhao (email: jdzhal@sem.ecnu.edu.cn), who holds the necessary certificate for experimental tests involving human subjects.
In our user studies, we involved 48 volunteers (32 males and 16 females) with driver licenses. Currently, people usually listen to music via smartphones in two ways: (a) Collecting and storing the preferable songs in personal smartphones, and playing such songs on the smartphones directly; or (b) Playing all the songs on the smartphones by accessing popular online music streaming services. Thus, we divided the 48 volunteers to two groups, \textit{group a} and \textit{group b}, each consisting of 24 people. Each driver in \textit{group a} picked 40 songs that were most preferable to him/her and used them in all his/her tests. Drivers in \textit{group b} all accessed the same music library consisting of 180 mp3 songs with different styles in all the tests. In both groups, each driver was required to finish three kinds of tests in the same road and traffic conditions: (i) Not listening music; (ii) Listening to music via the smartphone’s (Huawei P7) default music player under shuffle play mode; (iii) Listening to music recommended by SAfeDJ. Each test lasted for 20 minutes, and the time interval between two tests was about 30 minutes for every driver, so as to ensure that the drivers could drive in the same fatigue and mood status as far as possible in each test. Also, in case the degree of familiarity to the roads may impact a driver’s mood-fatigue status, the sequence of the three tests for each driver was randomized.

As shown in Figure C1(b), when a driver started using SAfeDJ, he could use SAfeDJ to log into his Facebook account, which enables SAfeDJ to get his SC like gender, age and cultural background for the initial prediction of his music preference. After all the tests were finished, we calculated and compared the average fatigue degree and negative mood degree (sad plus angry degrees) of each group’s 24 drivers in the three different kinds of tests.

The comparison results about drivers’ fatigue degrees in \textit{group a} and \textit{group b} are summarized in Figure C2(a) and Figure C2(b), respectively. We observe that in both groups,
the drivers’ fatigue degree is slowly increasing in their driving when they do not listen to music or listen to music via the smartphone’s default music player. Also, it is interesting that in group b, the drivers’ recorded fatigue degree is even higher in most of the time periods when they listen to music through the default music player (average value is 0.4435) comparing to not listen to music while driving (average value is 0.3606). While in group a the average value of fatigue degree is 0.3166 when using default music player and 0.3607 when non-listening music. In contrast, the drivers’ fatigue degree is the lowest when they listen to music recommended by SAfeDJ during driving, and the related average value is only 0.1898 in group a and 0.1972 in group b, which means it is about half lower than using default music player in overall.

On the other hand, from Figure C3, we find that the drivers’ recorded negative mood degree is relatively complicated. In group a, the drivers’ negative mood degree is close between using default music player and SAfeDJ. On the other hand, in group b, the drivers’ negative mood degree may become the highest when they listen to music via the default music player in some time periods, but lower than that from listening to music recommended by SAfeDJ in a few time periods. Nevertheless, in general the recorded negative mood degree of
drivers is still the lowest when they listen to music provided by SAfeDJ with an average value 0.0712 in group a and 0.0752 in group b. In contrast, the average value is 0.0961 in group a and 0.1339 in group b when using the default music player. Thus, it means that SAfeDJ decreases drivers’ negative mood degree by around one third compared with the smartphone’s default music player overall. In addition, the average value is 0.2183 in group a and 0.2104 in group b when the drivers are not listening to music while driving.

![Figure 5.7: Comparisons of drivers’ negative mood degree](image)

In addition, as the tests of group (a) and group (b) are in different time periods by different group of people, thus the changing trend of fatigue degree and negative mood degree of drivers are slightly different even in the case non-listening music in these two groups. Due to the experimental constrains, such as currently there is no baseline to validate the accuracy of the mood-fatigue detector (see Appendix B) we used in the case study, and only 24 people are involved in each group which may not sufficient enough, these results by no means to verify the effectiveness of SAfeDJ thoroughly. The main intention here is to demonstrate the SAfeDJ application built on our crowdsensing platform can help to deliver preferable music to drivers while driving in some common situations.